Roman Wyrzykowski
Jack Dongarra
Marcin Paprzycki
Jerzy Waśniewski (Eds.)

LNCS 3019

Parallel Processing and Applied Mathematics

5th International Conference, PPAM 2003
Częstochowa, Poland, September 2003
Revised Papers

Springer

Lecture Notes in Computer Science 3019

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board:

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Friedemann Mattern
ETH Zurich, Switzerland
John C. Mitchell
Stanford University, CA, USA
Oscar Nierstrasz
University of Bern, Switzerland
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
Dortmund University, Germany
Demetri Terzopoulos
New York University, NY, USA
Doug Tygar
University of California at Berkeley, CA, USA
Moshe Y. Vardi
Rice University, Houston, TX, USA

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Roman Wyrzykowski Jack Dongarra
Marcin Paprzycki Jerzy Waśniewski (Eds.)

Parallel Processing and Applied Mathematics

5th International Conference, PPAM 2003
Częstochowa, Poland, September 7-10, 2003
Revised Papers

Springer

eBook ISBN: 3-540-24669-X
Print ISBN: 3-540-21946-3

©2005 Springer Science + Business Media, Inc.

Print ©2004 Springer-Verlag
Berlin Heidelberg

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Springer's eBookstore at: http://ebooks.springerlink.com
and the Springer Global Website Online at: http://www.springeronline.com

Preface

It is our pleasure to provide you with the volume containing the proceedings of the 5th International Conference on Parallel Processing and Applied Mathematics, which was held in Częstochowa, a Polish city famous for its Jasna Gora Monastery, on September 7–10, 2003. The first PPAM conference was held in 1994 and was organized by the Institute of Mathematics and Computer Science of the Częstochowa University of Technology in its hometown. The main idea behind the event was to provide a forum for researchers involved in applied and computational mathematics and parallel computing to exchange ideas in a relaxed atmosphere. Conference organizers hoped that this arrangement would result in cross-pollination and lead to successful research collaborations. In addition, they hoped that the initially mostly Polish conference would grow into an international event. The fact that these assumptions were correct was proven by the growth of the event. While the first conference consisted of 41 presentations, the conference reached 150 participants in Nałęczów in 2001. In this way the PPAM conference has become one of the premiere Polish conferences, and definitely the most important one in the area of parallel/distributed computing and applied mathematics.

This year's meeting gathered almost 200 participants from 32 countries. A strict refereeing process resulted in the acceptance of approximately 150 contributed presentations, while the rejection rate was approximately 33%. Regular tracks of the conference covered such important fields of parallel/distributed computing and applied mathematics as

- parallel and distributed architectures
- scheduling and load balancing
- performance analysis and prediction
- parallel and distributed non-numerical algorithms
- parallel and distributed programming
- tools and environments for parallel and distributed processing
- numerical and non-numerical applications of parallel and distributed computing
- evolutionary and soft computing
- data and knowledge management
- mathematical and computer methods in mechanics and material processing, biology and medicine, physics and chemistry, environmental modeling and seismology, and financial engineering

The plenary and invited talks were presented by S.B. Baden, C. Kesselman, B. Chapman, J. Dongarra, S. Gorlatch, J. Kaufman, M. Livny, T. Stricker, V. Sunderam, B.K. Szymański, D. Trystram, J. Węglarz, and Z. Zlatev.

Important and integral parts of the PPAM 2003 conference were the workshops and special sessions:

- Application Grid Workshop – AGW 2003
- International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks – HeteroPar 2003
- Workshop on High Perfomance Numerical Algorithms
- Workshop on Large Scale Scientific Computations
- Special Session on Parallel and Distributed Bioinformatic Applications

Finally, the meeting began with two half-day tutorials

- Globus Toolkit version 3, by Ewa Deelman
- Load distribution strategies for HPC applications on clusters of symmetric multiprocessors, by Peter Luksch

that proved to be very successful, attracting a large number of participants.

We would like to express our gratitude to our sponsors: Intel Corp., IBM Corp., Optimus S.A., Solidex S.A. We would also like to say thank you to all members of the International Program Committee who worked diligently refereeing the submissions. Finally, we would like to thank all of the local organizers, including the Ph.D. students from the Częstochowa University of Technology, who helped us run the event very smoothly.

We hope that this volume will be useful to you. We would also like everyone who reads it to feel invited to the next conference, which will take place in Poland in 2005.

January 2004

Roman Wyrzykowski
Jack Dongarra
Marcin Paprzycki
Jerzy Waśniewski

Table of Contents

Parallel and Distributed Architectures

Session Guarantees to Achieve PRAM Consistency of Replicated Shared Objects 1
Jerzy Brzezinski, Cezary Sobaniec, Dariusz Wawrzyniak

An Extended Atomic Consistency Protocol for Recoverable DSM Systems 9
Jerzy Brzezinski, Michal Szychowiak

Hyper-Threading Technology Speeds Clusters 17
Kazimierz Wackowski, Pawel Gepner

Building Autonomic Clusters: A Response to IBM's Autonomic Computing Challenge 27
Andrzej Goscinski, Jackie Silcock, Michael Hobbs

Configurable Microprocessor Array for DSP Applications 36
Oleg Maslennikow, Juri Shevtshenko, Anatoli Sergyienko

On Generalized Moore Digraphs 42
Michael Sampels

RDMA Communication Based on Rotating Buffers for Efficient Parallel Fine-Grain Computations 50
Adam Smyk, Marek Tudruj

Communication on the Fly in Dynamic SMP Clusters – Towards Efficient Fine Grain Numerical Computations 59
Marek Tudruj, Lukasz Masko

Wait-Free Publish/Subscribe Using Atomic Registers 69
Pradeep Varma

Scheduling and Load Balancing

Accelerated Diffusion Algorithms on General Dynamic Networks 77
Jacques Bahi, Raphaël Couturier, Flavien Vernier

Suitability of Load Scheduling Algorithms to Workload Characteristics 83
Eunmi Choi, Dugki Min

Minimizing Time-Dependent Total Completion Time
on Parallel Identical Machines 89
Stanisław Gawiejnowicz, Wiesław Kurc, Lidia Pankowska

Diffusion Based Scheduling in the Agent-Oriented Computing System ... 97
Marek Grochowski, Robert Schaefer, Piotr Uhruski

Approximation Algorithms for Scheduling Jobs
with Chain Precedence Constraints 105
Klaus Jansen, Roberto Solis-Oba

Combining Vector Quantization and Ant-Colony Algorithm
for Mesh-Partitioning 113
Jurij Šilc, Peter Korošec, Borut Robič

Wavelet-Neuronal Resource Load Prediction
for Multiprocessor Environment 119
Paweł Hajto, Marcin Skrzypek

Fault-Tolerant Scheduling in Distributed Real-Time Systems 125
Nguyen Duc Thai

Online Scheduling of Multiprocessor Jobs
with Idle Regulation 131
Andrei Tchernykh, Denis Trystram

Performance Analysis and Prediction

Predicting the Response Time of a New Task on a Beowulf Cluster 145
Marta Beltrán, Jose L. Bosque

Space Decomposition Solvers and Their Performance
in PC-Based Parallel Computing Environments 153
Radim Blaheta, Ondřej Jakl, Jiří Starý

Evaluation of Execution Time of Mathematical Library Functions
Based on Historical Performance Information 161
Maciej Brzezniak, Norbert Meyer

Empirical Modelling of Parallel Linear Algebra Routines 169
Javier Cuenca, Luis-Pedro García, Domingo Giménez, José González, Antonio Vidal

Efficiency of Divisible Load Processing 175
Maciej Drozdowski, Łukasz Wielebski

Gray Box Based Data Access Time Estimation for Tertiary Storage
in Grid Environment 181
Darin Nikolow, Renata Słota, Jacek Kitowski

Performance Modeling of Parallel FEM Computations on Clusters 189
Tomasz Olas, Roman Wyrzykowski, Adam Tomas, Konrad Karczewski

Asymptotical Behaviour of the Communication Complexity of One Parallel Algorithm 201
Pavol Purcz

Analytical Modeling of Optimized Sparse Linear Code 207
Pavel Tvrdík, Ivan Šimeček

Parallel and Distributed Non-numerical Algorithms

A BSP Parallel Model for the Göttfert Algorithm over F_2 217
Fatima Abu Salem

Parallelizing the Unsupervised k-Windows Clustering Algorithm 225
Panagiotis D. Alevizos, Dimitris K. Tasoulis, Michael N. Vrahatis

Parallel Simulated Annealing for Bicriterion Optimization Problems 233
Piotr Czarnas, Zbigniew J. Czech, Przemysław Gocyła

Data Decomposition for Parallel K-means Clustering 241
Attila Gursoy

On Generation of Permutations through Suffix/Prefix Reversing in a Cellular Network 249
Zbigniew Kokosiński

A Parallel Dynamic Programming Algorithm for Unranking t–ary Trees 255
Zbigniew Kokosiński

Adaptive Pareto Differential Evolution and Its Parallelization 261
Daniela Zaharie, Dana Petcu

Parallel and Distributed Programming

Global Predicates for Online Control of Distributed Applications 269
Janusz Borkowski

A Thread Partitioning Algorithm for Data Locality Improvement 278
Alexander Chernov, Andrey Belevantsev, Oleg Malikov

Parallel Branch-and-Bound Skeletons: Message Passing and Shared Memory Implementations 286
Isabel Dorta, Coromoto Leon, Casiano Rodriguez

Selfverifying Solvers for Linear Systems of Equations in C-XSC 292
Carlos Amaral Hölbig, Paulo Sérgio Morandi Júnior, Bernardo Frederes Krämer Alcalde, Tiarajú Asmuz Diverio

Process Decomposition via Synchronization Events and Its Application to Counter-Process Decomposition 298
Susumu Kiyamura, Yoshiaki Takata, Hiroyuki Seki

Exception Handling Model with Influence Factors for Distributed Systems 306
Paweł L. Kaczmarek, Henryk Krawczyk

Program Structuring Heuristics for Parallel Systems Based on Multiple Crossbar Switches 314
Eryk Laskowski

Automatic Generation of Optimized Parallel Codes for N-body Simulations 323
David E. Singh, María J. Martín, Francisco F. Rivera

Tools and Environments for Parallel and Distributed Processing

Monitoring Threaded Application with Thread-Enabled OMIS Monitor 331
Bartosz Baliś, Marian Bubak, Włodzimierz Funika, Roland Wismüller, Grzegorz Kaplita

Parallel Program Design Tool with Application Control Methods Based on Global States 338
Janusz Borkowski, Marek Tudruj, Damian Kopanski

Event Handling in the J-OCM Monitoring System 344
Marian Bubak, Włodzimierz Funika, Marcin Smętek, Zbigniew Kiliański, Roland Wismüller

Request Processing in the Java-Oriented OMIS Compliant Monitoring System 352
Marian Bubak, Włodzimierz Funika, Marcin Smętek, Zbigniew Kiliański, Roland Wismüller

Architecture and Implementation of Distributed Data Storage Using Web Services, CORBA and PVM 360
Pawel Czarnul

Online Visualization of OpenMP Programs in the DeWiz Environment 368
Rene Kobler, Dieter Kranzlmüller, Jens Volkert

Cluster Monitoring and Management in the WebCI Environment 375
Tomasz Kuczynski, Roman Wyrzykowski, Grzegorz Studzinski

Fine-Grained System-Call Scheduling in CEFOS on Commodity Processors 383
Shigeru Kusakabe, Kentaro Iio, Hideo Taniguchi, Makoto Amamiya

Dynamic Process Communication in the GDE Environment 389
Jan Kwiatkowski, Daniel Abrich

A Lightweight Message Logging Scheme for Fault Tolerant MPI 397
Inseon Lee, Heon Y. Yeom, Taesoon Park, Hyoungwoo Park

Improving the Performances of a Distributed NFS Implementation 405
Pierre Lombard, Yves Denneulin, Olivier Valentin, Adrien Lebre

Testability of Distributed Objects 413
Magdalena Sławińska

Dynamically Reconfigurable Scientific Computing on Large-Scale Heterogeneous Grids 419
Boleslaw Szymanski, Carlos Varela, John Cummings, Jim Napolitano

Applications of Parallel and Distributed Computing

Parallelization of Large Scale Adaptive Finite Element Computations 431
Krzysztof Banaś

A Multi-agent System Based on the Information Metabolism Theory 439
Andrzej Bielecki, Dominika Nowak

Application of Cellular Automata for Cryptography 447
Pascal Bouvry, Franciszek Seredyński, Albert Y. Zomaya

A Monte Carlo Study of Continuous Non-Ising Phase Transitions in the 3D Ashkin-Teller Model Using the OpenMosix Cluster of Linux PCs 455
Lech Dębski, Grzegorz Musiał, Jos Rogiers

Parallelization of the QC-Lib Quantum Computer Simulator Library 461
Ian Glendinning, Bernhard Ömer

Parallel Simulation of Czochralski Crystal Growth 469
Denis Lukanin, Vladimir Kalaev, Alexander Zhmakin

Application of Parallel Computing in the Transfer-Matrix Simulations of the Supramolecular Rings 475
Ryszard Matysiak, Monika Haglauer, Grzegorz Kamieniarz, Alvaro Caramico D'Auria, Filippo Esposito

Hierarchical Communication for the Parallel Simulations in the Distributed Environment 481
Rafał Metkowski, Piotr Bała

Stepwise Development of Distributed Interactive Simulation Systems 489
Tomasz Orłowski, Bogdan Wiszniewski

Some Aspects of Parallel Performance of a Seismic Ray Analysis Algorithm 497
Marcin Paprzycki, Boris Digas, John Kopsky

Fish Schools: PDES Simulation and Real Time 3D Animation 505
Remo Suppi, Daniel Fernández, Emilio Luque

Consuming Environment with Transportation Network Modelled Using Graph of Cellular Automata 513
Pawel Topa, Witold Dzwinel

Parallelizing Flood Model for Linux Clusters with MPI 521
Viet D. Tran, Ladislav Hluchy, Dave Froehlich, William Castaings

High Frequency Electromagnetic Fields Analysis with the Use of the Parallel FDTD Method 528
Wojciech Walendziuk, Jaroslaw Forenc, Andrzej Jordan

Evolutionary Computing with Applications

Genetic Clustering as a Parallel Algorithm for Approximating Basins of Attraction 536
Katarzyna Adamska

Multiple-Deme Parallel Estimation of Distribution Algorithms: Basic Framework and Application 544
Chang Wook Ahn, David E. Goldberg, R.S. Ramakrishna

A Memory-Efficient Elitist Genetic Algorithm 552
Chang Wook Ahn, Ki Pyo Kim, R.S. Ramakrishna

Augmented Compact Genetic Algorithm 560
Chang Wook Ahn, R.S. Ramakrishna

Parallel Genetic Algorithm for the Flow Shop Scheduling Problem 566
Wojciech Bożejko, Mieczysław Wodecki

Optimization of Structures Using Distributed and Parallel Evolutionary Algorithms 572
Tadeusz Burczynski, Waclaw Kus

A Parallel Evolutionary Algorithm for Discovery of Decision Rules 580
Wojciech Kwedlo

An Evolutionary Programming Algorithm for Automatic Engineering Design 586
Andrew Lewis, David Abramson, Tom Peachey

Weighted Vector Directional Filters Optimized by Genetic Algorithms ... 595
Rastislav Lukac, Bogdan Smolka, Andrzej Swierniak, Konstantinos N. Plataniotis, Anastasios N. Venetsanopoulos

Soft Computing

Systolic Architectures for Soft Computing Algorithms 601
Jarosław Bilski, Jacek Smolęg, Jacek Żurada

Image Compression Based on Soft Computing Techniques 609
Robert Cierniak

A Flexible Connectionist Fuzzy System 618
Krzysztof Cpałka

Recursive Probabilistic Neural Networks 626
Marcin Korytkowski, Marcin Gabryel, Adam Gaweda

Neuro-Fuzzy versus Non-parametric Approach to System Modeling and Classification 632
Robert Nowicki

On Designing of Neuro-Fuzzy Systems 641
Robert Nowicki, Agata Pokropińska, Yoichi Hayashi

Multi-expert Systems 650
Danuta Rutkowska

New Methods for Uncertainty Representations in Neuro-Fuzzy Systems 659
Rafał Scherer, Janusz Starczewski, Adam Gawęda

Interval Comparison Based on Dempster-Shafer Theory of Evidence 668
Pawel Sevastjanow

Data and Knowledge Management

Distributed Spatial Data Warehouse 676
Marcin Gorawski, Rafal Malczok

Improving Load Balance and Fault Tolerance for PC Cluster-Based Parallel Information Retrieval 682
Jaeho Kang, Hyunju Ahn, Sung- Won Jung, Kwang Ryel Ryu, Hyuk-Chul Kwon, Sang-Hwa Chung

An Efficient Conflict Detection Method for Maintaining Consistency of Mobile Database System 688
Sung-Hee Kim, Jae-Dong Lee, Jae-Hong Kim, Hae- Young Bae

Distributed Knowledge Management Based on Software Agents and Ontology 694
Michal Laclavik, Zoltan Balogh, Ladislav Hluchy, Renata Słota, Krzysztof Krawczyk, Mariusz Dziewierz

Ontology Assisted Access to Document Repositories in Public Sector Organizations 700
Renata Słota, Marta Majewska, Mariusz Dziewierz, Krzysztof Krawczyk, Michal Laclavik, Zoltan Balogh, Ladislav Hluchy, Jacek Kitowski, Simon Lambert

Numerical Methods and Their Applications

Simulations of Granular Cohesion Dynamics on Rough Surfaces of Contacting Particles 706
Jacek S. Leszczynski

Adaptive Noise Reduction in Microarray Images Based on the Center-Weighted Vector Medians 714
Rastislav Lukac, Bogdan Smolka, Andrzej Swierniak, Konstantinos N. Plataniotis, Anastasios N. Venetsanopoulos

Implementation Aspects of a Recovery-Based Error Estimator in Finite Element Analysis 722
Arkadiusz Nagórka, Norbert Sczygiol

Optimization Using Nimrod/O and Its Application to Robust Mechanical Design 730
Tom Peachey, David Abramson, Andrew Lewis, Donny Kurniawan, Rhys Jones

Object Oriented Implementation of Modelling Bi-phase Gas-Particle Flows 738
Roman Wyrzykowski, Sebastian Pluta, Jacek Leszczynski

Multi-dimensional Systems – Applications and Computations

Mathematical Linguistics Model for Medical Diagnostics of Organ of Hearing in Neonates 746
Mariusz Flasiński, Elżbieta Reroń, Janusz Jurek, Piotr Wójtowicz, Krzysztof Atłasiewicz

Parallelization in an Algorithm of Multi-dimensional Nonconvex Optimization: An Application to Insurance Network Design 754
Arkady Kryazhimskiy, Vyacheslav Maksimov

Discrimination between Models of Distributed Parameter Systems Using T-optimum Experimental Design 762
Bartosz Kuczewski, Maciej Patan, Dariusz Uciński

Robust Activation Strategy of Scanning Sensors via Sequential Design in Parameter Estimation of Distributed Systems 770
Maciej Patan, Dariusz Uciński

Application Grid Workshop

Security in the OCM-G Grid Application Monitoring System 779
Bartosz Baliś, Marian Bubak, Wojciech Rząsa, Tomasz Szepieniec, Roland Wismüller

Mediators in the Architecture of Grid Information Systems 788
Peter Brezany, A. Min Tjoa, Helmut Wanek, Alexander Wöhrer

Towards the Wisdom Grid: Goals and Architecture 796
Ivan Janciak, Peter Brezany, A. Min Tjoa

Automatic Flow Building for Component Grid Applications 804
Marian Bubak, Kamil Górka, Tomasz Gubała, Maciej Malawski, Katarzyna Zając

Grid Architecture for Interactive Applications 812
Marian Bubak, Maciej Malawski, Katarzyna Zając

Pegasus and the Pulsar Search: From Metadata to Execution on the Grid 821
Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Scott Koranda, Albert Lazzarini, Gaurang Mehta, Maria Alessandra Papa, Karan Vahi

Flood Forecasting in a Grid Computing Environment 831
Ladislav Hluchý, Ján Astaloš, Miroslav Dobrucký, Ondrej Habala, Branislav Šimo, Viet D. Tran

Adaptive Grid Scheduling of a High-Throughput Bioinformatics Application 840
Eduardo Huedo, Rubén S. Montero, Ignacio M. Llorente

Advanced Replica Management with Reptor 848
Peter Kunszt, Erwin Laure, Heinz Stockinger, Kurt Stockinger

SuperVise: Using Grid Tools to Simplify Visualization 856
James Osborne, Helen Wright

Metrics for Grid Applicability: A Distributed Elliptic Curve Platform Assessment 864
Paulo Trezentos, Arlindo L. Oliveira

Execution and Migration Management of HLA-Based Interactive Simulations on the Grid 872
Katarzyna Zając, Marian Bubak, Maciej Malawski, Peter Sloot

HeteroPar'03

Asymptotically Optimal Algorithm for Laplace Task Graphs on Heterogeneous Platforms 880
Olivier Beaumont, Pierre Ramet, Jean Roman

Dynamic Tasks Assignment for Real Heterogeneous Clusters 888
Marta Beltrán, Antonio Guzmán, Jose L. Bosque

Messages Scheduling for Data Redistribution between Clusters 896
Johanne Cohen, Emmanuel Jeannot, Nicolas Padoy

Multidimensional Static Block Data Decomposition for Heterogeneous Clusters 907
Alexey Kalinov, Sergey Klimov

A Job Scheduling Strategy for Heterogeneous Multiprogrammed Systems 915
Piyush Maheshwari

Classification of Partitioning Problems for Networks of Heterogeneous Computers 921
Alexey Lastovetsky, Ravi Reddy

Load-Balancing Iterative Computations on Heterogeneous Clusters with Shared Communication Links 930
Arnaud Legrand, Hélène Renard, Yves Robert, Frederic Vivien

Large Scale Peer to Peer Performance Evaluations, with Gauss-Jordan Method as an Example 938
Serge G. Petiton, Lamine M. Aouad

Anticipating Performance Information of Newly Portable Computers on the WLAN for Load Balancing 946
David Sánchez, Elsa M. Macías, Álvaro Suárez

Performance Study of Scheduling Mechanisms for Peer-to-Peer Computing Environments 954
Ilias Savvas, Tahar Kechadi

Workshop on High Perfomance Numerical Algorithms

Analyzing the Efficiency of Block-Cyclic Checkerboard Partitioning in Neville Elimination 963
Policarpo Abascal, Pedro Alonso, Raquel Cortina, Irene Díaz, José Ranilla

Parallel Algorithms for the Solution of Toeplitz Systems of Linear Equations 969
Pedro Alonso, José M. Badía, Antonio M. Vidal

An Embedded Iterative Scheme in Electromagnetism 977
Bruno Carpentieri, Iain S. Duff, Luc Giraud, Guillaume Sylvand

Cholesky Factorization of Matrices in Parallel and Ranking of Graphs 985
Dariusz Dereniowski, Marek Kubale

Parallel Square Modular Computer Algebra 993
Sergey A. Inutin

Semi-systolic Architecture for AB^2 Operation over $GF(2^m)$ 998
Hyun-Sung Kim, Il-Soo Jeon, Jin-Ho Lee

A Class of Block Smoothers for Multigrid Solution of Saddle Point Problems with Application to Fluid Flow 1006
Piotr Krzyżanowski

Parallelizable Password-Authenticated Key Exchange Protocol 1014
Sung- Woon Lee, Kee- Young Yoo

GRIBB – Branch-and-Bound Methods on the Internet 1020
Randi Moe

Parallel Modular Multiplication Algorithm in Residue Number System 1028
Hyun-Sung Kim, Hee-Joo Park, Sung-Ho Hwang

A Combined Fractional Step Domain Decomposition Method for the Numerical Integration of Parabolic Problems 1034
Laura Portero, Blanca Bujanda, Juan Carlos Jorge

Incomplete Cholesky Factorization in Fixed Memory 1042
Sergey Saukh

A Multigrid Poisson Solver on General 3-Dimensional Domains 1052
Marjan Šterk, Roman Trobec

Solving Linear Recurrence Systems
Using Level 2 and 3 BLAS Routines 1059
Przemysław Stpiczyński

Workshop on Large Scale Scientific Computations

Accelerating Optimization of Input Parameters
in Wildland Fire Simulation 1067
Baker Abdalhaq, Ana Cortés, Tomàs Margalef, Emilio Luque

A Tool to Execute ASSIST Applications on Globus-Based Grids 1075
Ranieri Baraglia, Domenico Laforenza, Nicola Tonellotto

Adaptive Computation over Dynamic and Heterogeneous Networks..... 1083
Kaoutar El Maghraoui, Joseph E. Flaherty,
Boleslaw K. Szymanski, James D. Teresco, Carlos Varela

Deterministic Large-Scale Simulations of the Low-Dimensional
Magnetic Spin Systems 1091
Grzegorz Kamieniarz, Ryszard Matysiak

Distributed File System for Clusters and Grids 1099
Olivier Valentin, Pierre Lombard, Adrien Lebre,
Christian Guinet, Yves Denneulin

New Adaptive GMRES(m) Method
with Choosing Suitable Restart Cycle m 1105
Kentaro Moriya, Takashi Nodera

The Non-blocking Programming Paradigm
in Large Scale Scientific Computations 1114
Philippas Tsigas, Yi Zhang

Comprehensive Air Pollution Studies with the Unified Danish
Eulerian Model 1125
Zahari Zlatev

Special Session on Parallel and Distributed Bioinformatic Applications

Parallel Algorithms for Evolutionary History Reconstruction 1138
Jacek Błażewicz, Piotr Formanowicz, Paweł Kędziora,
Paweł Wojciechowski

A Hierarchical Model of Parallel Genetic Programming Applied to Bioinformatic Problems 1146
Julien Frey, Robin Gras, Patricia Hernandez, Ron Appel

A Fault-Tolerant Protocol for Resource Allocation in a Grid Dedicated to Genomic Applications 1154
Michel Hurfin, Jean-Pierre Le Narzul, Julien Pley, Philippe Raïpin Parvédy

Parallel Stochastic Search for Protein Secondary Structure Prediction 1162
Victor Robles, María S. Pérez, Vanessa Herves, José M. Peña, Pedro Larrañaga

Author Index 1171

This page intentionally left blank

Session Guarantees to Achieve PRAM Consistency of Replicated Shared Objects*

Jerzy Brzezinski, Cezary Sobaniec, and Dariusz Wawrzyniak

Institute of Computing Science
Poznan University of Technology
Piotrowo 3a, 60-965 Poznan, POLAND

Abstract. In this paper we discuss relationships between client-centric consistency models (known as session guarantees), and data-centric consistency models. Appropriate models are formally defined in the context of replicated shared objects using a consistent notation. We prove that PRAM consistency model may be achieved as a proper combination of different session guarantees.

Keywords: Replicated objects, consistency models, session guarantees

1 Introduction

Replication is a common way to improve data availability in a distributed system. However, replicas introduce a side-effect in a form of data inconsistency. Replication management aimed at keeping replicas consistent is the purpose of *consistency protocol.* Guarantees provided by the consistency protocol are expressed by *consistency model.*

There are several consistency models defined mainly as a result of research in the area of *Distributed Shared Memory* (DSM), e.g.: *atomic consistency* [8], *sequential consistency* [6], *causal consistency* [2], *processor consistency* [4,1], *PRAM consistency* [7]. In the context of client-server applications these models are termed *data-centric* [9], because they specify the restriction imposed on the order in which data is updated on individual servers.

Client-centric consistency models in contrast specify the requirements concerning data consistency that are based only on the history of interaction between individual clients and the system (servers). An important distinction between data- and client-centric consistency models lies in separation of clients and servers. In data-centric models it is assumed that the client is bound to a server and forms a single processing node in the sense that everything known to the server is also known to the client. Client-centric consistency models address the problem of issuing different operations at different servers.

There is a question of the relationships between the consistency models families. In this paper, the relationships between PRAM consistency model and

* This work was supported in part by the State Committee for Scientific Research (KBN), Poland, under grant KBN 7 T11C 036 21

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1–8, 2004.
© Springer-Verlag Berlin Heidelberg 2004

client-centric consistency models are analysed. In fact, two views of PRAM consistency are differentiated: client view (the consistency at the client side) and server view (the consistency at the server side), resulting from the system model which allows clients to switch between servers. The relationships are shown and formally proven for both views.

2 Client-Server Interaction via Replicated Objects

Objects are instances of abstract data types that encapsulate some private instance variables accessed by public methods. Operation on an object means execution of one of its methods. An operation is executed as a result of method invocation issued by a client. Every client is a sequential process in the sense that it invokes a method after the execution of the previously invoked method is completed. In other words, from the viewpoint of a given client, at most one operation on one object is pending at a time.

Objects exist on servers, so the operations on objects are executed by servers. Consequently, there is an interaction between a server and a client during the operation performance, which proceeds as follows:

1. as a result of invocation the client sends a request message to the server and blocks until results of the execution are received,
2. the server receives the message, executes the requested method and sends back a response with the results of execution,
3. the client receives the response and continues the execution of its program (possibly invokes another method).

As for the methods themselves, they consist of a number of *elementary* read and write operations on instance variables (of a simple data type) encapsulated within objects. Thus the execution of a method can be reduced to a sequence of read and write operations. If a client invokes a method, it indirectly issues the execution of some elementary operations that implement the method. These elementary operations are said to be *issued by the client* although the client may not be aware of what operations are included in the implementation.

Operations issued by a given client, say C_i, can be totally ordered. Let O_{C_i} denote the set of operations issued be the client. The issue order of operations is defined as follows.

Definition 1. *Let* $o1$ *and* $o2$ *be operations issued by a client* C_i*, i.e.* $o1 \in O_{C_i}$ *and* $o2 \in O_{C_i}$*. Operation* $o1$ *precedes* $o2$ *in issue order* ($o1 \xrightarrow{C_i} o2$) *if one of the following conditions holds:*

1) $o1$ *and* $o2$ *belong to the implementation of the same method and* $o1$ *precedes* $o2$ *in the program order (*$o1$ *is performed at the server side before* $o2$*), or*
2) $o1$ *and* $o2$ *belong to the implementations of different methods, say* $m1$ *and* $m2$*, respectively, and* $m1$ *is issued by the client* C_i *before* $m2$*.* □

To execute a method, a client may send the request to any server keeping a replica of the referenced object. When the invoked method changes the state of the object, the servers are obliged to interact between one another to achieve a consistent state of object replicas. Consequently, subsequent elementary write operations are to be executed by each server that keeps any replica of the object. It is assumed for simplicity that each server keeps a replica of every object. Therefore, write operations must be performed by each server.

The order in which the elementary operations are executed by the servers or observed by the clients is a question of consistency model. For the sake of formal specification, let OW denote a set of all elementary write operations on shared objects and let O_{S_i} denote a set of operations executed locally by the server S_i as a result of direct invocations of methods by clients. In order to distinguish different replicas of the same instance variable let x_i means a replica of x kept by the server S_i. The operations themselves are denoted as follows:

$w_i(x_j)v$ — write of a value v issued by C_i, to the replica of x kept by S_j,
$r_i(x_j)v$ — read of the replica of x kept by S_j, issued by C_i, returning a value v,
$o_i(x_j)v$ — any operation on the replica of x kept by S_j, issued by C_i.

If something (e.g. the client index, the value) is omitted, it is either not important or evident from the current context.

Definition 2 (Server's view). *For a given server S_i the view of elementary operations on shared objects is the set $OW \cup O_{S_i}$ totally ordered by a relation $\overset{S_i}{\longmapsto}$ that satisfies the following condition, called legality*

$$\forall_{\substack{w(x)v, r(x)v \\ \in OW \cup O_{S_i}}} \left[w(x)v \overset{S_i}{\longmapsto} r(x)v \wedge \nexists_{\substack{w(x)u \\ \in OW \cup O_{S_i}}} \left(w(x)v \overset{S_i}{\longmapsto} w(x)u \overset{S_i}{\longmapsto} r(x)v \right) \right] \quad (1)$$

□

Let us call the relation $\overset{S_i}{\longmapsto}$ *server's view order.*

Because the clients can invoke each method on different servers, it is possible that a server's view of operation execution is different from the client's view. In general, there is no guarantee that an overwritten value cannot be read by a client, because the client can access an object on a server whose state has not been updated yet. Additional guarantees depend on a specific consistency model. Consequently, the definition of client's view does not include the legality condition in the form of the condition (1).

Definition 3 (Client's view). *For a given client, say C_i, the view of elementary operations on shared objects is the set $OW \cup O_{C_i}$ totally ordered by a relation $\overset{C_i}{\longmapsto}$ that satisfies the following condition*[1]*:*

$$\forall_{\substack{w(x)v \in OW \cup O_{C_i} \\ r(x)v \in OW \cup O_{C_i}}} w(x)v \overset{C_i}{\longmapsto} r(x)v \quad (2)$$

□

[1] To simplify the identification of write operations, it is assumed that each write operation defines a unique value of a given instance variable.

Similarly to server's view order, let us call the relation $\overset{C_i}{\rightarrowtail}$ *clients's view order.*

Consistency models impose additional restrictions on the relation according to which operations are ordered in the view of each process. The restrictions are specified in the form of consistency conditions that must be preserved in the view order.

When a client issues a read operation, it expects to get the result before issuing another operation. Consequently, we assume as an axiom the following condition:

$$\underset{r(x)v,o(y)u\in O_{C_i}}{\forall}\left(r(x_j)v \overset{C_i}{\rightarrow} o(y_j)u \Rightarrow r(x)v \overset{S_j}{\rightarrowtail} o(y)u\right) \tag{3}$$

Similarly, for the client's view:

$$\underset{r(x)v,o(y)u\in O_{C_i}}{\forall}\left(r(x)v \overset{C_i}{\rightarrow} o(y)u \Rightarrow r(x)v \overset{C_i}{\rightarrowtail} o(y)u\right) \tag{4}$$

3 Client-Centric Consistency Models

The idea of client-centric consistency models has been proposed in [10], and called session guarantees. A formal specification from the viewpoint of a client process is presented in [3]. Kermarrec et al. [5] analyse the application of these models to the replication of Web documents.

In this section we formally define models from the viewpoint of client-server interaction, assuming that one client process corresponds to one session in terms of the definitions in [10].

Generally, in the client-centric consistency models it is assumed that non-commutative write operations are observed by every server in the same order. We assume in this case that non-commutative writes are the write operations that concern the same variable. This means formally that:

$$\underset{x}{\forall}\left(\underset{S_i}{\forall} w(x)v \overset{S_i}{\rightarrowtail} w(x)u \vee \underset{S_i}{\forall} w(x)u \overset{S_i}{\rightarrowtail} w(x)v\right) \tag{5}$$

Read Your Writes. (RYW) guarantee states that a read operation requested by a client can be executed on the server that has performed all write operations previously issued by the requesting client. This is formally expressed by the following condition:

$$\underset{C_i}{\forall}\underset{S_j}{\forall}\left(w(x)v \overset{C_i}{\rightarrow} r(y_j)u \Rightarrow w(x)v \overset{S_j}{\rightarrowtail} r(y)u\right) \tag{6}$$

Monotonic Writes. (MW) guarantee states that write operations are observed by the servers in issue order. In other words, when a write operation issued by a given client is performed by a server, all write operations issued previously

by the client must have been performed by the server before. Consequently, the consistency condition is formulated as follows:

$$\underset{C_i}{\exists}\ w(x)v \overset{C_i}{\rightarrow} w(y)u \Rightarrow \underset{S_j}{\forall}\ w(x)v \overset{S_j}{\mapsto} w(y)u \tag{7}$$

Writes follow reads. (WFR) guarantee has been also called *session causality* in [3]. This model can be informally defined as follows: a write operation by a client process .on data item y following a previous read operation on x by the same process, is guaranteed to follow write operation on x of the value that was read. In other words, causal precedence established by one client must be preserved also by all servers (in their views).

A formal definition states that:

$$\underset{C_i}{\exists}\ r(x)v \overset{C_i}{\rightarrow} w(y)u \Rightarrow \underset{S_j}{\forall}\ w(x)v \overset{S_j}{\mapsto} w(y)u \tag{8}$$

Monotonic Reads. (MR) requires that read operation can be executed by the server that has executed all write operations whose effect has been already read by the client.

Formally, this requires the following condition to be satisfied:

$$\underset{C_i}{\forall}\ \underset{S_j}{\forall}\left(r(x)v \overset{C_i}{\rightarrow} r(y_j)u \Rightarrow w(x)v \overset{S_j}{\mapsto} r(y)u\right) \tag{9}$$

4 Session Guarantees for Pipelined RAM Consistency

Pipelined RAM (PRAM) [7] consistency is one of the weakest data-centric consistency models. It guarantees that the order in which operations are observed preserves local issue order.

From the viewpoint of client-server application data-centric consistency models can be specified both for the client side and for the server side. The distinction between client side and server side results from the fact that a client process can interact with several servers within its session. If it uses the same server all the time the distinction makes no sense, because the server's view and the client's view are the same.

Data-centric consistency models require legality, which is not the case of client-centric models at the client side. Consequently, the legality condition must also be preserved in client's view, i.e:

$$\underset{\substack{w(x)v,r(x)v\\ \in OW\cup O_{C_i}}}{\forall}\left[w(x)v \overset{C_i}{\mapsto} r(x)v \wedge \underset{\substack{w(x)u\in\\ OW\cup O_{C_i}}}{\nexists}\left(w(x)v \overset{C_i}{\mapsto} w(x)u \overset{C_i}{\mapsto} r(x)v\right)\right] \tag{10}$$

Server side. At the server side, PRAM means that the following condition holds:

$$\underset{S_i}{\forall}\ \underset{o1,o2\in OW\cup O_{S_i}}{\forall}\left(\underset{C_j}{\exists}\ o1 \overset{C_j}{\rightarrow} o2 \Rightarrow o1 \overset{S_i}{\mapsto} o2\right) \tag{11}$$

The relationship between PRAM consistency and client-centric consistency models at the server side is stated by the following theorem:

Theorem 1. *At the server side, PRAM consistency is fulfilled if and only if RYW (6) and MW (7) conditions are preserved.* □

Proof. In order to prove the equality ((6) and (7)) $\Leftrightarrow$ (11) let us decompose it into two implications.

$\Rightarrow$ Let us consider the following three cases of operations:
1) $o1 = r(x_j)v$ — following the condition (3) the implication holds.
2) $o1 = w(x)v$ and $o2 = r(y_j)u$ — following the condition (6) the implication holds.
3) $o1 = w(x)v$ and $o2 = w(y)u$ — following the condition (7) the implication holds.

$\Leftarrow$ It is easy to notice that the definition of PRAM consistency (11) contradicts the following sentences resulting from the negation of (6) and (7):
1) $\underset{C_j}{\exists}\,\underset{S_i}{\exists}\left(w(x)v \overset{C_j}{\rightarrow} r(x_i)u \wedge w(x)v \overset{S_i}{\not\mapsto} w(x)u\right)$
2) $\underset{w1,w2\in OW}{\exists}\left(\underset{C_j}{\exists}\, w1 \overset{C_j}{\rightarrow} w2 \wedge \underset{S_i}{\exists}\, w1 \overset{S_i}{\not\mapsto} w2\right)$

□

Client side. At the client side, assuming that the client's view is legal, the definition of PRAM is very similar to the definition at the server side:

$$\underset{C_i}{\forall}\ \underset{o1,o2\in OW\cup O_{C_i}}{\forall}\left(\underset{C_j}{\exists}\, o1 \overset{C_j}{\rightarrow} o2 \Rightarrow o1 \overset{C_i}{\rightarrowtail} o2\right) \tag{12}$$

Lemma 1. *If RYW (6), MW (7) and MR (9) guarantees hold, client's view preserves local order, i.e. the condition (12) is satisfied.* □

Proof. Let us consider the following three cases of operations observed by C_i:

1) $o1 = r_i(x_k)v$ — following the condition (4) the implication holds.
2) $o1 = w(x)v$ and $o2 = r_i(y_k)u$
   Read operations appear only in the client's view of the issuing processes, so the order has to be preserved only in the view of C_i. Following the condition (6) the implication holds for any k (any replica of y).
3) $o1 = w_j(x)v$ and $o2 = w_j(y)u$
   For the sake of contradiction, let us assume without loss of generality that the operations are issued on the same variable, say x, and are observed by C_i in the reverse order. This is apparent if $r(x_k)u \overset{C_i}{\rightarrow} r(x_l)v$, and forces the view $w_j(x)u \overset{C_i}{\rightarrowtail} r(x_k)u \overset{C_i}{\rightarrowtail} w_j(x)v \overset{C_i}{\rightarrowtail} r(x_l)v$. If $k = l$, then the condition (7) is violated. If $k \neq l$ and the condition (7) is preserved on the server S_l, then the condition (9) is violated, because $w_j(x)v \overset{S_l}{\rightarrowtail} r_i(x)v \overset{S_l}{\rightarrowtail} w_j(x)u$. □

Theorem 2. *If RYW (6), MW (7) and MR (9) guarantees are preserved, client's view satisfies PRAM consistency.* □

Proof. As results from Lemma 1, the guarantees (6), (7) and (9) preserved together are sufficient for the condition (12) to be satisfied in the client's view. Let us prove that the view is also legal, i.e. satisfies the condition (10).

Let us assume by contradiction that the conditions (6), (7) and (9) are preserved and there is a client whose view is not legal. This means that for a given client, say C_i, the following condition holds:

$$\exists_{\substack{w(x)v \in OW \cup O_{C_i} \\ w(x)u \in OW \cup O_{C_i} \\ r(x_j)v \in OW \cup O_{C_i}}} w(x)v \overset{C_i}{\rightarrowtail} w(x)u \overset{C_i}{\rightarrowtail} r(x_j)v \tag{13}$$

Consequently, at the server S_j either $w(x)v \overset{S_j}{\rightarrowtail} r_i(x)v \overset{S_j}{\rightarrowtail} w(x)u$ or $w(x)u \overset{S_j}{\rightarrowtail} w(x)v \overset{S_j}{\rightarrowtail} r_i(x)v$ holds. Let us consider the following four cases:

1) $w(x)v \in O_{C_i}$ and $w(x)u \in O_{C_i}$
   Following Lemma 1 and the fact that both client's view order and issue order are linear, the client's view order corresponds to the issue order of these operations, i.e.: $w(x)v \overset{C_i}{\rightarrow} w(x)u \overset{C_i}{\rightarrow} r(x_j)v$. Consequently, the first case of the S_j's view order violates RYW guarantee (6), and the second case violates MW guarantee (7).
2) $w(x)v \notin O_{C_i}$ and $w(x)u \in O_{C_i}$
   The view order defined by the condition (13) is forced by the following issue order: $r(x_k)v \overset{C_i}{\rightarrow} w(x)u \overset{C_i}{\rightarrow} r(x_j)v$. Consequently, the first case of the S_j's view order violates RYW guarantee (6), and the second case violates the condition (5), because — according to RYW guarantee (6) — $w(x)u \overset{S_j}{\rightarrowtail} r(x)v \Rightarrow w(x)u \overset{S_j}{\rightarrowtail} w(x)v$), while, following the conditions (3) and (6), $w(x)v \overset{S_k}{\rightarrowtail} w(x)u$.
3) $w(x)v \in O_{C_i}$ and $w(x)u \notin O_{C_i}$
   The view order defined by the condition (13) is forced by the following issue order: $w(x)v \overset{C_i}{\rightarrow} r(x_k)u \overset{C_i}{\rightarrow} r(x_j)v$. Consequently, the first case of the S_j's view order violates MR guarantee (9), and the second case violates RYW guarantee (6), because — according to the condition (5) — $w(x)u \overset{S_k}{\rightarrowtail} w(x)v$.
4) $w(x)v \notin O_{C_i}$ and $w(x)u \notin O_{C_i}$
   The view order defined by the condition (13) is forced by the following issue order: $r(x)v \overset{C_i}{\rightarrow} r(x_k)u \overset{C_i}{\rightarrow} r(x_j)v$. Consequently, the first case of the S_j's view order violates MR guarantee (9), and the second case violates the condition (5) because of MR guarantee (9) at S_k, according to which $r(x)v \overset{C_i}{\rightarrow} r(x_k)u \Rightarrow w(x)v \overset{S_k}{\rightarrowtail} r(x)u \Rightarrow w(x)v \overset{S_k}{\rightarrowtail} w(x)u$. □

5 Conclusions

In this paper we have presented formal specifications of client-centric and one data-centric (PRAM) consistency models. Proofs have been conducted claiming that PRAM consistency may be provided as a combination of different session guarantees. This shows that only when the session guarantees are applied together, the system ensures a model from the class of data-centric. We differentiate two views of consistency: client view and server view, resulting from the system model which allows clients to switch between servers. To ensure PRAM at the server side *read your writes* and *monotonic writes* are necessary and sufficient guarantees, while at the client side *monotonic reads* is also required. The difference shows that additional effort is needed to ensure PRAM at the client side even if this model is kept at the server side, which is the consequence of client migration.

References

1. M. Ahamad, R. A. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency (extended abstract). In *Proc. of the 5th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA '93),* pages 251–260, June 1993.
2. M. Ahamad, G. Neiger, P. Kohli, J. E. Burns, and P. W. Hutto. Casual memory: Definitions, implementation and programming. *Distributed Computing,* 9:37–49, 1995.
3. G. Chockler, R. Friedman, and R. Vitenberg. Consistency conditions for a CORBA caching service. *Lecture Notes in Computer Science,* 1914:374–388, 2000.
4. J. R. Goodman. Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group, March 1989.
5. Anne-Marie Kermarrec, Ihor Kuz, Maarten van Steen, and Andrew S. Tanenbaum. A framework for consistent, replicated Web objects. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS),* May 1998. `http://www.cs.vu.nl/pub/papers/globe/icdcs.98.pdf`.
6. L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers,* C-28(9):690–691, September 1979.
7. R. J. Lipton and J. S. Sandberg. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Dept. of Computer Science, Princeton University, September 1988.
8. J. Misra. Axioms for memory access in asynchronous hardware systems. *ACM Transactions on Programming Languages and Systems,* 8(1):142–153, January 1986.
9. A. S. Tanenbaum and M. van Steen. *Distributed Systems – Principles and Paradigms.* Prentice Hall, New Jersey, 2002.
10. Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent W. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems (PDIS 94),* Austin, Texas, September 28–30, 1994, pages 140–149. IEEE Computer Society, 1994.

An Extended Atomic Consistency Protocol for Recoverable DSM Systems*

Jerzy Brzezinski and Michal Szychowiak

Institute of Computing Science
Poznan University of Technology
Piotrowo 3a, 60-965 Poznan, POLAND

Abstract. This paper describes a new checkpoint recovery protocol for Distributed Shared Memory (DSM) systems with read-write objects. It is based on independent checkpointing integrated with a coherence protocol for the atomic consistency model. The protocol offers high availability of shared objects in spite of multiple node and communication failures, introducing little overhead. It ensures fast recovery in case of multiple node failures and enables a DSM system to circumvent the network partitioning, as far as a majority partition can be constituted. A formal proof of correctness of the protocol is also presented.

1 Introduction

Modern Distributed Shared Memory (DSM) systems require high availability of shared data in spite of failures. To meet this requirement one can apply a recovery technique which consists in restoring the values of lost data in such a way that the whole memory remains in a consistent state, according to the consistency model used. The recovery techniques originally developed for general distributed systems, imposed on the DSM ([3],[6],[8], among the latest) require accesses to a stable secondary storage in order to save the current state of the computation (a *checkpoint*), and therefore suffer from significant overhead.

Recently, a new category of checkpoint protocols for the DSM has appeared in literature ([1],[2],[4]). It was originated with the recovery protocol of [7] which stores checkpoints in the local memory of distinct nodes instead of stable storage. In [1] we have proposed an outline of a new *invalidation-based* coherence protocol extended to manage both data access and checkpoint operations. As invalidation-based coherence protocols incur lower overhead of the write operation than the write-update protocols, they seem to be a better solution for object-based DSM systems in which the read-to-write ratio is typically low. Compared to the protocols in [2],[4],[7] we fully exploit the DSM replication mechanism to improve the reliability and reduce the overall overhead of checkpointing and coherence operations. In contrast to all the previous protocols, our solution can overcome multiple node failures and network partitioning. In this paper we give a formal description and the proof of correctness of this protocol.

* This work has been partially supported by the State Committee for Scientific Research grant no. 7T11C 036 21

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 9–16, 2004.
© Springer-Verlag Berlin Heidelberg 2004

This paper is organized as follows. In Section 2 we define the system model. Section 3 presents the extended coherence protocol. The correctness of the extended protocol is analyzed in Section 4. Some concluding remarks are given in Section 5.

2 System Model and Problem Formulation

2.1 DSM System

A DSM system is an asynchronous distributed system composed of a finite set of sequential processes P_1, P_2, ..., P_n that can access a finite set O of shared objects. Each $P_i$is executed on a DSM node n_i composed of a local processor and a volatile local memory used to store shared objects accessed by P_i. Each object consists of several values *(object members)* and *object methods* which read and modify object members (here we adopt the object-oriented approach; however, our work is also applicable to variable-based or page-based shared memory). The concatenation of the values of all members of object $x \in O$ is referred to as *object value* of x. We consider only read-write objects, i.e. each method of x has been classified either as read-only (if it does not change the value of x, and, in case of nested method invocation, all invoked methods are also read-only) or read-and-modify (otherwise). Read access $r_i(x)$ to object x is issued when process P_i invokes a read-only method of object x. Write access $w_i(x)$ to object x is issued when process P_i invokes any other method of x. Each write access results in a new object value of x. By $r_i(x)v$ we denote that the read operation returns value $v$of x, and by $w_i(x)v$ that the write operation stores value $v$to x. For the sake of simplicity of the presentation we assume that each write access to an object writes a unique value.

To increase the efficiency of DSM, objects are replicated on distinct hosts, allowing concurrent access to the same data. A consistent state of DSM objects replicated on distinct nodes is maintained by a *coherence protocol* and depends on the assumed *consistency model.*

2.2 Atomic Consistency

Informally, the *atomic consistency* model [5] guarantees that any access to a shared object operates on its most recent value in real time. Let H_i denote the set of all access operations to shared objects issued by P_i and H – the set of all operations issued by the system. Let $o^1 \rightarrow_{RT} o^2$ mean that operation o^1 finishes in real time before o^2 starts.

Definition 1. *An execution of access operations is atomically consistent if there exists a total order $\mapsto$of the operations in H preserving real-time ordering, i.e.:*

$$\underset{o^1,\ o^2 \in H}{\forall} o^1 \rightarrow_{RT} o^2 \Rightarrow o^1 \mapsto o^2,$$

and satisfying the following conditions:

AC1 *(legality):* $\forall_{w(x)v,\, r(x)v \in H} \; \nexists_{o(x)u \in H} :: u \neq v \wedge w(x)v \mapsto o(x)u \wedge o(x)u \mapsto w(x)v$

AC2 *(ordered writing):* $\forall_{w^1(x),\, w^2(x) \in H} :: w^1(x) \mapsto w^2(x) \vee w^2(x) \mapsto w^1(x).$

For the above described model we investigate the coherence protocol proposed by Li and Hudak [5], here referred to as original protocol. The original protocol uses a *write-invalidate* schema to guarantee the atomic consistency by allowing only one process to write a given object x (an exclusive replica of the object, in *writable* state – WR); meanwhile the other processes are not allowed to access the object (i.e. all the other existing replicas are *invalidated* – they are in INV state). When no write access to this object is being performed, simultaneous existence of several read-only replicas (i.e. in *read-only* state – RO) of x is possible. The exclusive replica of object x is called *master replica* of x and the process currently holding it is called x's *owner*. A set containing the identities of all processes currently holding a replica of object x is called *copyset* of x, $\mathrm{CS}(x)$.

Briefly, if process P_i issues a read access to object x locally unavailable (i.e. the local replica of x is in INV state), the protocol issues a read request $r_i(x)$ to the current owner of x. The owner adds the identity of P_i to $\mathrm{CS}(x)$ and sends back to P_i a new RO replica of x. If P_i performs a write access to object x locally unavailable, the protocol relays a write request $w_i(x)$ to the object owner. On this request, the owner invalidates all the replicas in $\mathrm{CS}(x)$, clears $\mathrm{CS}(x)$ and then sends a new replica of x along with the ownership of x back to P_i. When arrived at P_i, the replica is set to the WR state and $\mathrm{CS}(x)$ includes only P_i.

Each object has been assigned a *manager* selected from among all processes, which keeps a directory entry that contains the identity of the current object owner. We require the reliability of directory services, to enable correct identification of current object owners at any time.

3 Extended Coherence Protocol for Atomic Consistency

Now we propose an extension of the original protocol [5]. The extension aims to offer a low-cost checkpointing of shared objects and high availability of checkpoints in spite of a failure of multiple nodes. Checkpointing is integrated with normal work of the coherence protocol, so the amount of additional synchronization is reduced.

Object checkpoints are stored in DSM as special-purpose replicas (called *checkpoint replicas*). The identities of DSM nodes holding checkpoint replicas are stored in CCS (*checkpoint copyset*) maintained by object owners. $\mathrm{CCS}(x)$ is initiated at the creation of x and does not include the owner. The content of $\mathrm{CCS}(x)$ can change accordingly to further access requests or failure pattern, or any load balancing mechanisms. Obviously, the size of $\mathrm{CCS}(x)$ influences not only the degree of checkpoint availability (i.e. failure resilience of the system) but also the cost of checkpointing operation (since the update operation is generally much more costly than the invalidate one). A possible solution is to always keep the number nc of checkpoint replicas between boundaries nc_{min} and nc_{max}. Value nc_{min} represents the minimum number of checkpoint replicas necessary to reach

the desired failure resilience. On the other hand, value nc_{max} represents the maximum number of checkpoint replicas allowed, to limit the protocol overhead. In the simplest case we can keep $nc = nc_{min} = nc_{max} = \lfloor \frac{n}{2} \rfloor$, where n is the total number of system nodes. This guarantees that in case of network partitioning there will always be at least one replica of each shared object in the majority partition of the DSM system.

The proposed extended protocol introduces two new replica states, in addition to RO, WR and INV states:

- C (checkpoint) – denoting that a checkpoint replica is used to restore the consistent memory state on recovery. Value of x in state C is not available for neither read nor write operations.
- ROC (read-only checkpoint) – denoting that a checkpoint replica is available for read access to object x. The value of a replica ROC reflects the state of x at the moment of the last checkpoint. As no further write has been performed on this object since then, a ROC replica reflects the current value of x. Therefore, on any read access to x, a ROC replica is used as if it was in RO state. Independently of ROC replicas there can be several RO replicas of x maintained as in the original protocol. All checkpoint replicas are in state ROC directly after the checkpointing operation, until some process issues a write request to x, and the protocol invalidates all replicas from $\mathrm{CS}(x)$ and $\mathrm{CCS}(x)$. The invalidation causes ROC replica to switch into state C.

The checkpointing is performed on each access to a WR replica from another process. When a recently modified value of x has to be transferred from P_i to any P_j, and that value has not been read by any other process than the owner yet and not yet checkpointed (we call this value of x "dirty"), the extended recovery protocol updates all checkpoint replicas of x prior to the transfer (this operation actually takes a new checkpoint of x). The reason is that even if P_i fails after having modified x, these modifications were not yet visible to any other processes (since none of them issued a read request to x), and therefore restoring the value of x from the last checkpoint (taken before the modifications of failed P_i) will preserve the atomic consistency of DSM memory. However, at the moment of checkpointing x, P_i can also own some other object y modified before the last modification of x (i.e. WR replica of y is dirty). Then, if P_i fails after checkpointing x but before checkpointing y, the atomic consistency of the memory will be violated on recovery, since the formerly checkpointed value of y is inconsistent with the recent checkpoint of x. Therefore, on each checkpoint, P_i is required to checkpoint all dirty objects.

The following rules describe actions of the extended protocol in response to access requests on process P_i:

on read request $r_i(x)$:

- if the local replica of x is in state RO, ROC or WR, the read operation is performed instantaneously;
- if the local replica of x is in state INV or C, the request is propagated to the current owner of x, P_k. If the master replica at P_k is in state WR, it is transformed into RO state and, if x is dirty, atomic checkpoint is taken of all dirty objects owned by P_k. The identity of P_i is added to $\mathrm{CS}(x)$ and the protocol updates the local replica of P_i, setting it into RO state.

on write request $w_i(x)$:

- if the local replica of x is in state WR, the modification is performed instantaneously;
- if the local replica of x is in state INV or C, the request is propagated to the current owner of x, P_k. If the master replica of x is dirty, atomic checkpoint is taken of all dirty objects owned by P_k. The replica at P_k is set to state C, the identity of P_k is excluded from $\mathrm{CS}(x)$. If P_i's replica of x was in C state, the identity of P_k is added to $\mathrm{CCS}(x)$ in place of P_i, in order to maintain the cardinality of $\mathrm{CCS}(x)$ constant. Then the protocol transfers the current value of x with $\mathrm{CS}(x)$ and $\mathrm{CCS}(x)$ to P_i, making P_i the current owner of x. The new owner invalidates all the replicas in $\mathrm{CS}(x)$ and $\mathrm{CCS}(x)$ and sets the state of x to WR;
- if the local replica of x is in state RO or ROC, no update is necessary, only the ownership is transferred, with the invalidation of other replicas of x.

3.1 Recovery

As long as the current owner is non-faulty in the majority partition, the extended coherence protocol assures the processing of all requests to x issued in the majority partition. If the current owner becomes unavailable, the recovery procedure elects a new owner from among all processes in $\mathrm{CS}(x)$ available in the primary partition. If no process from $\mathrm{CS}(x)$ is available in the primary partition (e.g. the master replica was in WR state at the moment of its owner's failure), the value of x must be recovered from a checkpoint replica at any process in $\mathrm{CCS}(x)$ available in the primary partition.

The atomic consistency is ensured in the majority partition. Each shared object can always be accessed in the majority partition, i.e. if its current owner becomes unavailable, the new owner will be constituted on a first subsequent access request. However, it is important to note that partitioning may result in temporal simultaneous existence of several master replicas in WR state, i.e. the elected new one in the majority partition and the old one in the minority partition *(dirty master replica).* Therefore, in order to avoid inconsistency, the recovery protocol must invalidate all replicas in minority partitions. This invalidation should be performed in each minority partition at the latest on the first external access to a dirty master replica.

4 Correctness of the Extended Protocol

We decompose the proof of correctness of the extended protocol into proofs of its two properties: safety and liveness. The first property asserts that the protocol correctly maintains the coherency of shared data, accordingly to the atomic consistency model, besides failures of processes and communication links (resulting in at most f processes crashed or separated from the majority partition). The liveness property, in turn, states that each access operation issued to any shared data will eventually be performed (in a finite time), even in the presence of failures.

Let $r_i^t(x)v$ denote a read operation issued by P_i on x, returning value v at time t. Similarly, $w_i^t(x)v$ will denote a write operation, issued by P_i, that changes the value of x to v at time t. We will omit value v if it is irrelevant to the current considerations. Moreover, we will call a process **correct** if it is nonfaulty and belongs to a majority partition at the considered time t.

Definition 2. ***Current*** *value of object x at time t is value v of the most recent (in real time) write operation performed on x until time t by any correct process P_i, i.e. $w_i^{t'}(x)v$, where $t' = max\{t'' :: t'' \leq t \wedge (\exists P_j :: w_j^{t''}(x))\}$.*

Definition 3. ***Current checkpoint*** *of x is a value of x saved in the last (in real time) checkpoint operation of x.*

Definition 1 of the atomic consistency model (Section 2.2) implicitly assumed correctness of all processes and therefore it is not suitable for systems with unreliable components. We extend the consistency model by introducing *reliable operations* performed only by correct processes.

Definition 4. *Access operation on x performed at time t by a correct process P_i is an* ***atomically consistent reliable operation*** *if*

AC1': *it is performed on a current value of x, and*

AC2': *if it is a write operation, no other write operation on the same object can be executed simultaneously in the same partition.*

Lemma 1. *The master replica of x always holds the current value of x.*

Proof. Assume P_i is the owner of x at some time t, and the last write operation performed on x by P_i at time $t' \leq t$ was $w_i^{t'}(x)v$. Since P_i is still the owner, it holds $\exists t'' :: t' \leq t'' \leq t :: w_j^{t''}(x)u$, and therefore v is the current value of x. □

Lemma 2. *If there are RO/ROC replicas of x, each contains the same value v which is the current value of x.*

Proof. Assume again that P_i is the owner of x, and at time t' it performs the last write operation $w_i^{t'}(x)v$. From this moment, the master replica is in state WR, and it will switch into RO state on the first read $r_j^t(x)v$, where $t > t'$ and $j \neq i$, updating checkpoint replicas with value v and switching them into ROC state. All subsequent read operations will eventually update INV replicas with value v in RO state. Until the next write operation $w_k^{t''}(x)u$, v remains the current value of x. Since the $w_k^{t''}(x)u$ operation will invalidate all the RO/ROC replicas before t'', RO/ROC replicas exist only at time $t :: t' < t < t''$, and v is the current value of x at t. □

Lemma 3. *At most one WR replica of x is available in any partition.*

Proof. The invalidation of all available replicas in CS and CCS before switching the master replica into WR state ensures that all other replicas of x in a partition are either in state INV or C but not RO, ROC or WR. □

Lemma 4. *No two write operations on x can be executed simultaneously in a partition.*

Proof. Any write is performed only on a replica in WR state. As in any partition, there are no two replicas of x in WR state (from Lemma 3), no two write operation can be processed at a time in the same partition. □

Theorem 1. *(safety): Every access to object x performed by a correct process is an atomically consistent reliable operation.*

Proof. As the condition AC2' of Definition 4 is satisfied for all correct processes directly from Lemma 4, it is necessary to prove only that AC1' is also satisfied. Remark that the write operation corresponds to an execution of a read-and-modify method of the accessed object, thus both read and write accesses behave identically from the point of view of the condition AC1'. It will be then sufficient to restrict the analysis only to a read access.

Let us analyze a read performed in P_i at moment t.

1. If P_i holds a WR replica, it is the owner of x, and its local replica is the master replica of x. Any subsequent read operation will return v of the master replica, and from Lemma 1 it is the current value of x.
2. If P_i holds a RO or ROC replica of x, any subsequent read operation will return v of the local replica, and from Lemma 2 it is the current value of x.
3. If P_i holds either a INV or C replica of x, the read operation cannot be performed on the local replica, and will require to receive an update v from the owner of x. Thus the access operation will eventually return the value v fetched from the master replica (which is the current value of x), or it will never return a value (if the master replica cannot be accessed). □

Lemma 5. *If there is no partitioning, there always exists a replica of x.*

Proof. By assumption, at most f processes can fail. From the cardinality of $\mathrm{CCS}(x)$: $|\mathrm{CCS}(x)| = \lfloor \frac{n}{2} \rfloor$, if the owner and $f - 1$ processes in $\mathrm{CCS}(x)$ fail at time t, at least one checkpoint replica of x remains. □

Lemma 6. *If a majority partition exists, there is a replica of x available in that partition.*

Proof. By definition, the majority partition is composed of at least $\lceil \frac{n+1}{2} \rceil$ correct processes. Again, from the cardinality of $\mathrm{CCS}(x)$: $|\mathrm{CCS}(x)| = \lfloor \frac{n}{2} \rfloor$. Therefore, the master replica or at least one checkpoint replica of x exists in the majority partition. □

Theorem 2. *(liveness): The protocol eventually brings a value of x to any correct process in the majority partition requesting the access.*

Proof. As the write access is a read-and-modify operation, we can restrict the analysis only to the read access. Let us analyze read access $r_i^t(x)v$ performed by P_i belonging to the majority partition.

1. If P_i holds either a RO, ROC or WR replica of x, then the read is performed instantaneously on the local replica of x.
2. If P_i holds either a INV or C replica of x, then:

- if the owner of x is available at the time t, v is fetched from the master replica of the owner after a finite communication delay
- if the owner of x has failed or is unavailable, then from Lemma 5 and Lemma 6:
  a) either there exist RO/ROC replicas in the system; and at least one with value v is available in the majority partition and can be fetched;
  b) or there exist no RO nor ROC replicas; in this case, the current value of x is the current checkpoint of x, and at least one C replica holding value $v$is available, and the recovery procedure will bring it to the requester. □

5 Conclusions

The coherence protocol proposed in this paper offers an efficient recovery technique for atomic consistency of shared objects and tolerates network partitioning as far as a majority partition exists. Further work should investigate dynamic management of the boundary restriction for CCS, as well as an appropriate method of determining a majority partition. Another open problem is the reliability of the protocol itself, e.g. the protocol should be able to circumvent failures of object managers in a nonblocking way.

References

1. Brzeziński, J., Szychowiak, M.: Replication of Checkpoints in Recoverable DSM Systems. Proc 21^{st} Int'l Conference on Parallel and Distributed Computing and Networks PDCN'2003, Innsbruck (2003)
2. Christodoulopoulou, R., Azimi, R., Bilas, A.: Dynamic data replication: An approach to providing fault-tolerant shared memory clusters. Proc. 9^{th} IEEE Symposium on High-Performance Computer Architecture HPCA9, Anaheim, California (2003)
3. Kongmunvattana, A., Tanchatchawal, S., Tzeng, N.-F.: Coherence-Based Coordinated Checkpointing for Software Distributed Shared Memory Systems. Proc. 20^{th} Conference on Distributed Computing Systems (2000) 556-563
4. Morin, C., Kermarrec, A.-M., Morin, C., Banâtre, M.: An Efficient and Scalable Approach for Implementing Fault Tolerant DSM Architectures. IEEE transactions on Computers, Vol. 49. No. 5. (2000) 414-430
5. Li, K., Hudak, P.: Memory Coherence in Shared Virtual Memory Systems. ACM Transactions on Computer Systems, Vol.24. No.8. (1989) 321-359
6. Park, T., Yeom, H. Y.: A Low Overhead Logging Scheme for Fast Recovery in Distributed Shared Memory Systems. Journal of Supercomputing Vo.15. No.3. (2002) 295-320
7. Stumm, M., Zhou, S.: Fault Tolerant Distributed Shared Memory Algorithms. Proc 2^{nd} IEEE Symposium on Parallel and Distributed Processing, Dallas (1990) 719-724
8. Sultan, F., Nguyen, T. D., Iftode, L.: Scalable Fault-Tolerant Distributed Shared Memory. Proc. Supercomputing SC2000, Dallas (2000) 54-68

Hyper-Threading Technology Speeds Clusters

Kazimierz Wackowski[1] and Pawel Gepner[2]

[1] Warsaw University of Technology
kwak@wship.edu.pl
[2] Intel Corporation
pawel.gepner@intel.com

Abstract. The paper describes a new approach to the use of the Hyper-Threading Technology for improving cluster performance. The Hyper-Threading Technology is enabled in 90% of all sold server's CPUs today and in the most cases it improves the overall application performance. The obvious question would be: Does Hyper-Threading Technology improves cluster performance and if so, how it works? In order to get the fully optimized Hyper-Threading Technology enabled cluster it is necessary to focus on 3 elements which can take advantage of Hyper-Threading Technology and moreover they all need to be tuned for this aim. These elements are: operating system, compiler and application.

1 Introduction

Hyper-Threading (HT) Technology enabled processors contain multiple logical processors per physical processor package. The state information necessary to support each logical processor is replicated, sharing, or partitioning the physical processor resources. The operating system 3Q(OS) makes a single processor behave like two logical processors. When HT is enabled, the OS allows the processor to execute multiple threads simultaneously, in parallel within each processor. The processor resources are typically underutilized by most applications. A CPU with Hyper-Threading Technology enabled can generally improve overall application performance. Multiple threads running in parallel can achieve higher processor utilization and increase throughput. In order to get the fully optimized benefit it is necessary to focus on 3 key areas which need to be aware of Hyper-Threading Technology and tuned for it. These areas are: operating system, compiler and application.

2 Operating System Optimization

The first and fundamental issue when we think about operating system optimization for HT is awareness and ability to run in multi processor environment, meaning support for symmetric multiprocessor (SMP) at the kernel. For clustered implementations the predominant OS in use is Linux so we will focus our attention here. The Linux kernel was HT capable since the release of 2.4.17. The

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 17–26, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2.4.17 kernel recognizes logical processors and behaves like a Hyper-Threaded processor with two physical processors [15]. HT support can be viewed by using the command **cat /proc/cpuinfo** to show the presence of two processors: **processor 0** and **processor 1.**

Typically, each physical processor has a single architectural state on a single processor core to service threads. With HT, each physical processor has two architectural states on a single core, making the physical processor appear as two logical processors to service threads [2]. The BIOS counts each architectural state on the physical processor.

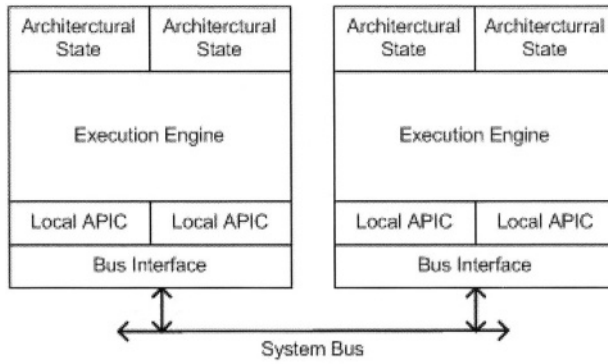


Fig. 1. Hyper-Threading technology on an SMP

Fig.l shows a typical, bus-based SMP scenario on a processor with Hyper-Threading technology. Each logical processor can execute a software thread, allowing a maximum of two software threads to execute simultaneously on one physical processor [11]. Since Hyper-Threading-aware operating systems take advantage of logical processors, those operating systems have twice as many resources to service threads. These replicated resources create copies of the resources for the two executed threads [9]:

- The register alias tables map the architectural registers (eax; ebx; ecx: etc.) to physical rename registers. Since we need to keep track of the architectural state of both logical processors independently, these tables have to be duplicated.
- The Return Stack Predictor has to be duplicated in order to accurately predict call-return instruction pairs.
- The next instruction pointers also needed to be duplicated because each logical processor needs to keep track of its progress through the program it is executing independently. There are two sets of next instruction pointers. One at the trace cache (the Trace Cache Next IP") which is a first-level instruction cache that stores decoded instructions and in the case of a Trace Cache miss another set of next instruction pointers at the fetch and decode logic.

- Some of the front-end buffers are duplicated (Instruction Streaming Buffers and Trace Cache Fill Buffers) to improve instruction prefetch behavior.
- The Instruction TLB was duplicated because it was simpler to duplicate it than to implement the logic to share this structure. Also there was some die area near the instruction TLB that was easy to use.
- In addition, there are also some miscellaneous pointers and control logic that are too small to point out.

Such duplication of resources extends far less than 5% of the total die area. The Xeon processor was the first member of the Hyper-Threading technology enabled CPUs.

To achieve the goal of executing two threads on a single physical processor, the processor simultaneously maintains the context of multiple threads allowing the scheduler to dispatch two potentially independent threads concurrently. The OS schedules and dispatches threads to each logical processor, just as it would in a dual-processor or multi-processor system. When a thread is not dispatched, the associated logical processor is kept idle. When a thread is scheduled and dispatched to a logical processor (#0), the Hyper-Threading technology utilizes the necessary processor resources to execute the thread. When a second thread is scheduled and dispatched on the second logical processor (#1), resources are replicated, divided, or shared as necessary in order to execute the second thread. Each processor makes selections at points in the pipeline to control and process the threads. As each thread finishes, the operating system idles the unused processor, freeing resources for the running CPU. Hyper-Threading technology is supported in Linux kernel 2.4.x, however the scheduler used in the kernel 2.4.x is not able to differentiate between two logical processors and two physical processors [15].

The support for Hyper-Threading in Linux kernel 2.4.x includes the following enhancements:

- 128-byte lock alignment
- Spin-wait loop optimization
- Non-execution based delay loops
- Detection of Hyper-Threading enabled processor and starting the logical processor as if machine was SMP Serialization in MTRR and Microcode Update driver as they affect shared state
- Optimization to scheduler when system is idle to prioritize scheduling on a physical processor before scheduling on logical processor
- Offset user stack to avoid 64K aliasing

All these enhancements can improve system performance in the areas such as scheduler, low-level kernel primitives, the file server, the network, and threaded support by 30%. Also compilation the Linux kernel with a parallel makes (make -j 2, for example) provide significant speedup.

Fig. 2 shows the absolute performance of doing a kernel build on one and two Intel Xeon processor MP with and without Hyper-Threading technology. This application scales nicely from 1 to 2 processors, showing an impressive 1.95

speedup. On a single processor with Hyper-Threading technology, the application has a speedup of 1.20. While the speedup is not close to the dual processor speedup, it goes to show that this technology is really promising [22]. Hyper-Threading technology is able to achieve a significant speedup while keeping the system cost constant, while a dual processor system costs significantly more than a single processor one.

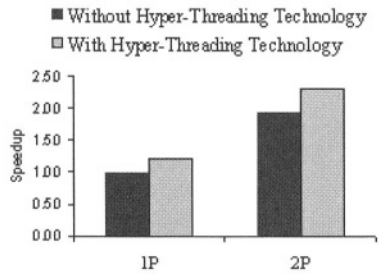


Fig. 2. Linux kernel compile performance

The Linux kernel 2.5.x may provide performance speedup even up to 51%, mainly via improvements to the scheduler. In addition to the optimized scheduler, there are other modifications added to the Linux kernel that increase performance [22]. Those changes are: **HT-aware passive load-balancing:**
The IRQ-driven balancing has to be per-physical-CPU, not per-logical-CPU. Otherwise, it might happen that one physical CPU runs two tasks while another physical CPU runs no task; the stock scheduler does not recognize this condition as "imbalance" because the stock scheduler does not realize that the two logical CPUs belong to the same physical CPU.

"Active" load-balancing:
This is when a logical CPU goes idle and causes a physical CPU imbalance. The imbalance caused by an idle CPU can be solved via the normal load-balancer. In the case of HT, the situation is special because the source physical CPU might have just two tasks running, both runnable. This is a situation that the stock load-balancer is unable to handle, because running tasks are hard to migrate. This migration is essential otherwise a physical CPU can get stuck running two tasks while another physical CPU stays idle.

HT-aware task pickup:
When the scheduler picks a new task, it should prefer all tasks that share the same physical CPU before trying to pull in tasks from other CPUs. The stock scheduler only picks tasks that were scheduled to that particular logical CPU.

HT-aware affinity:
Tasks should attempt to "link" to physical CPUs, not logical CPUs.

HT-aware wakeup:
The stock scheduler only knows about the "current" CPU, it does not know

about any sibling. On HT, if a thread is woken up on a logical CPU that is already executing a task, and if a sibling CPU is idle, then the sibling CPU has to be woken up and has to execute the newly woken-up task immediately.

3 Compiler Optimization

Intel processors have a rich set of performance-enabling features such as the Streaming-SIMD-Extensions (SSE and SSE2) in the IA-32 architecture, large register files, prediction, control and data speculation in the Itanium-based architecture. These features allow the compiler to exploit parallelism at various levels [4].

The Hyper-Threading Technology-enabled processors can significantly increase the performance of application programs with a high degree of parallelism. These potential performance gains are only obtained when an application is efficiently multithreaded, either manually or automatically [2]. The Intel C++/Fortran high-performance compiler supports several such techniques. One of those techniques is automatic loop parallelization.

In addition to automatic loop level parallelization, Intel compilers support OpenMP directive, which significantly increase the domain of various applications amenable to effective parallelism. For example, users can use OpenMP parallel sections to develop an application where section-1 calls an integer-intensive routine and where section-2 calls a floating-point intensive routine. Higher performance is obtained by scheduling section-1 and section-2 onto two different logical processors that share the same physical processor to fully utilize processor resources based on the Hyper-Threading Technology. The OpenMP standard API supports a multi-platform, shared-memory, parallel programming paradigm in C++/C/Fortran95 on all Intel architectures and popular operating systems such as Windows NT*, Linux*, and Unix*. OpenMP directives and programs have emerged as the de facto standard of expressing parallelism in various applications as they substantially simplify the notoriously complex task of writing multithreaded programs.

The Intel compilers support the OpenMP pragmas and directives in the languages C++/C/Fortran95, on IA-32 and IPF architectures. The Intel OpenMP implementation in the compiler strives to generate multithreaded code which gains a speed-up due to Hyper-Threading Technology over optimized uniprocessor code, integrate parallelization tightly with advanced scalar and loop optimizations such as intra-register vectorization and memory optimizations to achieve better cache locality and efficiently exploit multi-level parallelism, and minimize the overhead of data-sharing among threads.

4 Application Optimization

Multi-threaded applications that perform well on SMP systems will generally perform well on Hyper-Threading enabled processors. But do not confuse Hyper-Threading enabled processors with SMP systems. Each processor in an SMP sys-

tem has all its physical processor resources available and will not experience any resource contention at this level. Well-designed multithreaded applications will perform better on SMP systems when running on Hyper-Threading enabled processors. Enterprise and technical computing users have a never-ending need for increased performance and capacity. Performance continues to be a key concern for them [12].

Processor resources, however, are often underutilized and the growing gap between core processor frequency and memory speed causes memory latency to become an increasing performance challenge. Intel's Hyper-Threading Technology brings Simultaneous Multi-Threading to the Intel Architecture and makes a single physical processor appear as two logical processors with duplicated architecture state, but with shared physical execution resources. This allows two tasks (two threads from a single application or two separate applications) to execute in parallel, increasing processor utilization and reducing the performance impact of memory latency by overlapping the latency of one task with the execution of another. Hyper-Threading Technology-capable processors offer significant performance improvements for multi-threaded and multi-tasking workloads without sacrificing compatibility with existing software or single-threaded performance.

The first step in multi-threading applications for Hyper-Threading is to follow the threading methodology for designing Symmetric Multi-Processor (SMP) solutions. The best way of designing for Hyper-Threading enabled processors is to avoid known traps.

There are several known pitfalls that developers can encounter when tuning an application for Hyper-Threading enabled processors. The pitfalls are covered in detail in the "Intel Pentium 4 and Intel Xeon Processor Optimization Manual". Short descriptions of each of the known issues are presented below [12].

4.1 Spin-Waits

A spin-wait loop is a technique used in multithreaded applications whereby one thread waits for other threads. The wait can be required for protection of a critical section, for barriers or for other necessary synchronizations. Typically the structure of a spin-wait loop consists of a loop that compares a synchronization variable with a predefined value. On a processor with a super-scalar speculative execution engine, a fast spin-wait loop results in the issue of multiple read requests by the waiting thread as it rapidly goes through the loop. These requests potentially execute out-of-order. When the processor detects a write by one thread to any read of the same data that is in progress from another thread, the processor must guarantee that no violations of memory order occur.

To ensure the proper order of outstanding memory operations, the processor incurs a severe penalty. The penalty from memory order violation can be reduced significantly by inserting a PAUSE instruction in the loop. If the duration

of the spin-wait is before a thread updates the variable, then the spinning loop consumes execution resources without accomplishing any useful work [13]. To prevent a spin-wait loop from consuming resources that a waiting thread may use, developers will insert a call to Sleep (0). This allows the thread to yield if another thread is waiting. But if there is no waiting thread, the spin wait loop will continue to execute. On a multi-processor system, the spin-wait loop consumes execution resources but does not affect the application performance. On a system with Hyper-Threading enabled processors, the consumption of execution resources without contribution to any useful work can negatively impact the overall application performance [7].

4.2 Write-Combining Store Buffers

Data is read from the first level cache - the fastest cache - if at all possible. If the data is not in that level, the processor attempts to read it from the next level out, and so on. When data is written, it is written to the first level cache only if that cache already contains the specific cache line being written, and "writes-through" to the second level cache in either case. If the data cache line is not in the second level cache, it will be fetched from further out in the memory hierarchy before the write can complete.

Data store operations place data into "store buffers", which stay allocated until the store completes. Furthermore, there are a number of "write combining" (WC) store buffers, each holding a 64 byte cache line. If a store is to an address within one of the cache lines of a store buffer, the data can often be quickly transferred to and combined with the data in the WC store buffer, essentially completing the store operation much faster than writing to the second level cache. This leaves the store buffer free to be re-used sooner - minimizing the likelihood of entering a state where all the store buffers are full and the processor must stop processing and wait for a store buffer to become available [22].

The Intel NetBurst architecture, as implemented in the Intel Pentium 4 and Xeon processors, has 6 WC store buffers. If an application is writing to more than 4 cache lines at about the same time, the WC store buffers will begin to be flushed to the second level cache. This is done to help insure that a WC store buffer is ready to combine data for writes to a new cache line. The "Intel Pentium 4 Processor and Intel Xeon Processor Optimization" guide recommends writing to no more than 4 distinct addresses or arrays in an inner loop, in essence writing to no more than 4 cache lines at a time, for best performance.

With Hyper-Threading enabled processors, the WC store buffers are shared between two logical processors on a single physical processor. Therefore, the total number of simultaneous writes by both threads running on the two logical processors must be counted in deciding whether the WC store buffers can handle all the writes [5]. In order to be reasonably certain of getting the best performance by taking fullest advantage of the WC store buffers, it is best to split inner loop code into multiple inner loops, each of which writes no more than two regions of memory. Generally look for data being written to arrays with in

incrementing index, or stores via pointers that move sequentially through memory. Writes to elements of a modest-sized structure or several sequential data locations can usually be counted as a single write, since they will often fall into the same cache line and be write combined on a single WC store buffer.

4.3 64K Alias Conflict

The Intel Xeon processor with Hyper-Threading Technology shares the first level data cache among logical processors. Two data virtual addresses that reside on cache lines that are modulo 64 KB apart will conflict for the same cache line in the first level data cache. This can affect both the first level data cache performance as well as impact the branch prediction unit. This alias conflict is particularly troublesome for applications that create multiple threads to perform the same operation but on different data. Subdividing the work into smaller tasks performing the identical operation is often referred to as data domain decomposition. Threads performing similar tasks and accessing local variables on their respective stacks will encounter the alias conflict condition resulting in significant overall application degraded performance [12].

4.4 Effective Cache Locality

There are many factors that impact cache performance. Effective use of data cache locality is one such significant factor. A well-known data cache blocking technique is used to take advantage of data cache locality. The cache blocking technique restructures loops with frequent iterations over large data arrays by sub-dividing the large array into smaller blocks, or tiles, such that the block of data fits within the data cache. Each data element in the array is reused within the data block before operating on the next block or tile. Depending on the application, a cache data blocking technique is very effective. It is widely used in numerical linear algebra and is a common transformation applied by compilers and application programmers [20]. The L2 cache contains instructions as well as data, compilers often try to take advantage of instruction locality by grouping related blocks of instructions close together as well [13]. However, the effectiveness of the technique is highly dependent on the data block size, the processor cache size, and the number of times the data is reused.

With the introduction of Hyper-Threading Technology in the Intel Xeon processor in which the cache is shared between logical processors, the relationship between block size and cache size holds. The relationship is relative to the number of logical processors supported by the physical processor as well. Applications should detect the data cache size using Intel's CPUID instruction and dynamically adjust cache blocking tile sizes to maximize performance across processor implementations. Be aware that a minimum block size should be established such that the overhead of threading and synchronization does not exceed the benefit from threading [4]. As a general rule, cache block sizes should target

approximately one-half to three-quarters the size of the physical cache for non-Hyper-Threading processors and one-quarter to one-half the physical cache size for a Hyper-Threading enabled processor supporting two logical processors.

5 Summary

HT brings additional performance to many applications but it is not automatic process. The speedup can be achieved via Operating System optimization, following the threading methodology for designing Hyper-Threading apps, avoiding known traps and applying smart thread management practices. In addition there are also a large number of dedicated engineers who are working to analyze and optimize applications for this technology; their contributions will continue to make a real difference to server applications and clustering solutions.

References

1. A. Agarwal, B.-H. Lim, D. Kranz and J. Kubiatowicz: APRIL: A processor Architecture for Multiprocessing. Proc. 17th Ann. Int. Symp. on Computer Archit., May 1990, pg.104-114.
2. R. Alverson et al.: The TERA Computer System. Int. Conf. on Supercomputing, June 1990, pg.1-6.
3. L. A. Barroso et al. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. Proc. 27th Ann. Int. Symp. on Computer Archit., June 2000, pg.282-293.
4. P. Doerffer, O Szulc: Usage of Hyper Threading Technology in CFD. Institute of Fluid-Flow Machinery, Polish Academy of Sciences, Gdansk, Poland, August 2003.
5. M. Fillo et al.: The M-Machine Multicomputer. Proc. 28th Ann. Int. Symposium on Microarchitecture, Nov. 1995.
6. L. Hammond, B. Nayfeh, and K. Olukotun. A Single-Chip Multiprocessor. Computer **30**(9)(1997) 79-85.
7. G. Hinton et al.: The Microarchitecture of the Intel Pentium 4 Processor. Intel Technology Journal. 1st quarter 2001.
8. G. Hinton and J. Shen: Intel's Multithreading Technology. Microprocessor Forum. October 2001, http://www.intel.com/research/mrl/Library/HintonShen.pdf
9. Intel Corporation. IA-32 Intel Architecture Software Developer's Manual, Vol. 2: Instruction Set Reference. Order number 245471, 2001, http://developer.intel.com/design/Pentium4/manuals
10. Intel Corporation. IA-32 Intel Architecture Software Developer's Manual, Vol. 3: System Programming Guide. Order number 245472, 2001, http://developer.intel.com/design/Pentium4/manuals
11. Intel Corporation. The Intel Vtunet Performance Analyzer, http://developer.intel.com/software/products/vtune
12. Intel Corporation. Intel OpenMP C++/Fortran Compiler for Hyper-Threading Technology: Implementation and Performance, Xinmin Tian et al.
13. Intel Corporation. Using Spin-Loops on Intel Pentium 4 Processor and Intel Xeon Processor MP, Application Note AP-949, http://developer.intel.com/software/products/itc/sse2/sse2_appnotes.htm

14. D. J. Johnson: HP's Mako Processor. Microprocessor Forum, October 2001, http://www.cpus.hp.com
15. J. A. Redstone, S. J. Eggers and H. M. Levy: An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture. Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000.
16. Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmark, http://www.spec.org/osg/cpu2000
17. B.J.Smith: Architecture and Applications Of the HEP Multiprocessor Computer System. SPIE Real Time Signal Processing IV, 1981, pg. 241-248.
18. A. Snavely and D. M. Tullsen. Symbiotic Job Scheduling for a Simultaneous Multithreading Processor. Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Nov. 2000.
19. J. M. Tendler, S. Dodson and S. Fields: POWER4 System Microarchitecture. Technical White Paper. IBM Server Group. October 2001.
20. D. Tullsen, S. Eggers, and H. Levy. Simultaneous Multithreading: Maximizing On-chip Parallelism. 22nd Ann. Int. Symp. on Computer Archit., June 1995.
21. D. Tullsen et al.: Exploiting Choice: Instruction Fetch and Issue on Unimplementable Simultaneous Multithreading Processor. 23nd Ann. Int. Symp. on Computer Archit., May 1996.
22. D. Vianney: Hyper-Threading speeds Linux. Linux Kernel Performance Group, Linux Technology Center, IBM, January 2003.

Building Autonomic Clusters: A Response to IBM's Autonomic Computing Challenge

Andrzej Goscinski, Jackie Silcock, and Michael Hobbs

School of Information Technology, Deakin University
Geelong, Vic 3217, Australia
{ang, Jackie, mick}@deakin.edu.au

Abstract. IBM has announced Autonomic Computing as one of its Grand Challenges. A list of characteristics was generated to describe an autonomic computing system. Many researchers and research group have responded positively to the challenge by initiating research around one or two of the characteristics identified by IBM as the requirements for Autonomic Computing. We have addressed the IBM's request to contribute to the challenge by building an autonomic operating system that possesses nearly all IBM characteristics. This paper contains our response to the IBM's Autonomic Computing Challenge.

1 Introduction

Execution performance has been practically the only issue addressed by researchers and developers of computers. Supercomputer research and development concentrates solely on fast processing. Only because of their very high ratio of cost to performance, there is a move toward shared memory systems and (dedicated) clusters. There is also a strong trend in parallel computing to move to cheaper, general- purpose distributed systems, called non- dedicated clusters, that consist of commodity off-the-shelf components such as PCs connected by LANs. However, clusters are still entirely characterized using only execution performance attributes.

Of course, execution performance is a critical assessment criterion. However, concentration on it has led to the neglect of other issues such as security, reliability, ease of use and ease of programming. A lot of theoretical work has been carried out to address these issues; however this research has not resulted in any effective solutions. This has become a major obstacle to the common exploitation of parallel processing on nondedicated clusters. Research and development projects have been concentrating on improving performance of non-dedicated cluster, in particular through the development and employment of middleware. Although middleware technologies allow building clusters, they do nothing to make them commonly used because programmers must be involved in many activities, which are of an operating system nature. However, for a truly effective solution these middleware applications need to be incorporated into a single comprehensive operating system for clusters. Some impressive applications have been developed but only by teams of specialists.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 27–35, 2004.
© Springer-Verlag Berlin Heidelberg 2004

A review by Goscinski [4] shows that none of the research performed thus far has looked at the problem of developing a technology that goes beyond high performance execution and allows cluster operating systems to be built for supporting high availability and provide services reliably to all users. The deficiencies of current research in parallel processing on clusters have also been identified in [1, 11], and the need for high availability and fault tolerance in clusters is advocated in [15, 17]. A similar situation exists in the area of Distributed Shared Memory (DSM). A comprehensive program to re-examine "an obsession with faster, smaller, and more powerful" and "to look at the evolution of computing from a more holistic perspective" has been launched by IBM in 2001 [9, 10].

We claim that Autonomic Computing on clusters could be achieved through autonomic operating systems. The aim of this paper is to substantiate this claim, to present the services underlying the autonomic cluster effort to build an autonomic operating system, and to show the outcomes of our work in the form of the Holos ('whole' in Greek) cluster operating system that is built to offer autonomic parallel computing on non-dedicated clusters.

2 Related Work

Autonomic computing is seen by IBM [10] as "the development of intelligent, open systems capable of running themselves, adapting to varying circumstances in accordance with business policies and objectives, and preparing their resources to most efficiently handle the workloads we put upon them".

According to [9], an autonomic computing system could be described as one that possesses at least the following characteristics: knows itself; configures and reconfigures itself under varying and unpredictable conditions; optimises its working; performs something akin to healing; provides self-protection; knows its surrounding environment; exists in an open environment; and anticipates the optimised resources needed while keeping its complexity hidden (most critical for the user).

IBM's Grand Challenge identifying Autonomic Computing as a priority research area has brought research carried out for many years on self-regulating computers into focus. While IBM is advocating a "holistic" approach to the design of computer systems much of the focus of researchers is upon failure recovery rather than uninterrupted, continuous, adaptable execution. The latter includes execution under varying loads as well as recovery from hardware and software failure. A number of projects related to Autonomous Computing are listed by IBM in [10]. While many of these systems engage in some aspects of Autonomic Computing none engage in research to develop a system which has all eight of the characteristics required.

3 Services Providing Autonomic Computing on Non-dedicated Clusters

We propose a set of services that must be provided to develop an autonomic computing environment on a non-dedicated cluster:

1. To allow a system to know itself:
   - The system must identify its components and have knowledge of static and dynamic parameters of cluster components and the whole cluster.
2. To allow a system to configure and reconfigure itself under varying and unpredictable conditions:
   - The system must adaptively and dynamically form a parallel virtual cluster;
   - The system must offer high availability of resources.
3. To allow a system to optimize its working:
   - Static allocation and load balancing must be employed;
   - Changing scheduling from static to dynamic, and vice versa;
   - Changing performance indices among computation-oriented, communication-oriented and computation and communication-oriented should be provided;
   - Process migration, creation and duplication must be used;
   - Dynamic setting of priorities of computations of different parallel applications must be provided.
4. To allow a system to perform something akin to healing:
   - Identification of faults, checkpointing and recovery through the use of migration and redundancy must be employed.
5. To allow a system to provide self- protection:
   - Anti-virus detection and recovery, resource protection should be mandatory;
   - Encryption and authentication should be used.
6. To allow a system to know its surrounding environment:
   - Resource discovery of other similar clusters must be provided;
   - Advertising and trading services should be used.
7. To prevent a system from existing in a hermetic environment:
   - The system should be able to communicate and share resources.
8. To allow a system to anticipate the optimized resources needed while keeping its complexity hidden (most critical for the user):
   - A programming environment that is transparent and simple to use is provided.

4 Holos' Autonomic Computing Services

To demonstrate that it is possible to develop an autonomic non-dedicated cluster, which could be routinely employed by ordinary engineers, managers, etc, we decided to build a new autonomic cluster operating system, called Holos.

4.1 Holos Architecture

Holos was developed as an extension to the Genesis System [4] based on the P2P paradigm and object-based approach (where each entity has a name) supported by a microkernel [3]. Holos uses a three level hierarchy for naming: user names, system names, and physical locations. The system name is a data structure, which allows objects in the cluster to be identified uniquely and serves as a capability for object protection [5].

The microkernel creates a virtual machine and provides local inter-agent communication (IAC). Other operating system services are provided by a set of cooperating agents. There are three groups of agents: kernel, system, and application agents. The kernel and system agents are stationary, application agents are mobile. All agents communicate using messages. The system agents, which form a basis of an autonomic operating system for nondedicated clusters, are as follows:

- Resource Discovery Agent – collects data about computation and communication load; and supports establishment of a parallel virtual cluster;
- Global Scheduling Agent – maps application agents on the computers that make up the Holos virtual parallel cluster;
- Execution Agent – coordinates the single, multiple and group creation and duplication of application agents on both local and remote computers;
- Migration Agent – coordinates the relocation of an application agent or a set of application agents on one computer to another computer or a set of computers;
- DSM Agent – hides the distributed nature of the cluster's memory and allows programmers to write their code as though using physically shared memory;
- Checkpoint Agent – coordinates checkpoint creation for an executing application;
- Inter-agent Communication (IAC) Agent – supports remote inter-agent communication through the discovery and re-direction of messages to agents located on remote computers and group communication.
- Trading Agent – supports advertising and sharing services through service exporting, importing and revoking.

4.2 Holos Possesses the Autonomic Computing Characteristics

The following subsections present the agents, which provide services that allow the operating system to offer autonomic operating system and support autonomic parallel computing on nondedicated clusters.

4.3 Communication among Parallel Agents

To hide distribution and make remote inter-agent communication look identical to communication between local application agents, we decided to build the whole operating system services of Holos around the inter-agent communication facility.

Table 1. Holos agents that offer services to provide AutonomicComputing

Autonomic Computing Requirement	Cooperating Holos Agents – Relationships Among Autonomic Elements
To allow a system to know itself	Resource Discovery Agent
A system must configure/reconfigure itself under varying and unpredictable conditions	Resource Discovery, Global Scheduling Migration, Execution, and Availability Agents
A system must optimize its working	Global Scheduling, Migration, and Execution Agents
A system must perform something akin to healing	Checkpoint, Migration, and Global Scheduling Agents
A system must provide self-protection	Capabilities in the form of System Names
A system must know its surrounding environment	Resource Discovery, and Trading Agents
A system can not exist in a hermetic environment	Inter-agent Communication, and Trading Agents
A system must anticipate the optimized resources needed, while keeping its complexity hidden (critical for the user)	DSM, and Execution Agents, DSM Programmming Environment, Message Passing Programming Environment, PVM/MPI Programming Environment

Communication among parallel agents of an application is supported in Holos by both the inter-agent communication component of the microkernel, responsible for local agent communication, and IAC Agent, responsible for communication among remote agents and group communication.

The IAC Agent is also responsible for both local and remote address resolution for group communication. Messages that are sent to a group require the IAC Agent to resolve the destination agent location and provide the mechanism for the transport of the message to the requested group members [13].

4.4 Establishment of a Parallel Virtual Cluster

The Resource Discovery Agent [5, 12] plays a key role in the establishment of the parallel virtual cluster upon a cluster. This agent identifies idle and/or lightly loaded computers and their resources; collects both computational load and communication patterns for each agent executing on a given computer, and provides this information to the Global Scheduling Agent, which uses it to establish a parallel virtual cluster.

The parallel virtual cluster changes dynamically in time as some computers become overloaded, and some computers become idle/lightly loaded and can become a component of the parallel virtual cluster. The dynamic nature of the parallel virtual cluster creates an execution environment that can address the requirements of application, which when executed can expand or shrink.

4.5 Mapping Parallel Agents to Computers

Mapping parallel agents to computers of a parallel virtual cluster is performed by the Global Scheduling Agent. This agent combines static allocation and dynamic load balancing components, which allow the system to provide mapping by finding the best locations for parallel agents of the application to be created remotely or for locally created agents to be moved to selected computers and to react to large fluctuations in system load. The decision to switch between the static allocation and dynamic load balancing policies is dictated by the scheduling policy using information gathered by the Resource Discovery Agent.

4.6 Agent Creation

In Holos, each computer is provided with a (Remote) Execution (REX) Agent, which is responsible for local agent creation [7]. A local REX Agent is capable of contacting a remote REX Agent to create a remote agent on its behalf. The remote agent creation service employs multiple agent creation that concurrently creates n parallel agents on a single computer, and group agent creation [7] that is able to concurrently create agents on m selected computers. These mechanisms are of great importance for instance for SPMD based applications. An agent is created from an image stored in a file. The File Agent supports this operation. To achieve high performance of the group agent creation operation, a copy of the file that contains a child image is distributed to selected computers by a group communication facility.

4.7 Agent Duplication and Migration

Parallel agents of an application can also be instantiated on the selected computers of the parallel virtual cluster by duplicating an agent locally by the REX Agent and, if necessary, migrating it to selected computer(s) [8].

Migrating an application agent involves moving the agent state, address space, communication state, and any other associated resources. Kernel agents, such as Agent, Space, and IAC Agents, are involved in agent migration. The Migration Agent only plays a coordinating role [2]. Group agent migration is performed, i.e., an agent can be concurrently migrated to n computers selected by the Global Scheduling Agent.

4.8 Computation Co-ordination

It is quite natural that when a parallel application is executed on a cluster a parent agent and a child agent may reside on different machines throughout their lifetimes. These agents can also move in order to improve performance by balancing load. Thus, in Holos, the parent's origin computer manages all agent "exits" and "waits" issued from the parent and its children. Furthermore, child agents must co- ordinate their execution by waiting for both data allocation at the beginning of their execution and the completion of the slowest agent in

the group in order to preserve the correctness of the application, implied by a data consistency requirement. In the Holos system barriers are employed for this purpose.

4.9 Checkpointing

Checkpointing and fault recovery have been selected to provide fault tolerance in Holos. Holos uses coordinated checkpointing, which requires that nondeterministic events be prevented during the creation of checkpoints. To control the creation of checkpoints the Checkpoint Agent is employed. This agent is placed on each computer and invokes the kernel agents to create a checkpoint of agents on the same computer [14]. Currently, fault recovery is the subject of our research. A basis of this research is agent migration, which moves a selected checkpoint to the specified computer.

4.10 Trading – Toward Grids

Trading and resource discovery have been studied to build basic service allowing Holos services and applications to be offered to both other users working with Holos and users of other systems. Note that research of availability and the development of services supporting availability in clusters has been initiated based on lessons learned when studying object naming and trading in distributed systems [12]. A copy of a trader agent runs on each computer of the cluster. Each Holos trader is such an agent that preserves user autonomy as in a centralized environment; and supports sharing both by advertising, exporting or withdrawing services.

4.11 Programming Interface

Holos provides transparent communication services of standard message passing (MP) and DSM as its integral components. The PVM communication is transparently provided by a service that is only a mapping of the standard PVM services onto the Holos communication services and benefits from additional services, which are not provided by operating systems such as Unix or Windows.
Holos Message Passing. The standard MP service within the Holos parallel execution environment is provided by the Local IAC component of the microkernel and the IAC Agent that is supported by the Network Agent.
Holos PVM. PVM has been ported to Holos as it allows exploiting advanced message passing based parallel environment [Rough and Goscinski 99]. The functionality that the PVM server provides in Unix systems has been effectively substituted with services provided by Holos. In this PVM server free environment, PVM servers communicate directly with each other, significantly improving the performance of IAC. Removing the PVM server from the PVM model also improves the reliability of PVM applications.

Distributed Shared Memory. Holos DSM exploits the conventional "old" memory sharing approach (to write shared memory code using concurrent programming skills) by using the basic concepts and mechanisms of memory management to provide DSM support [Silcock and Goscinski 99]. The unique feature of Holos DSM is that it was integrated into the memory management of the operating system, in particular into the Space Agent, which provides conventional memory management functions. Holos DSM employs semaphores and barriers to synchronize DSM agents.

5 Conclusion

The Holos autonomic operating system has been built from scratch to offer an autonomic non-dedicated cluster by providing availability, parallelism management, transparency fault tolerance, and easy programming. This system relieves developers from programming operating system oriented activities, and provides to developers of next generation application software both message passing and DSM. The mechanisms that provide autonomic computing services have been implemented by respective agents or part of other system agents. All the cooperating agents that employ these mechanisms offer availability, support parallelism management, transparency, fault tolerance and ease of programming. In summary, the development of the Holos cluster operating system demonstrates that it is possible to build an autonomic non-dedicated cluster.

References

1. Cluster Computing White Paper, Version 2.0, M. Baker (Editor), Dec. 2000.
2. D. De Paoli and A. Goscinski: "The RHODOS Migration Facility". J. of Systems and Software, **40** (1998) 51-65.
3. D. De Paoli et al.: "The RHODOS Microkernel, Kernel Servers and Their Cooperation". Proc. First IEEE Int. Conf. on Algorithms and Architectures for Parallel Processing - ICA3PP'95, April 1995.
4. A. Goscinski: "Towards an Operating System Managing Parallelism of Computing on Clusters of Workstations". Future Generation Computer Systems, 2000, 293-314.
5. A. Goscinski and A. Haddock: A Naming and Trading Facility for a Distributed System. The Australian Computer Journal, No. 1, 1994.
6. A. Goscinski, M. Hobbs and J. Silcock: "GENESIS: An Efficient, Transparent and Easy to Use Cluster Operating System". Parallel Computing, 2002.
7. M. Hobbs and A. Goscinski: "A Concurrent Process Creation Service to Support SPMD Based Parallel Processing on COWs". Concurrency: Practice and Experience. **11**(13) 1999.
8. M. Hobbs and A. Goscinski: "Remote and Concurrent Process Duplication for SPMD Based Parallel Processing on COWs". Proc. Int. Conf. on High Performance Computing and Networking - HPCN Europe'99, Amsterdam, April 1999.
9. P. Horn: Autonomic Computing: IBM's Perspective on the State of Information Technology, IBM, October 2001.

10. IBM: http://www.research.ibm.com/autonomic/academic, (accessed 6 May 2003).
11. R. Lottiaux and C. Morin: "Containers: A Sound Basis for a True Single System Image". Proc. First IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Brisbane, May 2001.
12. Y. Ni and A. Goscinski: "Trader Cooperation to Enable Object Sharing Among Users of Homogeneous Distributed Systems". Computer Communications, **17**(3) (1994) 218-229.
13. J. Rough and A. Goscinski: "Comparison Between PVM on RHODOS and Unix", Proc. Fourth Int. Symp. on Parallel Architectures, Algorithms and Networks, I-SPAN'99, Freemantle, June 1999.
14. J. Rough and A. Goscinski: "The Development of an Efficient Checkpointing Operating System of the GENESIS Cluster Operating System". Future Generation Computer Systems, 2003, (in press).
15. K. Shirriff, et al.: Single-System Image: The Solaris MC Approach. Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications, PDPTA'97, Las Vegas, June 30 – July 3, 1997.
16. J. Silcock and A. Goscinski: "A Comprehensive DSM System That Provides Ease of Programming and Parallelism Management". Distributed Systems Engineering, **6** (1999) 121–128.
17. B. Walker: Implementing a Full Single System Image UnixWare Cluster: Middleware vs Underware, Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications, PDPTA'99, 1999.

Configurable Microprocessor Array for DSP Applications

Oleg Maslennikow, Juri Shevtshenko, and Anatoli Sergyienko

Technical University of Koszalin, ul.Partyzantow 17,
75-411 Koszalin, Poland
oleg@ie.tu.koszalin.pl

Abstract. The configurable microprocessor array for DSP applications is proposed, in which each cell is the microprocessor with RISC architecture, represented as a soft IP-core. This IP-core is generated automatically by the special soft-core generator, which is based on the approach to optimization of a microprocessor architecture for its further implementation in FPGA devices. Soft-core generator analyzes the executing program of each microprocessor of the array and eliminates all unused units from the resulting VHDL-model of the microprocessor. Therefore, hardware volume of each cell of this array is minimized, and is adapted to the used instruction subset. The soft-core generator provides both high throughput and minimized hardware volume with speedups the design process. It was probed in design the microprocessor array for solving the linear equation system with Toeplitz matrices.

1 Introduction

Modern DSP applications, like MPEG-4 standard packing, vocoders, etc. are characterized by both high algorithm complexity (thousands of instruction rows in the programs) and computational intensiveness (several millions and billions of operations per second). Besides, the algorithm set in the device can be substituted dynamically according to the data stream parameters, or to the application exchanging. Both hardware and power consumption minimization are of great demand. As a rule, such applications are implemented in the signal microprocessors. To achieve the high throughput, the signal microprocessors become parallel processor systems with several processor units (PUs). But the hardware of such microprocessors is not utilized very well. This is explained by that many DSP operators could not directly mapped into the microprocessor instructions. Besides, the modern compilers for such signal microprocessors are not effective ones.

Reconfigurable computing is the real alternative both to ASICs and signal microprocessors now. Its advantage is provided by broad applicability, due to reconfiguration properties and high performance, through the potential parallelism exploitation and direct implementation of any algorithm operator. The field programmable gate arrays (FPGAs) is the most commonly used raw for the reconngurable computing. At present, the configurable computer architecture based on FPGAs is of great interest due to its flexibility and high throughput in a wide set of applications [1]. In the last decade the density of FPGAs,

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 36–41, 2004.
© Springer-Verlag Berlin Heidelberg 2004

their clock frequency, and routing capabilities are increased dramatically. The high density and clock frequency of modern FPGAs provide their extremely high throughput. For example, the Xilinx Virtex-2 FPGAs consist of several tenths of combinational multipliers, data RAMs, and it takes about 200 configurable logic block (CLB) slices per one multiplier. The expanding of FPGAs at the field of modern DSP applications is limited now because of the labor consumable process of mapping the algorithms into FPGA. The author's experience in the development of the configurable parallel system for the QR - decomposition has proved this thesis [2]. One of the way to solve this problem is the intellectual property (IP) core reuse. The another way is programming the parallel system of processing units (PUs), which is configured in FPGA.

In this paper, the configurable microprocessor array for DSP applications is proposed, in which each cell is the consists of the microprocessor with RISC architecture (represented as a soft IP-core), and optionally the extension unit destined to the different DSP application. The hardware volume of the microprocessor soft core is adapted to the used instruction subset, due to elimination of the unused units. The VHDL-model of each cell is generated automatically by the special soft-core generator, which analyzes the executing program of each microprocessor of the array and eliminates all unused units. Therefore, hardware volume of each cell of this array is minimized, and adapted to the used instruction subset. The soft-core generator provides both high throughput and minimized hardware volume, and provides balancing the PU hardware volume with the FPGA resources.

2 Mapping Parallel Algorithms into Configurable Microprocessor Array

In the ref. [4], the PU array for image processing applications is proposed which is configured in FPGA. Each PU is implemented as the IP soft core with the architecture of the well-known microcontroller i8051. The core hardware volume is exchanged in the range of 500 - 1100 CLB slices, depending on the core functionality [3]. This means that it takes from 3 to 6 multipliers per one PU core in the Xilinx Virtex-2 FPGA. As a result, the most of multipliers are unused, and hardware utilization is not effective one. Such situation also occurs when another IP soft cores of RISC microprocessors are used in the configurable microcontroller array, like ARC core, or Leon SPARC core, which consist of much more CLBs.

The proposed array for realization of parallel DSP algorithms provides balancing the PU hardware volume with the FPGA resources. That means that the PU soft core has the hardware volume less than 200 - 400 CLB slices, and 1 - 2 multipliers. As the PU core the RISC-ST microprocessor soft core is selected, which is in details described in [5]. Its architecture is represented in Fig 1, where PC denotes the program counter block, RRAM denotes the register memory block,PRAM - the program memory block, INT-CNTRL - the interrupt service block, INSTR-R - the local control unit. This core consists of the base core and

the hardware extension unit. The base core has the RISC 16-bit architecture with the 2-staged instruction pipeline. It performs each instruction for a single clock cycle. The instruction RAM (Program RAM) is separated from the data RAM. To achieve the high performance in the control intensive applications, the delayed branch mechanism is used. The high speed interrupts, and subroutine calls are supported by the hardware stack. After the interrupt routine end the instruction pipeline recovery has not any difficulties, because the heavy instructions, like jump instructions, delayed branch instructions, are not interruptible. The base PU core has the hardware volume only 190 CLB slices in the Xilinx Virtex-2 FPGA devices.

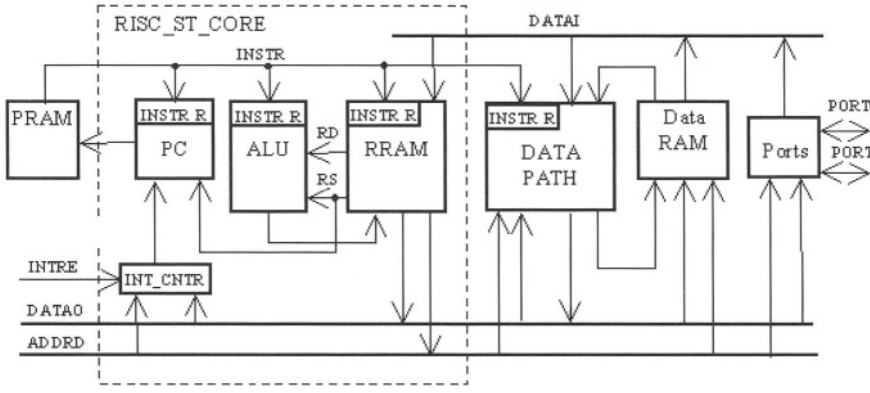


Fig. 1. Architecture of configurable array for DSP applications

The PU hardware extension unit implements the proper instruction set extension. This extension is adapted to the different DSP applications. Therefore, hardware volume of this unit can very depending on the given instruction set, and precision of computations. The unit for implementing the FFT algorithm is different from the unit for filter calculations, and provides, for example, the hardware implementation of the bit reverse addressing.

The PU core is described by VHDL, and runs in Xilinx Virtex-2 devices at the clock frequency, which is equal to 90 MHz. The core hardware volume is exchanged in the range of 190 - 500 CLB slices, depending on the implemented instruction set extension. The assembler was developed, which generates the program codes, and outputs the table of generic constants for the hardware extension unit selection. Moreover, IP core generator was developed, which generates this PU soft core with the program RAM, constant ROM content, and proper hardware extension unit.

The parallel processor system has the ring structure. But it can be freely exchanged to any structure, which is supported by the reconfigurable nature of

FPGA. The PUs interact each other by data buffers and interrupt mechanism. Such architecture provides the wave propagation processor implementation, and expanding the PU number due with increase of the FPGA device number in the system. Due to small PU hardware volume one FPGA device can contain more than hundred of PUs, and provide the throughput up to ten billions of multiplications and additions per second.

The system configuring process has the following three stages. On the first stage, the user microprocessor programs are designed and compiled. By this process the software pipelining for the processor array is used, which described in [6]. The derived parallel program is debugged, tested, and tuned using the behavioral model of the processor array in the VHDL simulator. Such process can be accelerated when the hardware accelerator is attached to the simulator, which is based on the FPGA device. During program compiling, the unused instruction codes and data addresses are fixed.

On the second stage, the PU cores are generated, in which the unused units and logic cells are taken off. Each node program is mapped into the separate PU core. If the resulting architecture is SIMD - architecture, then the only one PU core is generated. When the application needs the intensive calculation of some special functions, for example, floating point operations, then the proper functional unit can be attached to the PU core. This unit has to be a fully pipelined data flow path with the high throughput, which can supersede the throughput of the signal microprocessor. The structure of such unit is synthesized by the method, described in [7].

On the third stage, all the PU cores, or copies of a single PU core are attached to the microprocessor array netlist, and the whole project is translated into the configuration file of the target FPGA device.

3 Experimental Results

The proposed IP-core generator was probed in programming the configurable microprocessor array for solving the linear equation system with the Toeplitz matrix. This problem is solved in DSP systems for adaptive filtering, spectrum estimating, voice coding, etc. Usually this problem is solved using the floating point data representation or the integer dates with doubled length, and specific algorithms which support the error minimization. The $(N + 1)$ processor systolic array solves the $N * N$ Toeplitz matrix problem for N iterations using the Schur algorithm [8]. Such computational schema is used in our example as well.

The disadvantage of this schema consists in that the nodes with the division operation form the critical path. Moreover, this operation is time consumable in the RISC processors. Therefore, it limits the throughput of the whole array. To minimize the division delays the untraditional data representation is used. Each data x is represented by two integer numbers, which are numerator n_x and denominator d_x, i.e. the data is equal to the fraction

$$x = n_x / d_x \tag{1}$$

At first *(N–1)* iterations all the calculations are implemented with such data. For example, multiplication, division, and addition operations are performed in accordance to the following expressions:

$$x * y = n_x * n_y/(d_x * d_y) \tag{2}$$

$$x/y = n_x * d_y/(d_x * n_y) \tag{3}$$

$$x + y = (n_x * d_y + n_y * d_x)/(d_x * d_y) \tag{4}$$

At the last iteration denominators divide numerators to derive the algorithm results. Such data representation provides both small calculation errors and expanded dynamic range comparing to the usual integer data representation.

The PU hardware extension unit consists of two multipliers, and implements the multiplication and division for a single clock cycle, and addition for two clock cycles. To provide the minimum calculation errors each operation is finished by the normalization of resulting numerator and denominator, shifting left their codes to the equal bit number.

The PU hardware volume is equal to 380 CLB slices, 2 multipliers and 2 RAM blocks. The PU system for N=10 is fitted the 75% of the hardware volume of XC2V1000 device. It implements the Schur algorithm for 1.23 microseconds (without to take to account the data input-output). The average speed is equal to 170 millions operations per second, like addition, multiplication, division of fractional dates. The system with up to 84 such PUs can be configured in the XC2V8000 device, and provide approximately 1900 millions operations per second when implementing this algorithm.

4 Conclusions

The configurable microprocessor array for DSP applications is proposed, in which each cell is the consists of the microprocessor with RISC architecture (represented as a soft IP-core), and optionally the extension unit destined to the different DSP application. The VHDL-model of each cell is generated automatically by the special soft-core generator, which analyzes the executing program of each microprocessor of the array and eliminates all unused units. Therefore, hardware volume of each cell of this array is minimized, and adapted to the used instruction subset. The soft-core generator is based on the approach to optimization of microprocessor architecture for its further implementation in FPGA devices. The proposed configurable microprocessor array is very useful in such DSP applications where logic intensive calculations, or computations of dates in the unusual format, or complex algorithm computing are of demand. These applications cover MPEG-4 packing, multi-channel CELP vocoders, open key encryption systems, etc. This generator was successfully verified in design the microprocessor array for solving the linear equation system with the Toeplitz matrix.

References

1. Villasenor, J., Hutchings, B.: The flexibility of configurable computing. IEEE Signal Processing Magazine, vol. 15, (1998), 67–84.
2. Sergyienko, A., Maslennikov, O.: Implementation of Givens QR Decomposition in FPGA. Lecture Notes in Computer Science, vol.2328, Springer, (2002), 453–459.
3. Lepekha, V., Sergyienko, A., Kaniewski, J.: VHDL-Model of Ultrafast Microcontroller 8051. Proc. 3-d Region. Conf. Reprogramowalne Uklady Cyfrowe, RUC'2000, Poland,(2000), 35–41.
4. Maslennikov, O., Shevtshenko, Ju., Sergyienko, A.: Configurable microcontroller array. Proc. of the 3-d Int. Conf. on Parallel Computing in Electrical Engineering, PARELEC'2002, Warszaw, Poland, (2002), 47–49.
5. Sergyienko, A.: VHDL for computer development. Kiev, Diasoft, (2003), (In Russian).
6. Sergyienko, A., Kaniewski, J., Maslennikov, O., Wyrzykowski, R.: Mapping regular algorithms into processor arrays using software pipelining. Proc. of the 1-st Int. Conf. on Parallel Computing in Electrical Engineering, PARELEC' 1998, Poland, (1998), 197–200.
7. Kanevski, Ju.S., Sergienko, A., Piech, H.: A Method for the Structural Synthesis of Pipelined Array Processors. Proc. of the 1-st Int. Conf. on Parallel Proc. and Appl. Math., PRAM'94, Poland, (1994), 100–109.
8. Kung, S.Y.: VLSI processor arrays. Prentice Hall, Englewood Cliffs, (1988).

On Generalized Moore Digraphs

Michael Sampels

Institut de Recherches Interdisciplinaires et de
Développements en Intelligence Artificielle,
Université Libre de Bruxelles, CP 194/6,
50, avenue Franklin D. Roosevelt,
1050 Bruxelles, Belgium
msampels@ulb.ac.be

Abstract. The transmission of a strongly connected digraph D is defined as the sum of all distances in D. A lower bound for the transmission in terms of the order n and the maximal outdegree Δ^+ of D can be regarded as a generalization of the Moore bound for digraphs. Bridges and Toueg showed that Moore digraphs in the strong sense exist only for the trivial cases $\Delta^+ = 1$ or $\Delta^+ = n-1$. Using techniques founded on Cayley digraphs, we constructed vertex-symmetric generalized Moore digraphs. Such graphs are applicable to interconnection networks of parallel computers, routers, switches, backbones, etc.

1 Preliminaries

We consider digraphs D that are finite, simple, and strongly connected. The vertex and the arc sets are denoted by $V(D)$ and $A(D)$, respectively. The order of the digraph is defined by $n(D) := |V(D)|$. The distance $d(v,w)$ is defined for $v, w \in V(D)$ as the number of arcs on a shortest path from v to w. The outdegree of a vertex v is defined as $\delta^+(v) := |\{w \in V(D) \mid (v,w) \in A(D)\}|$. The maximal outdegree is denoted by $\Delta^+(D) := \max_{v \in V(D)} \delta^+(v)$. A digraph with all vertices having the same outdegree δ^+ is called δ^+-regular. The diameter of a digraph is defined as the maximal distance between vertices:

$$dm(D) := \max_{v,w \in V(D)} d(v,w) \ .$$

Its transmission is defined as the sum of all distances:

$$\sigma(D) := \sum_{v,w \in V(D)} d(v,w) \ .$$

2 Motivation

Digraphs with small distances between the vertices are relevant to the design of communication networks with high bandwidth and small latency: backbone networks for internet service providers, telecommunication networks with multimedia support, internal structures of routers and switches, architectures of parallel

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 42–49, 2004.
© Springer-Verlag Berlin Heidelberg 2004

and distributed systems, etc. Because of technical and economical reasons each component of the network can only be connected to a limited number of further components. The diameter of a network is a clue for the slowest communication between network components, which is often regarded as the bottleneck for parallel or distributed algorithms. The transmission of a network is, divided by the number of all vertex pairs, an indicator for the speed of an average communication, thus it can be interpreted as a general quality measure.

For undirected graphs the question about their maximal order $n(\Delta, dm)$ for maximal degree Δ and diameter dm is a classical problem in extremal graph theory [3]. It is a well-known result (Moore bound) that

$$n(\Delta, dm) \leq n_{Moore}(\Delta, dm) := 1 + \Delta \sum_{i=1}^{dm} (\Delta - 1)^{i-1} \quad .$$

The equality $n(\Delta, dm) = n_{Moore}(\Delta, dm)$ is satisfied only for the (Δ, dm)-pairs $(\Delta, 1)$ (complete graphs), $(2, dm)$ (circles of odd order), (3,2) (Petersen graph), (7,2) (Hoffman-Singleton graph [14]), and probably for (57,2) [2]; the existence of such a graph with $n_{Moore}(57, 2) = 3\,250$ vertices is still unclear. These graphs are called *Moore graphs.* Although a lot of research has been done on bounds for $n(\Delta, dm)$, only very little is known about the values of $n(\Delta, dm)$. Buset [5] surveyed the recent results, and Comellas [13] maintains a table of lower bounds for $n(\Delta, dm)$.

The question about the maximal order $\bar{n}(\Delta^+, dm)$ of a digraph with maximal outdegree Δ^+ and diameter dm has received less attention than for the undirected case. It can be bounded by

$$\bar{n}(\Delta^+, dm) \leq \bar{n}_{Moore}(\Delta^+, dm) := \sum_{i=0}^{dm} \Delta^{+i} \quad .$$

As Bridges and Toueg [4] showed, the equality $\bar{n}(\Delta^+, dm) = \bar{n}_{Moore}(\Delta^+, dm)$ holds only for the trivial cases of digraphs with $\Delta^+ = 1$ or $dm = 1$. These digraphs are called *Moore digraphs.* Miller and Fris [16] proved that $\bar{n}(2, dm) \leq \bar{n}(2, dm) - 2$ for $dm \geq 3$. Comellas and Fiol [12] constructed lower bounds on $\bar{n}(\Delta^+, dm)$ for $2 \leq \Delta^+ \leq 13$ and $2 \leq dm \leq 11$. Comellas [13] maintains a table with updated results.

As the computation of exact values for $\bar{n}(\Delta^+, dm)$ seems to be very hard, it is interesting to look at a related question. One might ask for the smallest transmission $\sigma(n, \Delta^+)$ of a digraph with order n and maximal outdegree Δ^+.

3 A Bound on the Transmission of a Digraph

Theorem 1. *Let D be a digraph of order n and maximal outdegree Δ^+. Its transmission is bounded by*

$$\sigma(D) \geq \sigma_0(n, \Delta^+) = n \left(\sum_{i=1}^{k-1} i\Delta^{+i} + kR \right)$$

where

$$R = n - \bar{n}_{Moore}(\Delta^+, k-1) > 0,$$

and k is the largest integer such that the last inequality holds.

Proof. The diameter $dm = dm(D)$ cannot be smaller than the largest integer k for which $\bar{n}_{Moore}(\Delta^+, k-1) < n$ holds, because $n \leq \bar{n}_{Moore}(\Delta^+, dm)$. Consider a vertex $v \in V(D)$. We denote $N_i(v) := \{w \in V(D) \mid d(v,w) = i\}$ for $i \in \{1, \ldots, dm\}$. As $\delta^+(v) \leq \Delta^+$, it holds $|N_i(v)| \leq \Delta^{+i}$ for all $i \in \{1, \ldots, dm\}$. Thus, we obtain $\sum_{i=0}^{k-1} |N_i(v)| \leq \sum_{i=0}^{k-1} \Delta^{+i} = \bar{n}_{Moore}(\Delta^+, k-1)$. Hence, $\sum_{i=k}^{dm} |N_i(v)| \geq n - \bar{n}_{Moore}(\Delta^+, k-1) =: R$. We derive the following bound:

$$\begin{aligned}
\sigma_v(D) := \sum_{w \in V(D), w \neq v} d(v,w) &= \sum_{i=1}^{dm} i \cdot |N_i(v)| \\
&= \sum_{i=1}^{k-1} i \cdot |N_i(v)| + \sum_{i=k}^{dm} i \cdot |N_i(v)| \\
&\geq \sum_{i=1}^{k-1} i\Delta^{+i} + kR \ .
\end{aligned}$$

It follows that $\sigma(D) = \sum_{v \in V(D)} \sigma_v(D) \geq n \left(\sum_{i=1}^{k-1} i\Delta^{+i} + kR \right)$. □

4 Generalized Moore Digraphs

The equality $\sigma(n, \Delta^+) = \sigma_0(n, \Delta^+)$ is obviously satisfied for the (n, Δ^+)-pairs of Moore digraphs. Therefore, we introduce for a digraph D with

$$\sigma(D) = \sigma_0(n(D), \Delta^+(D))$$

the term *generalized Moore digraph.* This definition is analogous to the definition of generalized Moore graphs in the undirected case (see [9]). While for the undirected case all 3-regular generalized Moore graphs up to 16 vertices [10,11] and all 4-regular generalized Moore graphs up to 13 vertices [6,7,8] are known, to our knowledge no article has been published on the directed case.

We are interested in the question whether non-trivial generalized Moore digraphs do exist.

5 Construction

With methods founded on Cayley graphs, it was possible to construct undirected graphs that are close to the Moore bound. [17]. Therefore, it is reasonable to use Cayley digraphs as a search space for generalized Moore digraphs. From a practical point of view, the property of Cayley digraphs to be vertex-symmetric is important. Vertex-symmetry is a fundamental virtue in the design of interconnection networks for parallel architectures, because it guarantees feasible routing schemes [15].

Let Γ be a finite group and S be a generator set of Γ, i.e. $\langle S \rangle = \Gamma$ with the property $1_\Gamma \notin S$. The Cayley digraph $C_S(\Gamma)$ is defined by $V(C_S(\Gamma)) = \Gamma$ and $A(C_S(\Gamma)) = \{(g, h) \mid g^{-1}h \in S\}$.

A digraph D is vertex-symmetric if the group of digraph automorphisms $\mathcal{A}(D)$ acts transitively on $V(D)$, i.e. for any two vertices $v, w \in V(D)$ there is a digraph automorphism $\alpha \in \mathcal{A}(D)$ with $v^\alpha = w$. Cayley digraphs are vertex-symmetric [2].

Besche and Eick [1] constructed all groups up to order 1000 except 512 and 768. This catalogue can be used for an exhaustive search for generalized Moore digraphs in the class of Cayley graphs. In order to speed up the calculation, the following observation is useful: A group-automorphism on Γ induces a digraph-automorphism on its Cayley digraphs.

Let α be a group-automorphism on Γ. The set of all group-automorphisms on Γ is denoted by $\mathcal{B}(\Gamma)$. For a generator set S of Γ we define $S^\alpha = \{s^\alpha \mid s \in S\}$. The Cayley digraphs $C_S(\Gamma)$ and $C_{S^\alpha}(\Gamma)$ are isomorphic, since $V(C_S(\Gamma)) = V(C_{S^\alpha}(\Gamma)) = \Gamma$ and

$$(v, w) \in A(C_S(\Gamma)) \Leftrightarrow \bigvee_{s \in S} vs = w \Leftrightarrow \bigvee_{s \in S} v^\alpha s^\alpha = w^\alpha \Leftrightarrow \bigvee_{t \in S^\alpha} v^\alpha t = w^\alpha$$
$$\Leftrightarrow (v^\alpha, w^\alpha) \in A(C_{S^\alpha}(\Gamma)) \ .$$

Thus, for the calculation of all Δ^+-regular Cayley digraphs of a group Γ it suffices to check only one representative per each orbit that $\mathcal{B}(\Gamma)$ induces on the set of generator sets of cardinality Δ^+. The transmission $\sigma(C_S(\Gamma))$ can be calculated by the following algorithm:

$N_0 := \{1_\Gamma\}$; k := 0;
while $\bigcup_{i=0}^{k} N_i \neq \Gamma$ do
  $k := k + 1$; $N_k := \emptyset$;
  for $t \in N_{k-1}$ do
    for $s \in S$ do
      if $ts \notin \bigcup_{i=0}^{k} N_i$ then $N_k := N_k \cup \{ts\}$; fi;
    od;
  od;
od;
$\sigma := |\Gamma| \cdot \sum_{i=0}^{k} i \cdot |N_i|$.

The algorithm is a breadth-first search strategy, which constructs sequentially the distance sets $N_i = \{w \in V(C_S(\Gamma)) \mid d(1_\Gamma, w) = i\}$. As $C_S(\Gamma)$ is vertex-symmetric, it suffices to compute the distance sets for a single root-vertex 1_Γ. Thus, the transmission can be calculated by the given formula.

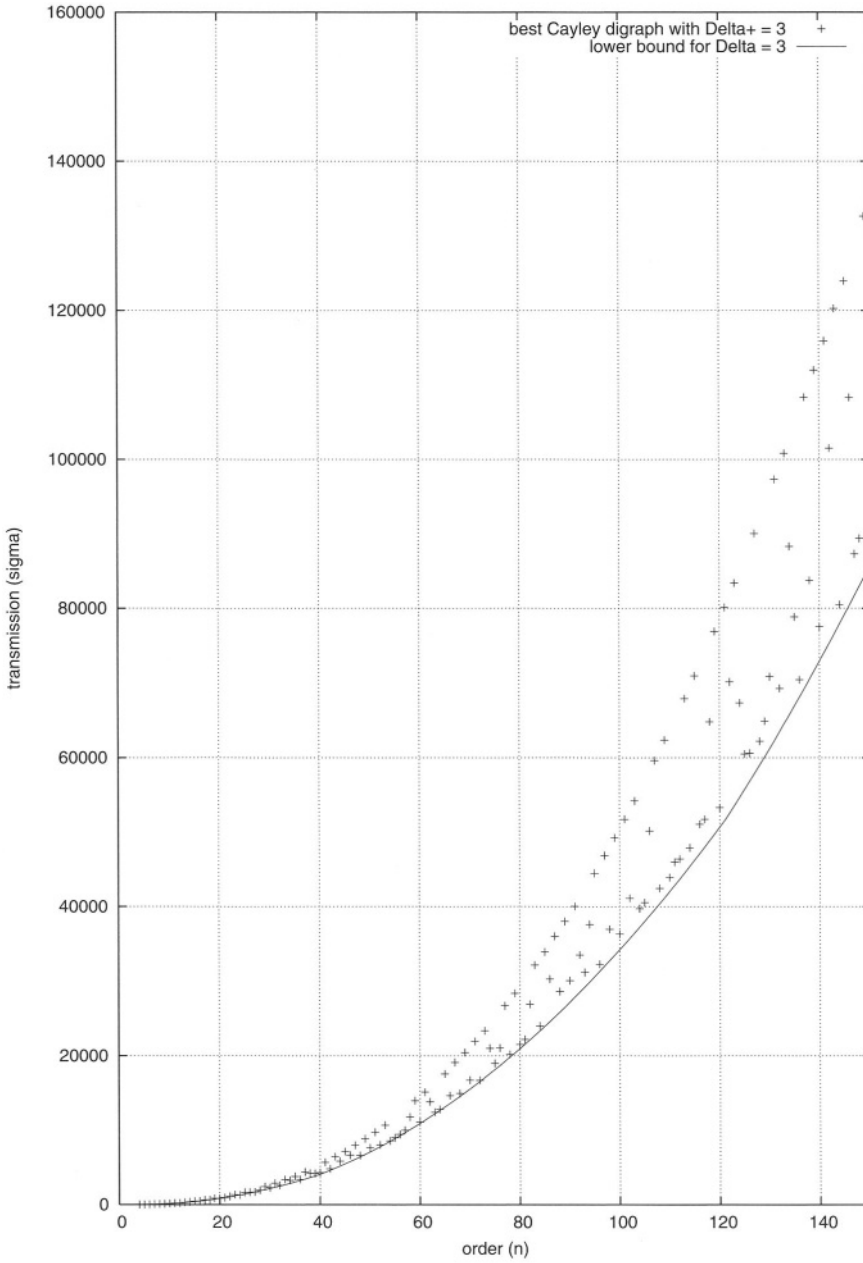


Fig. 1. Smallest transmission of all 3-regular Cayley digraphs of groups with order ≤ 150

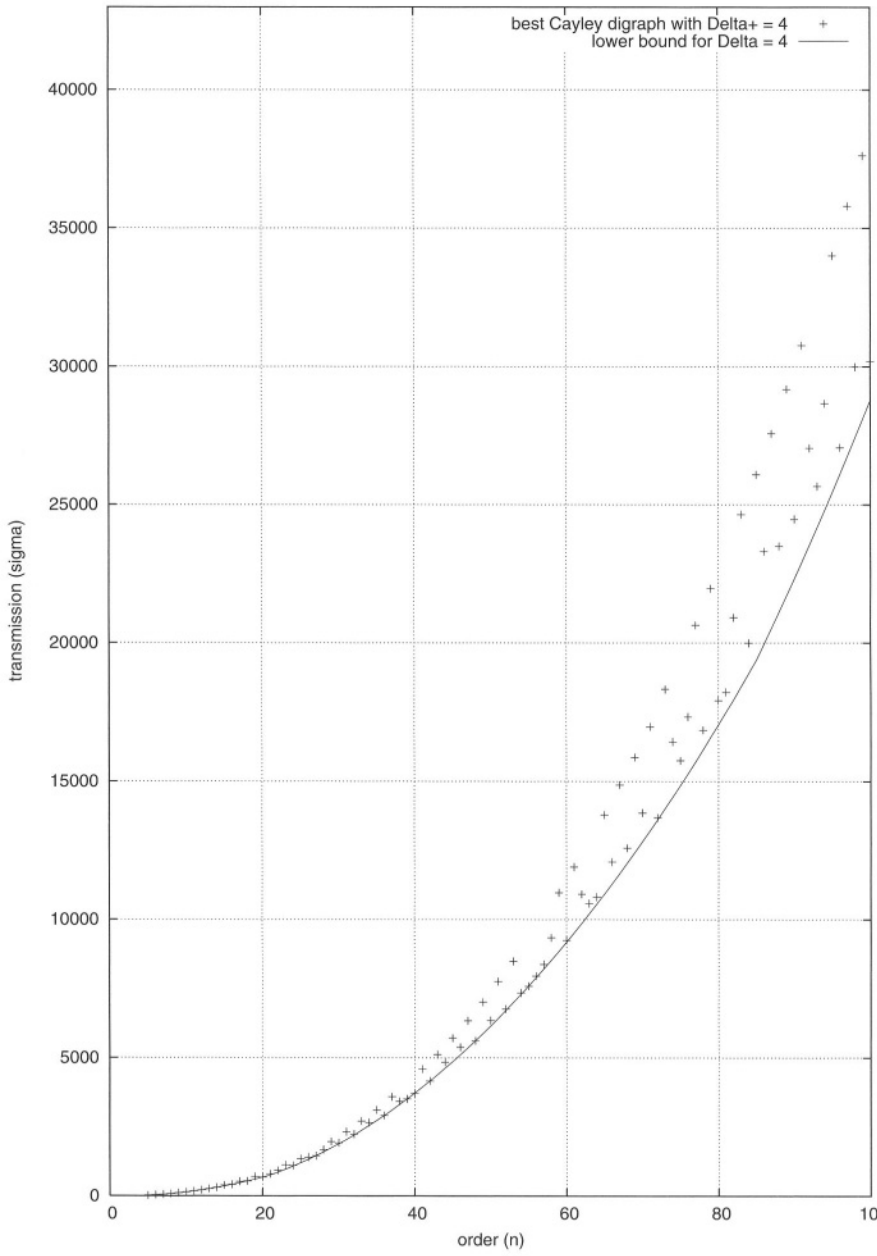


Fig. 2. Smallest transmission of all 4-regular Cayley digraphs of groups with order ≤ 100

6 Results

We analyzed all 3-regular Cayley digraphs of order {4,... ,150} and all 4-regular Cayley digraphs of order {5,... ,100}. For each order the smallest transmission that was discovered is depicted in Fig. 1 and 2.

It is possible to construct a lot of digraphs that are very close to σ_0. The adjacency tables of the digraphs with the smallest transmissions are electronically available on `http://www.cs.uni-essen.de/sampels/mean_distance.html` . In particular, we found generalized Moore digraphs for the (n, Δ^+)-pairs

$$(4,3),\ldots,(10,3),\ (12,3),\ (20,3),\ (21,3),\ (24,3),\ (27,3) \text{ and}$$
$$(5,4),\ldots,(14,4),\ (16,4),\ (18,4),\ (20,4),\ (27,4),\ (36,4),$$
$$(39,4),\ (40,4),\ (42,4),\ (48,4),\ (55,4)\ .$$

Hence, the answer to the last question of Sect. 4 is affirmative. All these digraphs are vertex-symmetric. They might probably be used as interconnection networks in hardware designs, in high-performance switches, or in parallel computers, as it is described in [18,19].

It is an open problem, whether the class of generalized Moore digraphs is finite. A weaker version of this question is whether the class of vertex-symmetric generalized Moore digraphs is finite.

Acknowledgements. This work was supported by the *Metaheuristics Network,* a Research Training Network funded by the Improving Human Potential Programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the author and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. Hans Ulrich Besche and Bettina Eick. The groups of order at most 1000 except 512 and 768. *Journal of Symbolic Computation,* 27:405–413, 1999.
2. Norman Biggs. *Algebraic Graph Theory.* Cambridge University Press, 2nd edition, 1993.
3. Béla Bollobás. *Extremal Graph Theory.* Academic Press, London, New York, San Francisco, 1978.
4. W. G. Bridges and Sam Toueg. On the impossibility of directed Moore graphs. *Journal of Combinatorial Theory, Series B,* 29:339–341, 1980.
5. Dominique Buset. Maximal cubic graphs with diameter 4. *Discrete Applied Mathematics,* 101:53–61, 2000.
6. R. W. Buskens, M. J. Rogers, and R. G. Stanton. A census of tetravalent generalized Moore networks. *Congressus Numerantium,* 52:255–296, 1986.
7. R. W. Buskens and R. G. Stanton. The generalized Moore graphs on eleven vertices. *Journal of Combinatorial Mathematics and Combinatorial Computing,* 1:23–66, 1987.

8. R. W. Buskens and R. G. Stanton. Generalized Moore graphs on twelve and thirteen vertices. *Ars Combinatoria,* 23-B:87–132, 1987.
9. V. G. Cerf, D. D. Cowan, R. C. Mullin, and R. G. Stanton. Computer networks and generalized Moore graphs. *Congressus Numerantium,* 9:379–398, 1973.
10. V. G. Cerf, D. D. Cowan, R. C. Mullin, and R. G. Stanton. Trivalent generalized Moore networks on sixteen nodes. *Utilitas Mathematica,* 6:259–283, 1974.
11. V. G. Cerf, D. D. Cowan, R. C. Mullin, and R. G. Stanton. A partial census of trivalent generalized Moore networks. In Anne Penfold Street and W. D. Wallis, editors, *Proceedings of the 3rd Australian Conference on Combinatorial Mathematics,* Lecture Notes in Mathematics 452, pages 1–27, Berlin, Heidelberg, New York, 1975. Springer-Verlag.
12. F. Comellas and M. A. Fiol. Vertex symmetric digraphs with small diameter. *Discrete Applied Mathematics,* 58:1–11, 1995.
13. Francesc Comellas. The (degree, diameter) problem for graphs. http://www_mat.upc.es/grup_de_grafs/table_g.html .
14. A. J. Hoffman and R. R. Singleton. On Moore graphs with diameters 2 and 3. *IBM Journal of Research and Development,* 4:497–504, November 1960.
15. S. Lakshmivarahan, Jung-Sing Jwo, and S. K. Dhall. Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey. *Parallel Computing,* 19:361–407, 1993.
16. Mirka Miller and Ivan Fris. Maximum order digraphs for diameter 2 or degree 2. In Rolf S. Rees, editor, *Graphs, matrices, and designs,* Lecture Notes in Pure and Applied Mathematics 139, pages 269–278. Dekker, 1993.
17. Michael Sampels. Large networks with small diameter. In Rolf H. Möhring, editor, *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG '97),* LNCS 1335, pages 288–302. Springer-Verlag, 1997.
18. Michael Sampels. *Algebraic Construction of Efficient Interconnection Networks (in German: Algebraische Konstruktion effizienter Verbindungsnetzwerke).* Dissertation, University of Oldenburg, Germany, 1998.
19. Michael Sampels and Mykhailo Vilents. Symmetric interconnection networks in the design of switches for WANs and LANs. In D. Baum, N. Müller, and R. Rodler, editors, *Proceedings of the 10th GI/ITG Special Interest Conference on Measurement, Modelling and Evaluation of Computer and Communication Systems (MMB '99),* pages 43–48. University of Trier, 1999.

RDMA Communication Based on Rotating Buffers for Efficient Parallel Fine-Grain Computations

Adam Smyk[1] and Marek Tudruj[2]

[1] Polish-Japanese Institute of Information Technology,
86 Koszykowa Str., 02-008 Warsaw, Poland
[2] Institute of Computer Science, Polish Academy of Sciences,
21 Ordona Str., 01-237 Warsaw, Poland,
{asmyk, tudruj}@pjwstk.edu.pl

Abstract. The paper presents a new method for efficient fine grain computations on distributed memory computers. *RDMA* (Remote Direct Memory Access) communication is applied which assures direct access to memories of remote processing nodes. To obtain high *RDMA* efficiency for fine-grain computations with very frequent transmissions of small size messages, a specially designed structure of *RDMA* rotating buffers (*RB*) is introduced. It allows to fully exploit available communication bandwidth by provision of a special communication control infrastructure prepared and activated in a program before effective computations start. As an example of a fine-grain problem implemented with the *RDMA* rotating buffers, the discrete Fast Fourier Transform *(FFT)* execution is presented. "The binary-exchange algorithm" of *FFT* is examined showing efficiency of the *RB* method in comparison to standard *MPI* communication.

1 Introduction

Very intensive and short message volume communication is a basic and the most representative feature of fine grain parallel algorithms. It makes that their efficient implementation requires a very efficient network system for data transmissions. Such a system has to be characterized by possibly minimal communication latency. Otherwise, the total program execution time might easily overpass that of sequential execution due to huge communication overhead. This phenomenon can be alleviated or even completely eliminated through applying appropriately designed hardware architectures [1]. At a higher abstraction layer and only in the case of clusters of workstations, improvement could be obtained by replacing less efficient networks like FastEthernet by much faster (even up to 20 times) and reliable networks Myrinet [2] but much more expensive at the same time. A more innovative solution can be also proposed in which standard communication solutions are personalized to specialized interfaces. They should enable full communication efficiency through implementation of transmission based on specific hardware features. A good example of such approach is the use of RAM

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 50–58, 2004.
© Springer-Verlag Berlin Heidelberg 2004

memory slots in workstations and personal computers to organize an extremely fast interface for inter-processor communication called DIMMnet [3]. Another solution for efficient parallel fine-grain computations is multithreading based on shared memory. It delivers smaller communication overhead and is characterized by the lowest implementation cost, but very often it needs additional control communication because of read/write synchronization requirement.

To achieve reduced fine grain communication overhead in distributed memory MIMD systems, programming at a possibly low-level can be applied which offers direct access to local memories of remote computational nodes. On the Hitachi SR2201 supercomputer, it can be accomplished by the use of RDMA communication. RDMA facility enables writing data to a remote node memory without any buffering by an operating system. That significantly shortens communication latency in comparison with message passing through standard communication libraries. However, there are no confirmations from receiving side concerning the reception completion. If sending appears too early, an unintentional memory overwriting on a receive node can happen. In standard MPI libraries, synchronization between communicating nodes is assured by send and receive functions which impose hidden supplementary communication. In the RDMA model, additional synchronizing instructions are required. Obviously, this synchronization increases time overhead related to the message passing communication. In the RDMA transmissions, control parameters for data transfers can be prepared online by execution of several control instructions or the parameters can be prepared in advance in a transmission control word TCW stored for iterative reuse. In both cases, no automatic read/write synchronization is provided.

In the paper we propose a new solution for the RDMA communication based on specially designed control infrastructure at the send and receive sides which provides automatic read/write synchronization. For a communication scheme of a computational algorithm this control infrastructure is prepared in advance on all computational nodes involved. It is composed of a number of buffer areas (rotating buffers) designed for data exchange and necessary synchronization. The buffer areas together with some pointer and operation environment are organized in the operating memory used by communication control of application programs. Data send operations are interleaved with synchronizing confirmations with the time stride which depends on the number of buffer areas used. Comparing the MPI library and the standard RDMA, the proposed solution provides speedup of fine-grain computations of about 6 and 2 times, respectively. Access to the buffers is controlled by periodically changing pointers that choose buffers in strict rotation.

This paper is composed of three parts. In the first part, the standard RDMA mechanism is described. In the second part, the idea of the RDMA rotating buffers is presented. In the third part, results of experiments (FFT "Binary-exchange algorithm")[5] that involve different parameters of the proposed RDMA solution are presented and discussed.

2 Remote DMA Communication

Hitachi SR2201 is a MIMD distributed memory multi-processor system that can contain from 8 to 2048 RISC processors. Processors communicate through fast crossbar switches. Standard parallel programming is based on communication libraries like MPI and PVM but also on the native Remote Direct Memory Access (RDMA) facility is available for programming in C and Fortran languages [4]. The idea of the RDMA communication, is based on the combination of the remote memory DMA write mechanism with the virtual address translation. It assures that application program data used in transmission are directly mapped into operating memory physical areas used by RDMA communication instructions. In this way, RDMA facility excludes buffering of transferred messages by the operating system by which its transfer rate for large messages can achieve 280 MB/s. To organize a RDMA transmission, all transmission parameters and some control infrastructure have to be prepared at the send and receive sides by execution of control instructions. A user first has to create RDMA objects that are reserved by the operating system, contiguous address spaces in the operating physical memories at the send and the receive sides. Then, mapping of send and receive objects into so called RDMA areas in virtual memory address space is performed. The mapping is done modulo virtual memory page size. This mapping will be obeyed by the virtual memory page fault mechanism when data to be transferred are referenced in the user program. Inside a RDMA area at the receive side, the user has to create one or many receive fields. The identifier of a field will be used by processes at the send side to specify the transmission. Before sending can be performed, the send side has to obtain the transmission authorization from the receive side which is issued only when the receive control infrastructure for a specified field has been prepared. At the send side, the parameters furnished by the control operations, including the processor numbers, are stored in a data structure called a Transfer Control Word (TCW), which is used by the effective send instruction. In the standard RDMA communication mode, all reserved areas and the TCW are cancelled after transmission completion. However, for iterative transmissions that are based on the same parameters, the TCW word can be created in advance by a special control instruction and next used for many transmissions by send instructions with the TCW reuse. Each RDMA transmission is transparent to processor and it is performed by communication co-coprocessor NIA (Network Interface Adapter). The communication will start when the newly created TCW is written into control registers of the NIA. To avoid data incoherency between cache and local memory, the NIA reads and writes data only from local memory (it pass over cache memory). If it is needed, on the receive side cache memory will be purged on demand of the NIA. The RDMA facility does not provide any automatic notification of the receiver nor the sender process about completion of the receive action. The receiver can check the receive completion but The sender process can be notified about the receive completion by an additional reverse communication that has to be programmed at the receive side. Such reverse communication is used when a single field is shared at the receive side by many different processes.

3 RDMA Rotating Buffers Control

The rotating buffers RB method is a special mode for using RDMA facility in application programs. It enables to include completely synchronized control of remote memory access into very efficient RDMA communication. It is based on periodical access to specially designed control infrastructure organized in application program which intervenes in the RDMA transmissions.

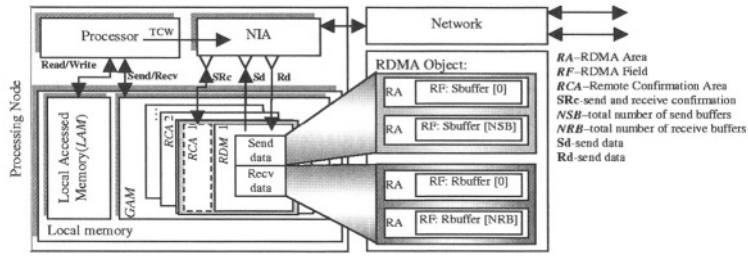


Fig. 1. Memory structure in rotating-buffers method (for one processing node)

The whole operating memory (Fig.1) available to one processing node can be seen as divided into two parts: the part with local memory access (LAM - Locally Accessed Memory) and the part with remote memory access (GAM - Globally Accessed Memory). Data stored in LAM are used only for local (one node) computations. Data from the GAM are either intended to be sent to another processor node or are just received from another node. GAM area is divided into pairs: RDM (Remote Data Memory) and RCA (Remote Confirmation Area), each pair organized to support communication of the given node with another remote processing node. Each of the RDM and RCA areas has been earlier declared as an independent (it posses its own identifier ID) RDMA object. A RDM is generally composed of many adjacent RDMA areas (4 KB per each). The whole RDM is additionally divided into send and receive parts. The numbers of RDMA areas in the send and receive parts are fixed and denoted by NSB and NRB, respectively. To receive data, inside each of the RDMA areas in the receive part, one RDMA field is created with a unique access code (Key). The total number of fields in a RDM equals NRB. Only one remote processor can have access to such defined receive part of RDM, which can be unambiguously specified by combination of two numbers: the ID and the Key. Before sending (writing) data to a remote memory field, each processor has to acquire a special permission from receiver that is the owner of this memory. It is done by sending to the remote processors requests (on for each Key) with the ID and the Keys of all involved receive fields. If on the receiving side the object specified by the ID exists and additionally, it contains the fields with access codes equal to Keys then the memory owner (receiver) sends back permission to access the requested parts of the memory. Then, the transmission, between the sender and the owner of memory can be executed. To facilitate access to RDMs, special arrays of

references are defined whose elements indicate consecutive areas (buffers) in RDMs (for each RDM separately). There are two separated arrays which define access to each area from the send (Sbuffer) and the receive parts (Rbuffer). The capacity of each buffer pointed at by Sbuffer and Rbuffer is equal at least 4KB. The last 32 bytes in each buffer are reserved for the fast RDMA communication which requires additional memory space to hold local confirmations of sending and receiving data. The set of buffers from a RDM area is used for data exchange and it needs an additional control to avoid data overwriting. This control is based on the RCA areas. To each RCA a set of buffers is assigned. A RCA is used to send and receive additional control messages which determine that the set of buffers of a RDM is ready to receive data. The inner structure of a RCA is similar to that of a RDM, except that the numbers of buffers NSB and NRB are usually greater than 1 for RDMs, while in the case of the RCA both of them are equal to 1. These two buffers in a RCA area are used to send confirmations only between two processors. To enable data exchange between any pair of two processors, the described above structure has to be built on each send and receive sides.

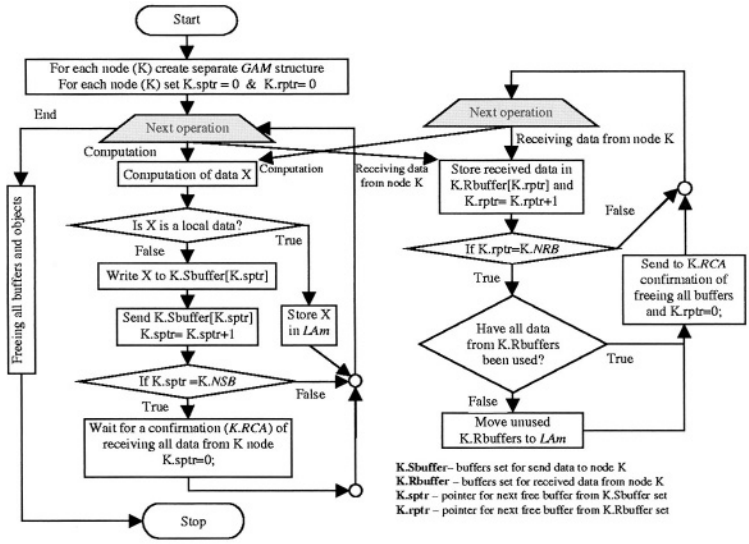


Fig. 2. Control flow of the rotating-buffers method (for one processing node)

The control flow in the RDMA rotating buffers method for single processing node (local node) is presented in Fig.2. The local node sends or receives data from the node K. After creation of the described above control and communication infrastructure, two (for each processor) additional pointers K.sptr and K.rptr are created. These pointers indicate a next free buffer where new data (to be sent to K) can be placed (K.sptr) or where new data just received from K will be written (K.rptr). The pointers K.rptr and K.sptr determine a rotating access

to a specific and not busy buffer (without any valid data) which can be used for next data transmission. They introduce periodical synchronization between two communicating nodes. It assures that no data which are transferred from one node to another, will be lost as a result of memory overwriting. To make such a synchronization possible, some conditions have to be met. For two communication nodes A and B, the values (A.NSB and B.NRB) and also (A.NRB and B.NSB) have to be pairwise equal. It allows to avoid deadlock, which otherwise would occur and it causes that the synchronization time for two processors is minimal.

4 Experiment Results

The RB method was implemented as a set of functions, for which a C++ class is formed. These functions enable creation of the RDMA control memory infrastructure for different numbers of processors. It was assumed, that data exchange can be done between all processors. To send data to a given processor, three RDMA communication functions (standard send, TCW send and TCW fast send) can be used. Depending on the sending function, a message can be received by `combuf_block_wait` or `combuf_spin_wait` function. The remote memory access synchronization is hidden inside the class. It is implemented with the introduction of additional messages (confirmations), which determine that the receiver is ready for new data from the sender. The RB method has been examined for the FFT computation with the use of Binary-Exchange algorithm (BE) [5]. The results obtained for the RB method have been compared with those when the MPI standard communication library was used. The efficiency of the RB method has been analyzed for different numbers of buffers (from 1 to 300). Depending on the number of buffers, obtained results have been split into 3 groups (and averaged): from 1 to 9 buffers - A-SNB (average for small number of buffers), from 10 to 50 buffers - A-MNB (average for medium number of buffers) and from 100 to 300 buffers - A-LNB (average for large number of buffers). All experiments has been carried out on the MIMD supercomputer Hitachi SR2201 with 8 computational nodes.

In Fig.3, speedup for the MPI and for the RB method with standard send RDMA function compared to standard RDMA (without RB) is presented. A Standard send RDMA function has much lower efficiency than the TCW reuse function on which the MPI implementation is based. That is why the MPI strongly speeds up communication (~10 times) especially for large problems while with the RB method with slow communication function merely the speedup of ~2.0 has been achieved. The use of standard send RDMA with the RB method for fine grain computation is not recommended.

Efficiency of the RB method increases significantly when much faster the RDMA communication based on TCW is applied, Fig.4. The MPI communication efficiency is lower about ~2.0 to ~3.4 times compared standard RDMA (without RB) with TCW. In the case of the RDMA RB method the speedup for A-MNB configuration is almost two times better in comparison to the standard RDMA. For A-SNB and A-LNB the slightly smaller (10-15%) speedup can

be observed. However, for very small problems it can fall even below ~1.0 (for A-LNB).

The efficiency of the RB method based on TCW communication is significantly better compared MPI - in the range ~3.0 to ~6.0. The best speedup ~6.0 was achieved for the A-MNB. This configuration again turned out to be the most efficient. Efficiency for A-SNB and A-LNB is visibly lower (~3.0-~5.0) and for the medium sizes of FFT problem they are almost equal each to other. However, when the problem size increases, the A-SNB efficiency falls to ~2.9. It can be explained by the significant number on confirmations which are generated in the case of A-SNB configuration with the total large number of communications. For small number of buffers confirmation transmissions are numerous and they increase total program execution time. However, in the case of the A-LNB configuration, we can suspect that the lower efficiency compared with the A-MNB results from the very large number of packets which are very frequently generated and sent to NIA overpasses the NIA throughput. It results in network congestion and reduces efficiency from ~4.7 to ~3.6.

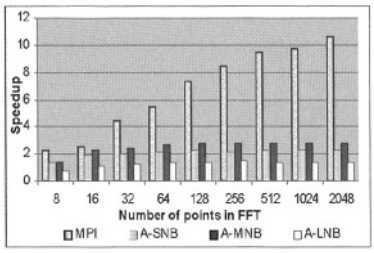


Fig. 3. Speedup for *MPI* and *RB* method based on *Standard send RDMA* communication versus standard *RDMA* for various numbers of buffers

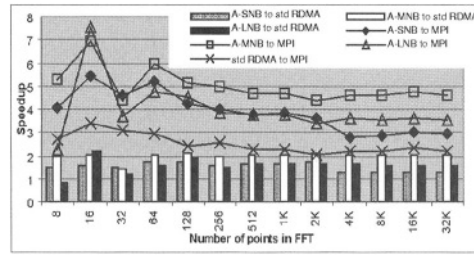


Fig. 4. Speedup for *TCW RDMA RB* communication versus *MPI* and standard *RDMA* for various numbers of buffers

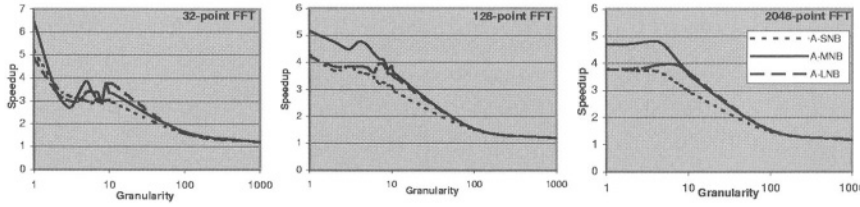


Fig. 5. Comparison of speedups for 32-point, 128-point and 2048-point *FFT* graphs in terms of granularity

In next experiment, the RB method sensitivity to parallelism granularity of problems was examined. The tests were performed for the same communication scheme as in the FFT BE case, but with a variable volume of computations. The

granularity can be expressed by the ratio (the volume of data to be sent)/(the volume of data to be computed). For example, the granularity equals to 10 if for each 10 computed data, only one of them will be sent to another processor. The tests were performed for three different sizes of the FFT: 32-point, 128-point and 2048-point. The results presented in Fig.5 confirm that the efficiency of the RB method depends on the communication intensity. The biggest speedup (comparing to MPI), was obtained for the finest computations (granularity ranges from ~1 to ~8) and it was equal to about ~5.0. When granularity increases, the speedup falls to ~1.0. For very coarse grain problems, the efficiency of the RB method is almost the same as for the MPI communication. It results that the RB method performed on SR2201 system behaves well for small grain of computation ie. when the granularity is smaller than 10. It can be also observed that the efficiency of the RB method is almost constant for different problem sizes. It is mainly dependant on the configuration (the number of buffers) of the RB method than on the problem size (for problems bigger than 128 points). The efficiency of the RB method depends rather on the control infrastructure configuration - the number of buffers than on the problem size.

5 Conclusions

In the paper, the RDMA communication based on rotating buffers (RB method) has been presented. It turned out to be a very efficient mechanism for fine grain communication with remote direct access to memory in MIMD machines (Hitachi SR2201). In comparison with standard MPI communication library, the RB method is characterized by much shorter communication time, especially in the case of fine grain computations (frequent communication of short messages) to a large degree decides on its efficient execution. The maximum speedup which is achieved by the use of the RB method is equal to about ~6.0 in comparison with the message passing with MPI. It directly depends on the configuration of send-receive control infrastructure (the number of buffers) and on intensity of sending synchronization messages. For fine grain computation the number of synchronization messages that occur in the RB method (dependent on the number of buffers use) should be properly tuned depending on the intensity of communication in application program not to exceed the maximum throughput of the processor communication controller (NIA). Efficiency of the RB method depends on the parallel distribution of tasks in the application programs maintains the fain grain character of execution of each involved processor.

This work was sponsored by the KBN Grant N. 4T11C 007 22 and by the internal grants of the PJIIT.

References

1. M.Tudruj, L.Masko: *Communication on the Fly and Program Execution Control in a System of Dynamically Configurable SMP Clusters,* 11-th Euromicro Conference on Parallel Distributed and Network based Processing, Genoa - Italy, February, 2003, IEEE Computer Society Press, pp. 67 - 74.

2. A.Smyk, M.Tudruj: *OpenMP/MPI programming in a multi-cluster system based on shared memory/message passing communication,* LNCS 2326, Springer Verlag, Advanced Environments, Tools and Applications for Cluster Computing, Mangalia, 1-6 Sept. 2001, pp. 241 - 248.
3. N.Tanabe, Y.Hamada, H.Nakajo, H.Imashiro, J.Yamamoto, T.Kudoh, H.Amano: *Low Latency Communication on DIMMnet-1 Network Interface Plugged into a DIMM Slot,* International Conference in Parallel Computing in Electrical Engineering PARELEC 2002, 22-25 September , Warsaw, pp. 9 - 14.
4. Hitachi Ltd: *HI-UX/MPP - Remote DMA -C- User's Guide Manual Number: 6A20-3-021-10(E),* Second Edition: January 1997
5. A.Grama, A.Gupta, G.Karypis, V.Kumar: *Introduction to Parallel Computing,* Second Edition, 2003 Addison Vesley.

Communication on the Fly in Dynamic SMP Clusters – Towards Efficient Fine Grain Numerical Computations

Marek Tudruj[1,2] and Lukasz Masko[1]

[1] Institute of Computer Science, Polish Academy of Sciences
ul. Ordona 21, 01-237 Warsaw, Poland
[2] Polish-Japanese Institute of Information Technology
ul. Koszykowa 86, 02-008 Warsaw, Poland
{tudruj, masko}@ ipipan.waw.pl

Abstract. The paper presents new architectural solutions for parallel systems built of run-time configurable shared memory processor clusters. The proposed architecture enables run-time switching of processors between clusters combined with parallel data transfers to processors that is called communication on the fly. Programs are executed according to cache controlled macro data flow paradigm. An extended macro-data flow graph representation is introduced that includes modeling of program execution control in the system. Programs can be optimized based on decomposition onto dynamic SMP clusters and the use of communication on the fly. Simulation results show potential of the proposed system architecture for execution of fine grain numerical parallel programs.

1 Introduction

The paper concerns efficiency of data exchange between processors in shared memory systems. Scalability of shared memory systems can be improved by application of the processor cluster-based system architecture. To adjust parallel program organization to system structure, concentrations of intensive interprocess communication in programs should be mapped into shared memory clusters. Systems based on shared memory processor clusters are very popular [1,2, 3,4,5,6,7,8,9,10]. However, in current implementations the size of processor clusters is fixed. We present a shared memory system architecture based on run-time reconfigurable shared memory processor clusters [12,13]. The dynamic cluster reconfigurability brings dynamic adjustment of system structure to program needs (optimal computation and communication mappings) but also it enables a new type of communication between clusters that we propose. It is called communication on the fly. It is composed of two mechanisms. The first one is dynamic switching of processors with data cache contents between SMP clusters. This is combined with the second mechanism, which consists in multiple parallel reads of data by many processors while they are displayed in the cluster, similar to cache injection proposed in [11]. The SMP cluster architecture is based on busses which enable collective data reads and snooping of information while being written to

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 59–68, 2004.
© Springer-Verlag Berlin Heidelberg 2004

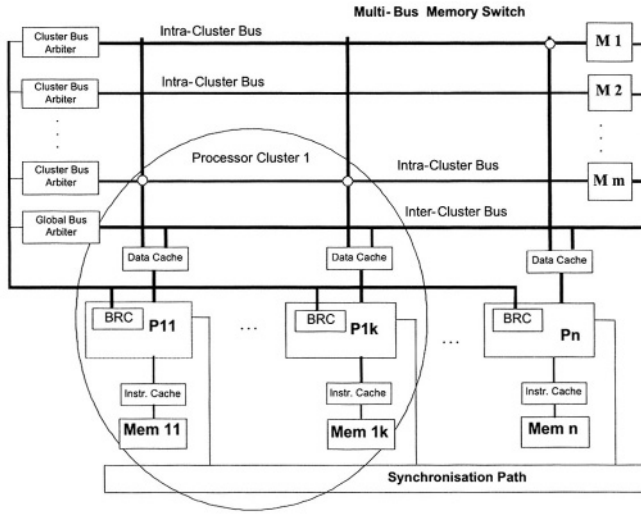


Fig. 1. General structure of the system

memory. Communication on the fly is very efficient. It replaces data transfers through global inter-cluster means by fast dynamic cluster reconfiguration with parallel writes of data brought in a processor's cache to data caches of processors in the target cluster. In complicated programs, the communication on the fly can give much better results than cache injection or processor dynamic switching between clusters applied separately. Efficient program execution can be obtained for fine-grain parallel computations and for the coarse-grain as well. This has been examined by simulation experiments based on a typical numerical example of matrix multiplication. The experiments were performed using a new graph program representation that enables adequate specification of program execution control and program graph structuring imposed by the paradigm of communication on the fly. A macro-data flow program execution model controlled by the data cache stability, additionally governs program design. Such model eliminates data cache thrashing and speeds up program execution.

The paper is composed of three parts. In the first part, features of the proposed system architecture are described. In the second part, a new graph representation for program execution control in the proposed architecture is discussed. In the third part, the new program execution strategy is explained on the example of matrix multiplication and results of simulation experiments are presented.

2 Dynamic SMP Clusters and Communication-on-the-Fly

The general structure of the proposed system of reconfigurable SMP clusters is shown in Fig.1. The processors (Pij) have separate memory modules for data (Mi) and instructions (Mem ij). All data memory modules are placed in a common address space shared by all processors. To belong to a SMP cluster, a pro-

cessor has to be attached to one Intra-Cluster Bus dedicated to a data memory module. All processors in the system are also connected to the Inter-Cluster Bus through which they can read data from any data memory module. Programs are divided into tasks that fulfil cache controlled macro data flow principle. Before a task starts being executed, the processor data cache has to be filled with all necessary data by Cache Pre-fetch instructions. Tasks are so defined as not to over-flow data cache volume. During task execution the data cache is not reloaded. A processor can fill data cache from a memory module via the Intra-Cluster of its cluster or via Inter-Cluster Bus. Data read from a non-local memory module are copied in the cluster memory module and in the processor data cache. Computation results are written to the data cache. After completion of a task, the results can be written back to the memory of the processor's cluster. All data in memory modules are provided with availability bits (with values Ready/Empty). They provide synchronization of reads and writes to variables. A new address is used when data modified by a task are useful for tasks in other processors. This single assignment principle together with the specific caching strategy prevents the memory/cache consistency problem and eliminates data cache thrashing. Each processor has a Bus Request Controller (BRC), which controls program?s access to data busses (memory modules). When a program writes or reads data through a bus to a memory module, other processors connected to the bus can simultaneously read the data on the fly from the bus to their caches. To be sure that all relevant processors can read data on the fly, all processes (including the writing one) have to be synchronized by a barrier. It is fulfilled when all reading processors have put read requests to the bus snooping table in their BRCs. The Synchronization Path performs the hardware synchronization of states of processes performed in processors (barriers). Switching of a processor between clusters is done in two steps, under control of bus arbiters. If processors read data that are written by a processor newly switched into a cluster, then the processor brings in its cache new data for the cluster. We call this data transfer paradigm communication on the fly. Programs can generate for BRCs write requests and 3 types of read requests: standard, standard/on the fly and synchronized on the fly. Reads on the fly and synchronized reads on the fly are based on reading data on the fly from a data bus snooped by the BRC. Standard/on the fly read requests and synchronized read requests are kept in a bus snooping table of the BRC. The BRC snoops addresses visible on the inter-cluster and intra-cluster bus. When the address on a bus is equal to the source address of a request stored in the table, data from the bus are sent to the processor?s data cache. If a read request from the table cannot be executed to the end because the transfer on a bus was shorter than requested, the BRC modifies the request to enable the rest of data to be read. All requests have priorities that depend on volume of data handled. Writes have higher priority than reads. For more details concerning the architecture see [12,13].

3 Program Representation Using an Extended Macro-Data-Flow Graph

An application program is first represented as a macro-data-flow graph. To describe behavior of processors, data caches, memory modules, busses and bus arbiters, new kinds of nodes are introduced: memory read nodes to the data cache (R), write nodes from processor's data cache to memory (W), intra-cluster memory bus arbiter nodes (CA), the inter-cluster global memory bus arbiter node (GA), processor switch nodes (Mi) and barriers (Bi). Read and write node weights correspond to volumes of data.

An example of the proposed extended macro-data flow program graph (EMDFG) for a macro-data flow graph (MDFG) of a node T4 that receives data through an Intra-Cluster Bus and the Inter-Cluster Bus from tasks T1, T2, T3, in processors that belong to the same and different clusters is shown in Fig.2. An arbiter node can be connected by edges with many memory read and write nodes. It scans the nodes and activates that one which has the highest priority. The selected node executes and sends the token back to the arbiter. To represent synchronized reads on the fly, each read node can be decomposed into two nodes: a read request issued to a BRC performed before the barrier and the read execution performed when the barrier is reached. A special node, labeled with the memory module identifier, represents switching of a processor to a new processor cluster. Communication on the fly can be represented as in Fig.3. Synchronized read requests are not serviced by bus arbiters contrary to write nodes.

In communication on the fly, processors can read parts of data on a bus. Labels of read nodes determine read starting times. A section in a program graph is a sub-graph executed by a fixed set of processors connected to the same

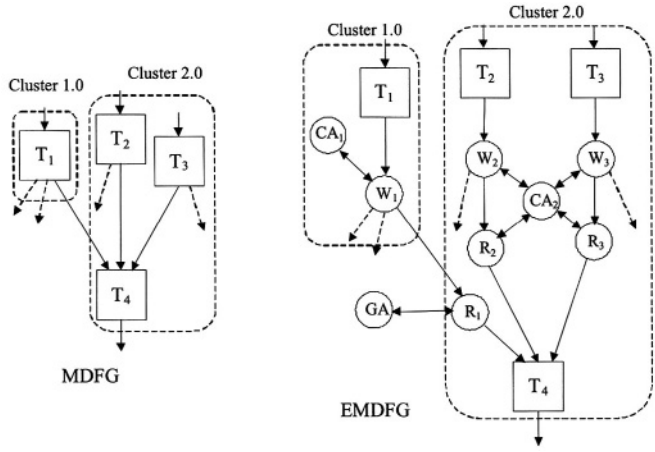


Fig. 2. Extended macro-data flow graph for multiple computational nodes

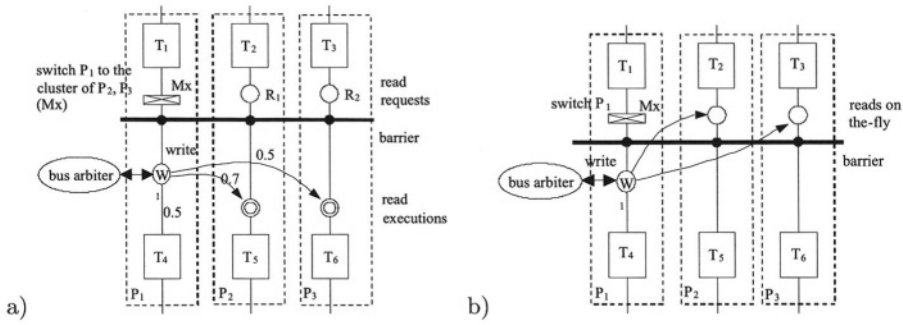


Fig. 3. Communication on the fly a), simplified graph notation b)

memory bus (cluster). Processors are notified about new sections to activate all relevant program parallel threads. Sections enable adjustment of memory read and write requests to current composition of clusters.

4 Simulation Results for Matrix Multiplication

We will show the use the proposed system architecture for square matrix multiplication $AxB = C$ with recursive decomposition of matrices into quarters (A_{ij}, B_{ij}, C_{ij}, $i,j \in \{0,1\}$). The EMDFGs of the algorithm are shown in Fig.4. At the 1st recursion level, the algorithm includes 8 multiplications: $M_1 = A_{00} \times B_{00}$, $M_2 = A_{01} \times B_{10}$, $M_3 = A_{00} \times B_{01}$, $M_4 = A_{01} \times B_{11}$, $M_5 = A_{10} \times B_{00}$, $M_6 = A_{11} \times B_{10}$, $M_7 = A_{10} \times B_{01}$, $M_8 = A_{11} \times B_{11}$. Four additions $AD_i = M_{2i-1} + M_{2i}$, $i = 1, \ldots, 4$, furnish quarters of the result C_{00}, C_{01}, C_{10}, C_{11}, respectively. At the 2nd recursion level, each multiplication M_i will be replaced by the entire graph from level 1. To speedup additions and reduce communication for fine-grain computations, each computed matrix M_i at the 1st recursion level is divided into two half-matrices M_{iL} (left) and M_{iR} (right). M_{2iL}, M_{2i-1L} and M_{2iR}, M_{2i-1R} are added in parallel by AD_{iL}, AD_{iR}, as shown for $i = 4$ in Fig.5a. These additions are two times less costly in time than for full matrices. In the entire graph, they will provide halves of the quarters of the result that will be written to the memory by the write nodes. We will assign (M_{2i-1L}, AD_{iL}) and (M_{2iR}, AD_{iR}) to the same processors, so that half of data for addition are transferred through data cache. Thus, also communication for addition can be reduced by a factor of 2. At the 2nd recursion level multiplication results are also divided into halves and are added in parallel. The complete algorithm graph at the 2nd recursion level contains 16 elementary sub-graphs such as shown in Fig.5b. Such elementary sub-graph has been transformed to include reads on the fly and communication on the fly, as shown in Fig.6. The sub-graph has been structured to the use of 2 memory modules (M1, M2) - 2 dynamic processor clusters. After execution of sections M1.0 and M2.0, processors P7,2 and P8,1 are switched to opposite clusters to bring in their caches results

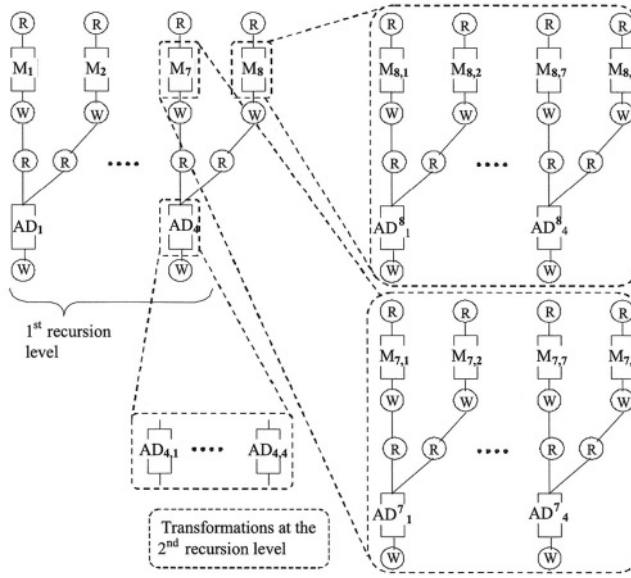


Fig. 4. EMDFG of matrix multiplication

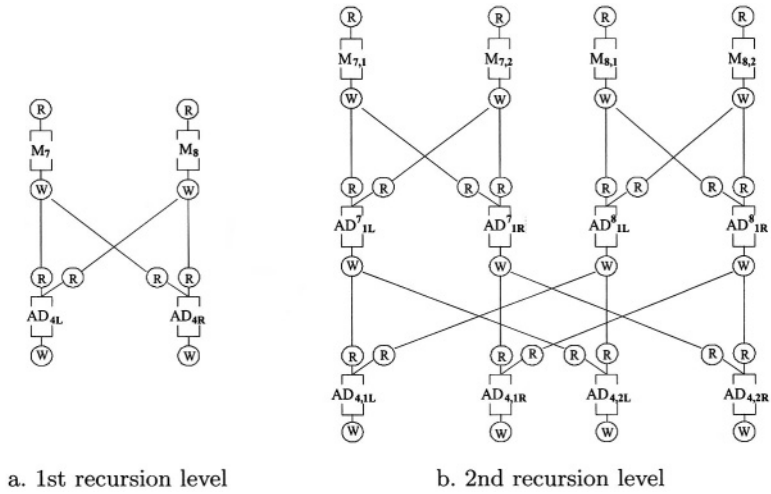


Fig. 5. Initial elementary EMDFG sub-graphs for matrix multiplication

used for further computations by other processors already existing in these clusters. There are 32 dynamic clusters each composed of 2 processors organized for execution of the total matrix multiplication graph.

Below we present results of symbolic execution of the discussed algorithm graphs for different configurations of programs (reads on the fly - ROF, proces-

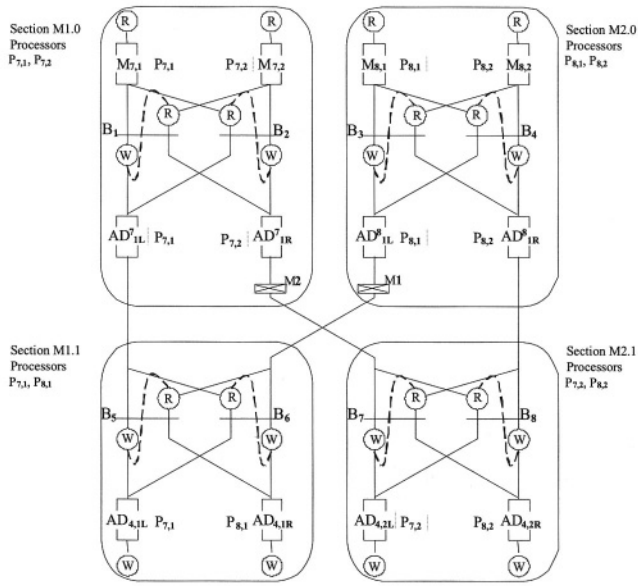


Fig. 6. Elementary sub-graph (2nd recursion level) with communication on the fly

sor switching - PS) and systems (number of clusters and processors in clusters). The assumed operation execution times were: 1 for floating point 8-byte scalar addition and multiplication, 6 for transmission of the scalar between data cache and memory (600 MFLOPS processors working with 100 MHz memory busses). Table 1 presents communication time and speedup evaluated for different program/system configurations against execution on 64 processors with a single bus.

For execution with low parallelism on 8 processors on a single bus with ROF communication speedup was not higher than 3. With 4 2-processor clusters with ROF communication speedup was from 10 to 7. For high level of parallelism, the use of 16 4-processor clusters with ROF gave communication speedup of 23.7 - 31.3. More fine distribution of communication to 32 clusters based on 2 processors has brought communication speedup of 13.4 - 14.7 without ROF and PS. The speedup was only in the range of 13.5 to 10 with ROF but without PS. It was due to high inter-cluster traffic through the global bus while economies due to local ROFs were relatively small. The situation was much better when PS without ROF was applied. The obtained speedup was from 37 to 27. That confirms efficiency of processor switching, especially for big number of processor clusters. The highest speedup, in the range of 39.5 to 29, was obtained when both PS and ROF - i.e. communication on the fly were applied. The speedup was the highest for the finest grain of computations. It confirms suitability of communication on the fly in this case. With communication on the fly applied to matrix multiplication there are no data reads through busses, except for initial data reads and final writes. All data communication is done by reads on the fly,

Table 1. Communication time and speedup versus execution on a 64-processor cluster

Configuration				Matrix size					
Clusters	Processors per cluster	Reads on–the–fly	Processor switching	32	64	128	256	512	1024
1	64	No	No	121872	483392	1900800	7341056	27267072	92291072
1	8	Yes	No	41600 (2.93)	159744 (3.03)	638976 (2.97)	2555904 (2.82)	10223616 (2.67)	40894464 (2.26)
4	2	Yes	No	12288 (9.92)	49152 (9.83)	196608 (9.67)	786432 (9.33)	3145728 (8.67)	12582912 (7.33)
16	4	Yes	No	5136 (23.76)	16448 (29.39)	60672 (31.33)	242688 (30.25)	970752 (28.09)	3883008 (23.77)
32	2	No	No	8512 (14.44)	36097 (13.39)	133120 (14.77)	532480 (14.77)	2129920 (14.77)	8519680 (14.77)
32	2	Yes	No	9025 (13.50)	36097 (13.39)	144385 (13.16)	577537 (12.71)	2310145 (11.80)	9240577 (9.99)
32	2	No	Yes	3265 (37.33)	13057 (37.02)	52225 (36.40)	208897 (35.14)	835585 (32.63)	3342337 (27.61)
32	2	Yes	Yes	3078 (39.59)	12294 (39.32)	49161 (38.66)	196617 (37.34)	786441 (34.67)	3145737 (29.34)

communication on the fly or data transfers through caches. It eliminates 60% of bus transactions - only writes remain.

Table 2 presents total program execution times and speedups versus execution on a single processor. It shows how communication improvements enter into synergy with computation parallelization. Execution with low level of parallelism (at the first recursion level) - with 8 processors configured in a single cluster or 4 clusters gives small speedup in the range of 1,79 to 7,73, depending on the granularity level. With high level of parallelism, the second recursion level is applied to programs distributed among 64 processors. With 16 clusters, each of 4 processors, and ROF applied we obtain high speedup of 14.5 to 58. Inter-cluster communication does not appear in this case and reads on the fly inside clusters are efficient. 64 processors on a single bus (cluster) give speedup up to 17 for coarse grain programs (large matrix sizes) but bad efficiency for fine grain cases. Distribution of execution between 32 clusters, each based on 2 processors without ROF nor PS gives speedup from 9.3 to 51.6, the high value for coarse grain. Inclusion of ROF without PS gives slightly smaller speedup from 9 to 50. It is due to barriers that synchronize pairs of processors in clusters and sometimes produce conflicts in accessing shared busses. Further speedup improvement is obtained when PS is applied. This eliminates the use of the global bus for inter-cluster communication. Without ROF speedup is from 20.8 to 58.8. When both ROF and PS are applied i.e. communication on the fly is used, the speedup is the highest, between 21.8 and 59.2. In this best configuration, speedup for coarse grain computations was 2.7 times higher than for the most fine grain computations. We have checked the impact of the relation of processor computational

Table 2. Execution time and speedup versus sequential execution

Configuration				Matrix size					
Clusters	Processors per cluster	Reads on–the–fly	Processor switching	32	64	128	256	512	1024
1	1	No	No	89088	618496	4571136	35061760	274464767	2171600896
1	64	No	No	122880 (0.73)	491520 (1.26)	1966080 (2.33)	7864320 (4.46)	31457280 (8.73)	125829120 (17.26)
1	8	Yes	No	49664 (1.79)	224768 (2.75)	1161216 (3.94)	6742016 (5.20)	43745280 (6.27)	309198848 (7.02)
4	2	Yes	No	20352 (4.38)	114176 (5.42)	718848 (6.36)	4972544 (7.05)	36667392 (7.49)	280887296 (7.73)
16	4	Yes	No	6144 (14.50)	24576 (25.17)	125952 (36.29)	765952 (45.78)	5160690 (53.18)	37427056 (58.03)
32	2	No	No	9520 (9.36)	41408 (14.94)	198400 (23.04)	1055744 (33.21)	6320128 (43.43)	42057728 (51.63)
32	2	Yes	No	10033 (8.88)	44225 (13.99)	209665 (21.80)	1100801 (31.85)	6500353 (42.22)	42778625 (50.76)
32	2	No	Yes	4273 (20.85)	21185 (29.19)	117505 (38.90)	732161 (47.89)	5025793 (54.61)	36880385 (58.88)
32	2	Yes	Yes	4086 (21.80)	20422 (30.29)	114441 (39.94)	719881 (48.70)	4976649 (55.15)	36683785 (59.20)

Table 3. Computation speedup for communication of the fly - 64 processors

Computation speed /communication speed	Matrix size					
	32	64	128	256	512	1024
6 : 1	21.80	30.29	39.94	48.70	55.15	59.20
3 : 1	30.12	39.88	48.68	55.14	59.20	61.49
1 : 1	44.96	52.70	57.75	60.70	62.30	63.14

speed against communication speed on this feature. The results are shown in Table 3. For the finest computation grain in the system of 32 2-processor clusters, the speedup was 30 when communication speed increased 2 times and 45 when communication speed and computation speed were equal (6-time increase of the communication speed). At the same time, the speedup for the coarse grain computations with this system configuration was approaching nearly 64.

5 Conclusions

The paper has presented the architecture of a multi-processor system with dynamic shared memory processor clusters based on busses. Communication on the fly composed of data transfers based on dynamic processor switching between clusters and data reads on the fly, is a powerful mechanism. It provides strong communication and program execution speedup due to elimination of

many transactions on data busses. This speedup is better than when reads on the fly and processor switching are applied separately for program execution.

Communication speedup obtained for fine grain computations was 35% higher than for coarse grain computations. It shows the potential of dynamic SMP clusters and communication on the fly for fine grain parallelism. This potential can be fully used if a proper relation between computation speed and communication speed is provided in the system. With communication on the fly and sufficiently fast communication between memory and processor data cache, matrix multiplication program execution speedup for very fine grain of parallel computations can achieve even 70% of the maximal speedup obtained for coarse grain computations. The introduced graph representation is currently further developed to enable automatic task clustering that will minimize program execution time.

This work was partially sponsored by the KBN Grant N. 4T11C 007 22.

References

1. J. Protic, M. Tomasevic, V. Milutinovic, A Survey of Shared Memory Systems, Proc of the 28th Annual Hawaii International Conference of System Sciences, Maui, Hawai, Jan. 1995, pp. 74 - 84
2. D. Sima, T. Fountain, P. Kacsuk, Advanced Computer Architectures; A Design Space Approach, Addison-Wesley, 1997.
3. Y. Kanaka, M. Matsuda, M. Ando, K. Kazuto, M. Sato, "COMPaS": A Pentium Pro PC-based SMP Cluster and its Experience, IPPS Workshop on Personal Computer Based Networks of Workstations, LNCS 1388, pp. 486-497. 1998.
4. Y. Kanaka, M. Matsuda, M. Ando, K. Kazuto, M. Sato, Performance Improvement by Overlapping Computation and Communication on SMP Clusters, Int'l Conference on PDPTA '98, Vol. 1, 1998, pp. 275-282.
5. "Pentium Pro Cluster Workshop", http://www.scl.ameslab.gov/workshops/
6. T. Ikedo, J. Yamada, Y. Nonoyama, J. Kimura, M. Yoshida, An Architecture based on the Memory Mapped Node Addressing in Reconfigurable Interconnection Network, 2nd Aizu Int'l Symp. on Parallel Algorithms/Architecture Synthesis, Aizu-Wakamatsu, March 1997, pp. 50-57.
7. "Scalable Clusters of Commodity Computers", http://www.csag.cs.uiuc.edu/projects/clusters.html
8. Multimax Technical Summary, Encore Computer Summary, March 1987.
9. D. Lenoski et al. The Stanford Dash multi-processor, IEEE Computer, Vol. 25, N. 3, 1992, pp. 63-79.
10. Convex Exemplar Architecture, Convex Press, 1994, p. 239.
11. A. Milenkovic, V. Milutinovic, Cache Injection: A Novel Technique for Tolerating Memory Latency in Bus-Based SMPs, Proceedings of the Euro-Par 2000, LNCS 1900, 2000, pp. 558-566.
12. M. Tudruj, L. Masko, Program Execution Control for Communication on the Fly in Dynamic Shared Memory Processor Clusters, Int. Conf. on Parallel Comp. in Electric. Eng., PARELEC 2002, September 2002, Warsaw, IEEE Comp. Society Press, pp. 15-20.
13. M. Tudruj, L. Masko, Communication on the Fly and Program Execution Control in a System of Dynamically Configurable SMP Clusters, 11-th Euromicro Conference on Parallel Distributed and Network-Based Processing, February, 2003, Genoa – Italy, IEEE Comp. Society Press, pp. 67-74.

Wait-Free Publish/Subscribe Using Atomic Registers

Pradeep Varma

IBM India Research Laboratory,
Block 1, Indian Institute of Technology, Hauz Khas, New Delhi 110016, India
pvarma@in.ibm.com

Abstract. We model wait-free publish/subscribe using multiple-reader, single-writer atomic registers as the sole inter-process communication primitive. The system supports dynamic publishers/subscribers, content-based matching, and durable subscriptions. Publication to *M* subscribers requires *O(M)* atomic register operations by the publisher and $O(1)$ similar operations by each subscriber. We also contribute: (a) A wait-free, constant-space, producer-consumer solution, that extends an earlier solution [1] to atomic reads/writes of *finite, N-valued* integers, where *N* is the queue size. (b) A conversion of our and earlier fixed-space queues into wait-free, variable-capacity, FIFO communication links which can increase and decrease space upon need from no traffic backlog to $O(N^K)$ backlog, while minimally consuming $\Omega(N)$ space. K is a programmable constant. (c) A proof of the correctness of our solution. We show that our system requires only boolean to *N*-valued, multiple-reader, single-writer registers, making it applicable in shared and distributed memories [2].

1 Introduction

Wait-free computing [3],[4],[5] is fault-tolerant computing approach that guarantees that each process can complete its operation on a wait-free object in a finite number of steps. Unlike concurrent implementations using critical sections where a process holding a critical section can block other processes indefinitely, a wait-free computing system guarantees that all non-faulty processes in the system can make progress. In this paper, we study a communication idiom, publish/subscribe [6],[7], from a wait-free computing perspective.

Publish/subscribe or pub/sub is multicast idiom between publishers and subscribers of common interest (see http://java.sun.com/products/jms/). A subscriber describes its topic of interest and publications that fit the topic are routed from publishers to the subscriber. Subscriber interests can be static in nature - well-defined subject headings - and they can also be content-based or dynamic in nature. Publishers and subscribers themselves can be dynamic, and can join and leave at any time. If durable subscriptions are supported, then temporary departures of subscribers are allowed. In this case, a subscriber gets to see all relevant publications including ones that occur during its temporary absence. We show that all these features of pub/sub systems can be provided in

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 69–76, 2004.
© Springer-Verlag Berlin Heidelberg 2004

a wait-free manner using the most elementary communication primitive of the asynchronous PRAM model - atomic registers [2], [8].

In [3], a hierarchy of objects in terms of their wait-free capabilities is constructed and it is shown that atomic registers have a consensus number of one, which puts them at the lowest-capability position in the hierarchy. By building wait-free pub/sub using only atomic registers, we establish that wait-free publish/subscribe can be built using any of the more powerful objects in the hierarchy. We address the expenses of wait-free computing [5] by carefully eliminating from our construction, costs such as copying, implicit (forced, failure based) serialization of updates, and memory contention that are present in [4]. [4] also uses more powerful primitives than atomic registers. In contrast to critical-sections (e.g. the lock-based, highly-concurrent queues in the tuplespaces of [9]), we also retain the efficiency motive of "truly concurrent" computing – maximizing parallelism and removing overheads (e.g. lock acquire/release).

2 A Wait-Free Producer-Consumer Queue

```
public class PCQueue [Type]{
private int bufsize;
private volatile int ppos = 0, cpos = 0;
private volatile boolean crossbackApart = false;
private volatile Type buffer[];
public PCQueue[Type](int b)
    {bufsize = b; buffer = new Type[bufsize];}
private int succ(int pos)  {return (pos + 1) % bufsize;}
private boolean cbehind(int sc, int p, boolean b)
    {return (sc <= p  &&  sc != 0) ||  b ;}
private boolean pbehind(int sp, int c, boolean b)
    {return (sp <= c  &&  sp != 0) || (! b);}

public boolean producer(Type data)
{ if (pbehind(succ(ppos), cpos, crossbackApart))
       {buffer[succ(ppos)] = data; ppos = succ(ppos);
        if (ppos == 0) crossbackApart = true; return true;};
  return false;}

public boolean consumer(Type * data)
{ if (cbehind(succ(cpos), ppos, crossbackApart))
      {* data = buffer[succ(cpos)]; cpos = succ(cpos);
       if (cpos == 0) crossbackApart = false; return true;};
  return false;}}
```

PCQueue is a wait-free queue, written in pseudo-code comprising Java extended with (a) type parameters, and (b) C-like pointers. A producer writes data into the queue using a producer pointer, ppos. Similarly, a consumer uses cpos. Unlike the unbounded head and tail pointers in [1], the pointers here acquire only a bounded

set of values, [0 ... bufsize-1] throughout the life of PCQueue. A boolean flag, crossbackApart, is used along with the pointers. Variables ppos, cpos, buffer[], and crossbackApart are labeled as volatile as they are read/updated by more than one process. Among the volatile variables, only ppos, cpos, and crossbackApart have atomic requirements. Buffer's non-atomic Type can well compile into multiple instructions for reading and writing.

Producer's method returns true if its data has been inserted successfully, else it returns false. The consumer method similarly returns true for a successful reading, with the read data returned by reference. Both the producer and consumer methods are wait-free - they always return after a finite number of steps regardless of the fault status of the other processes in the system. A failure returned by the methods suggests that the operation should be reattempted after some time. The producer produces data ahead of the consumer and can increase the gap between itself and the consumer by at most bufsize number of slots (any more would imply overwriting). The consumer can decrease the gap at best to zero (any less would imply reading unwritten data). Truth of predicate pbehind (similarly cbehind) means producer is behind its bound and can produce.

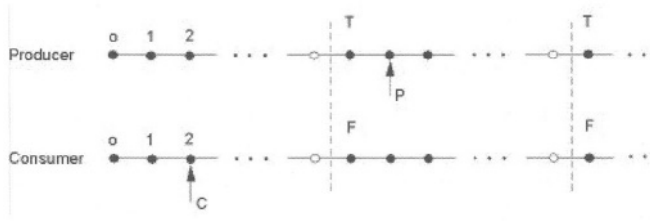


Fig. 1. Behavior of Producer, Consumer Pointers

The details of PCQueue are best explained using figure 1. Starting from the left, figure 1 represents the transitions undergone by ppos and cpos in PCQueue as time progresses. The transitions are counted uniquely (count not shown) while the domain of values taken by ppos and cpos remains [0 ... bufsize-1]. Each (large) dot in figure 1 represents a value of ppos or cpos $\in$ [0 ... bufsize-1], while each line connecting two adjacent dots represents a transition from one state to the next. Ppos dots fall in the producer half (top) of the figure while cpos dots fall in the bottom. The arrow P represents the number of preceding ppos transitions. P's value ranges from 0 to ∞. Similarly, C represents the number of preceding cpos transitions. The unfilled dots in figure 1 highlight the positions when ppos and cpos have the value bufsize-1. The transition succeeding each unfilled dot, highlighted by a (long) dashed vertical line, resets the ppos/cpos to 0. Each such transition is numbered some (not shown) multiple of bufsize and is referred to as a crossback. The dot succeeding a crossback is annotated with either T or F. This Boolean annotation does not represent any value of ppos/cpos, but rather represents the setting of crossbackApart to true or false. T represents the setting of crossbackApart to true, which always succeeds the setting of ppos

to 0 and precedes the setting of ppos to 1. Similarly, F represents the setting of crossbackApart to false. The properties of the queue's working are captured formally as follows. Proofs are often omitted due to space limitations.

Lemma 1. *P and C increase monotonically with increments of zero or one with each producer and consumer request.*

Lemma 2. *(i) A producer request cannot crossback or modify crossbackApart with a true reading of crossbackApart. (ii) A consumer request cannot crossback or modify crossbackApart with a false reading of crossbackApart.*

Lemma 3. *Consecutive C crossbacks or P crossbacks are not possible.*

Corollary 1. *At any time,* $|P - C| < 2\times$ *bufsize.*

Corollary 2. *C crossbacks and P crossbacks alternate.*

Lemma 4. *(i) Writes on crossbackApart are totally ordered in time, (ii) The producer and consumer write crossbackApart alternately.*

PROOF. Consider a P crossback followed by a C crossback. From Lemma 2, a crossbackApart = false reading precedes the P crossback. Only producer can set crossbackApart to true, and without a crossbackApart = true reading, the C crossback cannot occur (Lemma 2). Thus the crossbackApart = true reading for the C crossback must succeed the crossbackApart modification associated with the P crossback. Thus for a P crossback followed by a C crossback, the following events are sequentially ordered: producer reading of crossbackApart, P crossback, producer modification of crossbackApart, consumer reading of crossbackApart, C crossback, consumer modification of crossbackApart. A similar sequence of ordered events is straightforward to generate for any sequence of alternating P and C crossbacks (which are the only kind of crossbacks possible - Corollary 2). In the sequence of ordered events, both (i) and (ii) orders are present (by transitivity of the temporal order). •

Corollary 3. *Reads and writes on crossbackApart are boolean, single-writer, multiple-reader.*

PROOF. From Lemma 4. •

The multiple-reader, single-writer nature of other atomic variables is straightforward.

Theorem 1. *At all times,* $0 \leq P - C \leq$ *bufsize.*

PROOF. We only prove $0 \leq$ P - C here since the rest is symmetric. Since C and P are initially identical (zero), and both increase monotonically (Lemma 1), for C to be greater than P, there has to exist a C transition which causes

this. The transition is made by a consumer request which makes a reading of crossbackApart prior to the transition. There are two possibilities:

Supposition (i) Such a transition exists with a crossbackApart = true reading. By Lemma 4 and due to the initial value of crossbackApart being false, prior to this crossbackApart reading, there have to be an odd number of crossbackApart modifications (and crossbacks). Since crossbacks alternate (Corollary 2), and start with a P crossback (due to the initial value of crossbackApart), the number of P crossbacks is k + 1, where k is the number of C crossbacks when the true reading of crossbackApart is made. After the supposed transition, P = bufsize × (k + 1) + u where $0 \leq u <$ bufsize. The value of C depends on whether the transition causes crossback or not. If it is not a crossback transition, then after the transition, C = bufsize × k + v where $0 \leq v <$ bufsize, otherwise C = bufsize × (k + 1). In both cases, $C \leq P$ (i.e. $0 \leq P - C$), which contradicts the supposition.

Supposition (ii) the transition exists with a crossbackApart = false reading. Similar to (i), this can be shown to be contradicted. •

Theorem 2. *In PCQueue, data produced with the ith P transition is consumed with the ith C transition.*

Corollary 4. *PCQueue is a FIFO queue.*

Corollary 5. *PCQueue is a wait-free FIFO queue.*

3 Wait-Free Variable-Capacity Links

Below is a variable-capacity, unidirectional, FIFO communication link built using our producer-consumer queue. Table contains a series of producer-consumer queues containing unconsumed messages (messages in transit). Table itself is a producer-consumer queue with its data elements being the message queues. When a message queue fills out, the producer can either wait for space to be freed up in the same queue by the consumer, or add an empty queue to table using producerAddBuffer. Return of a new queue by producerAddBuffer signals successful addition of the new buffer to table. The producer must shift to the new buffer and produce messages to only this. Table itself is FIFO ordered, so consumption of messages through multiple queues remains FIFO straightforwardly.

Like producer, consumer has only one current working buffer. When consumer empties out the buffer, it has to check the possibility of the producer having shifted to another buffer in table. The return of a buffer different than the argument passed into consumerDelBuffer signals to the consumer that a shift from its current buffer (the argument) to the returned buffer is imminent. The consumer has to ensure that the argument buffer has indeed been emptied out prior to shifting to the new buffer. For this, the consumer has to consume from the argument buffer till it again returns empty. After this, the consumer can delete the argument buffer and shift to the new buffer. Immediate shift to the new buffer without re-emptying the old buffer is disallowed for the (rare) chance of consumer initially finding the old buffer to be "empty" when it is

not (sampling ppos = 0 and crossbackApart = false). Once a new buffer has been obtained, re-emptying the old buffer occurs without the above sampling possibilities, after which the old buffer can be deleted/shifted from safely.

```
public class Link [Type]{
private int bufsize, totalBufs;
private PCQueue[Object] table;
public Link[Type](int s, int bufs)
    {bufsize = s; totalBufs = bufs;
     table = new PCQueue[Object](totalBufs);}

public PCQueue[Type] producerAddBuffer(PCQueue[Type] currentPBuf)
    {PCQueue[Type] q = new PCQueue[Type](bufsize);
     return table.producer(q) ? q : currentPBuf;}

public PCQueue[Type] consumerDelBuffer(PCQueue[Type] currentCBuf)
    {Object * nextBuffer;
     return table.consumer(nextBuffer) ?
        ( PCQueue[Type] ) * nextBuffer : currentCBuf;}};
```

Our link, comprising of a maximum of totalBufs active buffers consumes a minimum of $\Omega(\text{totalBufs} + \text{bufsize})$ space and provides space for a maximum of O (totalBufs $\times$ bufsize) messages. If the maximum buffer space is desired to be larger, then the above can be generalized straightforwardly to a tree structure: a buffer of buffer of ... of buffers. For a K-level, N-ary tree, the minimum space consumed becomes $\Omega(K \times N)$, and the maximum messages supported becomes $O(N^K)$. Indeed the code given above is for a simple, 2-level tree.

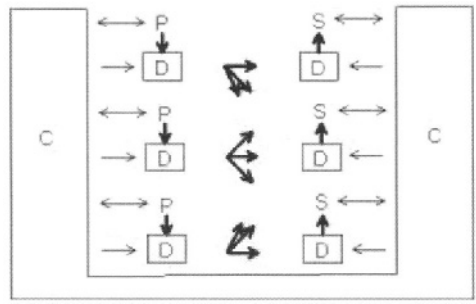


Fig. 2. Wait-Free Pub/Sub Architecture. C - Broker Central; D - Broker Desk; P - Publisher; S - Subscriber

4 Wait-Free Distributed Pub/Sub Architecture

Figure 2 shows our wait-free pub/sub model comprising of one centralized control process and decentralized broker-desk processes. Each pub and sub gets

a broker desk for itself. Our communication links are used between various processes as shown. Bi-directional links are made up of a pair of unidirectional links. Each publisher desk receives publications and uses filters [7] to do content/subject-based matching before forwarding the publications to relevant subscriber desks. Each subscriber desk gets its incoming publications from various publishers and sends them in some interleaved order to the sub.

Wait-Free Publisher/Subscriber Control Via Broker Central

A. Request for subscriber joining:
1. Broker central creates a broker desk and its current links (see figure), followed by sending the subscriber its link from the desk.
2. Broker central sends a command to current publisher desks about the matching filters of the new subscriber and the links to it.

B. Request for temporary subscriber departure (for durable subscriptions; no acknowledgement, immediate exit):
Broker central asks the subscriber's desk to store (and not forward) all publications either in an expandable link itself or in a separate space.

C. Request for subscriber rejoining a durable subscription:
Broker central provides the subscriber the existing link from its existing broker desk and asks the desk to add any separately stored messages to the link in usual interleave order.

D. Request for subscriber leaving permanently (no ack):
1. Each current publisher desk is asked to discontinue traffic to the subscriber desk after sending it a link termination message.
2. The subscriber desk is asked to delete all its links (a pub link can be terminated only after its termination message) followed by self termination.

E. Request for a publisher joining:
1. Broker central creates a publisher desk, associated links (see figure above), and current subscribers' matching filters.
2. Each subscriber desk is informed of its link from new desk, followed by the publisher being informed of its link to its desk.

F. Request for publisher leaving permanently (no ack):
1. The publisher desk is asked to publish all pending messages followed by subscriber link termination messages,deletion of incoming links and self termination. Each subscriber desk responds to a link termination message by deleting the corresponding link.

Broker central exists for centralized control. All links with the broker central are thus shown in thin lines compared to thick lines for links carrying publication

traffic in figure 2. Broker central retains a unidirectional link to each broker desk, for commanding the desk. Requests from publishers and subscriber to broker central are processed as given in the protocol above. Centralization of updates in one process ensures that references to "current" state actually reflect one global state. Since all links are FIFO, the (total) order in which broker central processes requests is the same as the order in which its commands are received by the broker desks. Weak fairness is assumed in the system so that each link receives attention periodically from its producer and consumer and is not blocked out indefinitely in favor of other links. For this a simple round-robin scheduling of link activities by processes is adequate.

5 Conclusion

We have built a wait-free publish/subscribe system using only N-valued, multiple-reader, single-writer atomic registers as the inter-process communication primitive. Using the same primitive, we have: (a) solved the producer-consumer problem in a wait-free manner using O(1) register accesses by both producer and consumer, and (b) built a large-capacity, adaptive, wait-free, FIFO, unidirectional communication link that grows and shrinks upon need.

References

1. Lamport, L.: Specifying concurrent program modules. ACM TOPLAS Vol. 5, No. 2, (1983) 190–222
2. Attiya, H.: Efficient and Robust Sharing of Memory in Message Passing Systems. J. Algorithms, Vol. 34, No. 1, (2000) 109–127
3. Herlihy, H.: Wait-Free Synchronization. ACM TOPLAS, Vol. 11, No. 5, (1991) 124–149
4. Herlihy, H.: A Methodology for Implementing Highly Concurrent Data Objects. ACM TOPLAS, Vol. 15, No. 5, (1993) 745–770
5. Attiya, H., Lynch, N., Shavit, N.: Are Wait-Free Algorithms Fast? J. ACM, Vol. 41, No. 4, (1994) 725–763
6. Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R. E., Sturman, D. C.: Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. Proc. ICDCS'99. (1999) 262–272
7. Fabret, F., Jacobsen, H. A., Llirbat, F., Pereira, J., Ross, K. A., Shasha, D.: Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. SIGMOD Rec., Vol. 30, No. 2, (2001) 115–126
8. Haldar, S., Vidyasankar, K.: Constructing 1-Writer Multireader Multivalued Atomic Variables from Regular Variables. J. ACM, Vol. 42, No. 1, (1995) 186–203
9. Varma, P.: Compile-time analyses and run-time support for a higher order, distributed data-structures based, parallel language. University Microfilms International, Ann Arbor, Michigan, (1995)

Accelerated Diffusion Algorithms on General Dynamic Networks

Jacques Bahi, Raphaël Couturier, and Flavien Vernier

Laboratoire d'Informatique de l'université de Franche-Comté (LIFC)
IUT de Belfort-Montbéliard, BP 527, 90016 Belfort Cedex, France.
{bahi, couturie, vernier}@iut-bm.univ-fcomte.fr

Abstract. In this paper, an adaptation of the first order diffusive scheme to dynamic networks and its accelerated version with an optimal relaxation parameter are given. A network with dynamic links assumes that edges of the network topology may be unpredictably broken down. Contrary to the static network framework, the connection graphs of the dynamic network may be always not connected. Various simulations on static and dynamic networks confirm our theoretical results.

1 Introduction

One of the most fundamental problems in distributed processing consists in balancing the work between all processors. The purpose of load (work) balancing is to achieve better performances of distributed computations, by improving load allocation. The load balancing problem was studied by several authors from different points of view [1,2,3]. In distributed systems, the schedules of load balancing problems are iterative in nature and their behavior can be characterized by iterative methods derived from linear system theory. Local iterative load balancing algorithms were first proposed by G. Cybenko in [2].

The diffusion load balancing algorithm has been developed for use on networks with fixed topologies. But nowadays, with grid computing possibilities, some problems of communication failures or communication time out (i.e. low bandwidth communication) and problems of heterogeneous networks appear. Load balancing on networks with dynamically changing topologies (i.e., dynamic networks) were first introduced in [3] and [4]. A dynamic network can be viewed here as a network with a fixed topology in which some edges fail unpredictably during the execution time.
In [4] an adaptation of the classical dimension exchange algorithm to dynamic networks was given. In this paper we first introduce an adaptation to general dynamic networks of first order diffusion algorithms and then we present their accelerated versions. Accelerated versions consist in choosing an optimal parameter in order to speed the convergence to the uniform distribution load. We call these accelerated algorithms the relaxed diffusion algorithms.

This paper is organized as follows. Section 2 is devoted to the first order diffusive scheme. Section 3 presents the generalization of first order diffusion

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 77–82, 2004.
© Springer-Verlag Berlin Heidelberg 2004

algorithms to dynamic networks. In this section the new model of such algorithms is introduced and the convergence theorem is given. In section 4, the relaxed diffusion algorithm is given, its convergence is proved and a discussion about the practical choice of the relaxation parameter is done. Section 5 is dedicated to the checking of the theoretical results by numerical simulations.

2 Diffusion on Static Networks

The diffusion algorithm on dynamic networks is derived from first order diffusive algorithm for static networks [2] Classically a static network topology is represented by a simple undirected connected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, $E \subseteq V \times V$. Each computing processor is a vertex of the graph and each communication link between two processors i, j is the edge $\{i, j\} \in E$ between the two vertices i and j $(i, j \in V)$. By definition, each vertex is labeled from 1 to n where n is the number of processors, so $|V| = n$. Let m be the number of communication links $(|E| = m)$.
In [2] Cybenko introduced the diffusion load balancing algorithm (first order). This algorithm assumes that a process i balances its load simultaneously with all its neighbors. To balance the load, a ratio α_{ij} of load difference between the process i and its neighbor j is swapped between i and j. For a process i, the load balancing step with all its neighbors is given by equation (1) where $w_i^{(t)}$ is the work load done by process i at time t.

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j \alpha_{ij} \left(w_j^{(t)} - w_i^{(t)} \right) \quad . \tag{1}$$

3 Diffusion on Dynamic Networks

This section introduces the adaptation of first order diffusion algorithms on general dynamic networks topologies. Assume that we have a non-bipartite and connected graph for the initial global network. In dynamic networks, a *living edge* at a given time, is an edge that can transmit one message in each direction [5]. We assume that at each time step, each node in a dynamic network knows which of its edges are alive. The principle of the first order diffusion algorithm on dynamic networks is that a node balances an amount of its load only on its living neighbors. To get our main result, we need the following definition.

Definition 1. *[4] At each time step, the communication graph is the graph which shows only the edges that are used for load balancing communications at this time. A superposed communication graph $G_{t,t+L}$ between the times t and $t + L$ is a graph that shows all edges used for load balancing communications between the times t and $t + L$.*

The algorithmic model needs a graph representation for dynamic networks. Denote by $G = (V, E)$ the classical undirected connected graph corresponding to the initial global network and define the set $E_B^{(t)}$ as the set of broken edges at

time t. Then $G^{(t)} = (V, E, E_B^{(t)})$ will denote the graph model for the dynamic network, $E_B^{(t)}$ is a subset of E. Remark that if $E_B^{(t)}$ is empty at any time t, then $G^{(t)} = (V, E, E_B^{(t)})$ is the static network $G = (V, E)$.
The first order diffusion algorithm for the dynamic network $G^{(t)}$ is defined by

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j \alpha_{ij}^t \left(w_j^{(t)} - w_i^{(t)}\right) \text{ where } \alpha_{ij}^t = \begin{cases} \alpha_{ij} & \forall (i,j) \notin E_B^{(t)} \\ 0 & \forall (i,j) \in E_B^{(t)} \end{cases}, \quad (2)$$

The equation (2) is linear and it expresses the vector equation (3) that updates the loads for all nodes at time t.

$$w^{(t+1)} = M^{(t)} w^{(t)} \text{ where } M_{ij}^{(t)} = \begin{cases} \alpha_{ij}^t & \text{if } (i,j) \in E \\ 1 - \sum_{j \neq i} \alpha_{ij}^t & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}, \quad (3)$$

Theorem 1. *Algorithm (2) converges toward the uniform load distribution if and only if to any time t, it corresponds a time $t + L$ such that the superposed communication graph $G_{t,t+L}$ is a connected graph. (The proof of this theorem is given in [4].)*

Remark 1. It should be noted that theorem 1 does not claim that all edges have to be alive during the load balancing process neither that the graphs have to be connected. Situations in which no graph is connected are possible, theorem 1 only requires that, at any time t, we are ensured to get a connected graph at a later time $t + L$ by the superposition of the communication graphs between t and $t + L$. Note also that L is not necessarily a constant integer.

4 The Relaxed Diffusion Algorithm

This section introduces the relaxed diffusion algorithm in order to speed up the convergence of the first order diffusive scheme (1). The parameter β is the relaxation parameter that defines the speedup of the convergence. It's known [6] that if M is a stochastic matrix, then the optimal parameter is

$$\beta = \frac{2}{2 - (s + l)}, \quad (4)$$

where s and l are respectively the smallest and the second largest eigenvalue of M. This definition of β does not imply that each $w_i^{(t)}$ of $w^{(t)}$ stays positive. Parameter β must be defined such that $w^{(t+1)} \geq 0$ for all t. Let's define $\beta^{(t)}$ to be the relaxation parameter at time t for the network $G^{(t)}$. The relaxed diffusion algorithm is defined by

$$w^{(t+1)} = \left(1 - \beta^{(t)}\right) w^{(t)} + \beta^{(t)} M^{(t)} w^{(t)}. \quad (5)$$

First, remark that theorem 1 is true for the relaxed algorithm (5). Indeed it is sufficient to replace the matrices $M^{(t)}$ by the matrices $(1 - \beta^{(t)})I + \beta^{(t)} M^{(t)}$. Then we have the corollary

Corollary 1. *Under hypotheses of theorem 1 the relaxed diffusion scheme converges to the uniform distribution workload vector.*

Let $s^{(t)}$ and $l^{(t)}$ denote respectively the smallest and the second largest eigenvalue of $M^{(t)}$. It can easily be verified that a sufficient condition to have $w_i^{(t+1)} \geq 0$ for all i, is

$$\beta^{(t)} \leq R^{(t)} \quad \text{where} \quad R^{(t)} = \min_i \frac{w_i^{(t)}}{\left(1-M_{ii}^{(t)}\right)\left(w_i^{(t)}-w_{min}^{(t)}\right)} \quad . \tag{6}$$

Consequently, in order to satisfy both equations (4) and (6), we define the relaxation parameter $\beta^{(t)}$ to be equal to

$$\beta^{(t)} = \min\left(R^{(t)}, \frac{2}{2-\left(s^{(t)}+l^{(t)}\right)}\right) \quad . \tag{7}$$

Note that as the scheme is convergent, $R^{(t)}$ is an isotone function. Remark that if $\beta = 1$ then the relaxed diffusion algorithm is equivalent to the diffusion algorithm and that if -1 is always the smallest eigenvalue of M, the diffusion algorithm does not converge but the relaxed diffusion algorithm converges.

For the relaxed diffusion algorithm, parameter $\beta^{(t)}$ must be determined at each time step. The calculation of $\beta^{(t)}$ needs a global information on the network, this is not convenient on distributed systems. Thus, in the practical case, β is defined as the optimal parameter corresponding to the initial network and used at each time step ($\beta^{(t)} = \beta$)

$$\beta = \min\left(R, \frac{2}{2-(s+l)}\right) \quad \text{with} \quad R = \min_i \frac{w_i^{(0)}}{(1-M_{ii})\left(w_i^{(0)}-w_{min}^{(0)}\right)} \quad . \tag{8}$$

The next section is dedicated to the experimental study. This study confirms that theorem 1, and corollary 1 are true and that the choice of the value of β is relevant.

5 Simulations

Simulations are presented to validate the theoretical results of this paper and to compare the behavior of first order and relaxed diffusion algorithms on static and dynamic networks.

In a first experimentation, we are interested in comparing the behavior of the diffusion and the relaxed diffusion algorithms in static and dynamic networks. For this simulation, a mesh (8x4) is used. To have readable graphics, the global load of the system is distributed on the 8 nodes of the first column. For the relaxed diffusion algorithm, β is defined according to equation (8), in this case $\beta = 1.25$. At each iteration for simulations on dynamic networks, 5 edges are randomly broken. Graphics on Fig.1 show the load repartition at each iteration and illustrate the convergence toward the uniform load distribution with and

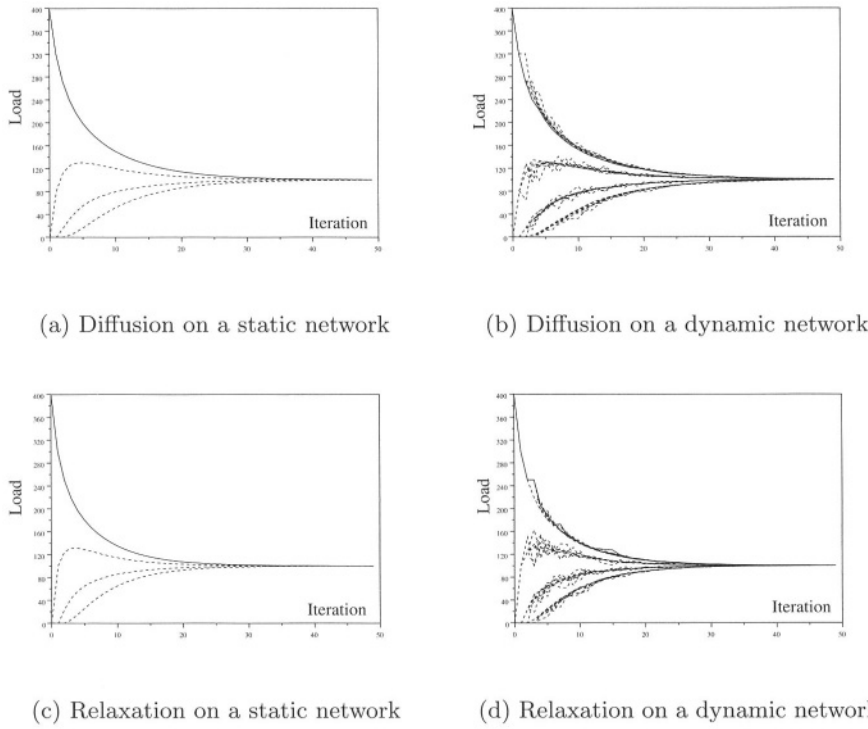


Fig. 1. Load evolution

without edge failures. On graphics 1(a) and 1(c) load of nodes of the same column behave uniformly due to symmetries of the topology and the initial load distribution. In the opposite, on graphics 1(b) and 1(d), some load variations between nodes of the same column appeared consequently to link failures.

Table 1. Results of simulations on various networks

		line (32)		ring (32)		mesh (8x4)		tore (8x4)		hyperc. (d5)	
		st.	dyn.	st.	dyn.	st.	dyn.	st.	dyn.	st.	dyn.
first order	ite.	1649	1935	412	484	172	191	44	48	16	19
relaxation	β	1.5		1.486		1.25		1.165		1	
	ite.	1098	1292	298	325	137	154	40	42	16	18
ratio		1.5	1.5	1.4	1.5	1.3	1.2	1.1	1.1	1	1

In a second experimentation, to emphasize the convergence of algorithms on dynamic networks and the speedup of the relaxed diffusion, others simulations are performed on various networks topologies. For these simulations, the load of

the system (3200) is initially located on node 0, and 5 edges are broken at each time step. Results of the experimentations are given in Table 1.

This table shows the number of iterations (ite) required to reach the convergence for each topology, in static (st) and dynamic (dyn) frameworks. For the relaxed diffusion, the first line gives β computed according to equation (8) before the load balancing (i.e. β without communication failure). Last line of the table gives the speedup for the relaxed diffusion, and highlight that in most cases, the relaxed diffusion algorithm achieves better performances. If the graph is strongly connected then β is close to 1. In this case, the speedup of the relaxed diffusion is not significant. The optimal β is 1 for a hypercube, so the relaxed diffusion is equivalent to diffusion.
These simulations illustrate that the diffusion on dynamic networks and the relaxed diffusion on static and dynamic networks converge toward the uniform load distribution. They also show that relaxed diffusion algorithms accelerate the convergence of diffusion ones and that the practical choice of β is relevant.

6 Conclusion and Future Work

This paper introduces a new adaptation of the first order diffusive algorithm and its relaxed scheme, the framework is that of dynamic networks. Relaxed diffusion schemes speed up the convergence of diffusion ones on static and dynamic networks. We determine the necessary and sufficient conditions for their convergence and these condition are close to real situation. This result is useful when the topology may change due to failures in the communication links, and is well-suited for large problems that need to share computations among distant processors, as it is the case in grid computing. It will be interesting in future work to study fine practical evaluations of the relaxation parameter β, and to apply the new algorithms to a real scientific application in the framework of grid computing.

References

1. Hosseini, S.H., Litow, B., Malkawi, M., McPherson, J., Vairavan, K.: Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm. Jou. of Para. and Dist. Comp. **10** (1990) 160–166
2. Cybenko, G.: Dynamic Load Balancing for Distributed Memory Multiprocessors. Jou. of Para. and Dist. Comp. **7** (1989) 279–301
3. Bahi, J.M., Gaber, J.: Load Balancing on Networks with Dynamically Changing Topology. Europar Conf. LNCS (2001) 175–182
4. Bahi, J.M., Couturier, R., Vernier, F.: Broken Edges and Dimension Exchange Algorithm on Hypercube Topology. Euromicro Conf. PDP (2003) 140–145
5. Leighton, F.T., Maggs, B.M., Sitaraman, R.K.: On the Fault Tolerance of Some Popular Bounded-Degree Networks. SIAM Jou. on Comp. **27** (1998) 1303–1333
6. Berman, A., Plemmons, R.J.: Nonnegative Matrices in the Mathematical Sciences. third edition, 1979, SIAM, Philadelphia (1994)

Suitability of Load Scheduling Algorithms to Workload Characteristics

Eunmi Choi[1] and Dugki Min[2]

[1] School of Computer Science and Electronic Engineering, Handong Global University, Heunghae-eub, Puk-ku, Pohang, Kyungbuk, 791-708, Korea
emchoi@handong.edu

[2] School of Computer Science and Engineering, Konkuk University, Hwayang-dong, Kwangjin-gu, Seoul, 133-701, Korea
dkmin@konkuk.ac.kr

Abstract. In this paper, we investigate the suitability of load scheduling algorithms to workload characteristics. With a set of workloads classified by the amount of allocated resources and their allocation times, we perform experiments with various load scheduling algorithms. As adaptive scheduling algorithms, ARR and ALC are introduced on top of the ALBM (adaptive load balancing and management) cluster system. We present the performance results with a number of classified workloads and a realistic workload, explaining the performance relationship between workloads and scheduling algorithms.

1 Introduction

A major research topic in designing an efficient web cluster systems is to develop or choose a scheduling algorithm for server load balancing[1]. Various scheduling algorithms can be found in the Linux Virtual Server(LVS)[2]. In Linux OS kernel level, LVS delivers NAT (Network Address Translation), IP Tunneling, Direct Routing schemes with several scheduling algorithms, such as Round-Robin (RR), Weighted RR (WRR), Least Connection (LC), and Weighted LC (WLC).

In this paper, we investigate the suitability of load scheduling algorithms to workload characteristics. We classify workloads into a number of classes according to the allocated amount of resources and their allocation times. Workload suitability tests are applied to RR, LC algorithms, and our adaptive version of RR and LC. The adaptive algorithms, called ARR and ALC, take into account the server state information that can be measured in the servers and reflect the information on adjusting their assignment decisions. We did not consider WRR and WLC in this paper, since we use homogeneous server nodes in experiments. We compare the performances of those algorithms on a number of artificial workloads and a realistic workload.

This paper is organized as follows. Section 2 presents the ALBM cluster system architecture considering the feature of adaptive load balancing mechanism. Section 3 introduces the issue of suitability of load scheduling algorithms to workload types. Section 4 shows performance results of load balancing algorithms on a number of workload groups. We summarize in Section 5.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 83–88, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 The ALBM Cluster System

Our web cluster system, called the ALBM (Adaptive Load Balancing and Management) cluster, has a hybrid cluster architecture employing both traffic managing switches and middleware services. Compared to switch-based clusters and to middleware-based clusters [3], the ALBM hybrid cluster is scalable as the switch-based cluster is, also it is reliable and manageable as the middleware-based cluster is.

2.1 ALBM Cluster Architecture

The ALBM system has a hybrid cluster architecture containing switch-level component and middleware-level components as shown in Figure 1. Although it is composed of tens of application servers, called nodes, it is published with one site name and one virtual IP address that are assigned to one or a number of front-end components, called Traffic Manager (TM)s. The TMs interface the rest of cluster nodes, making the distributed nature of the architecture transparent to Internet clients. All inbound packets to the system are received and forwarded to application servers through the TMs. It provides network-level traffic distribution services by keeping on balancing the servers loads. On each of application servers, a middleware service, called Node Agent (NA), is deployed. The NA is in charge of management and operation of the ALBM cluster, such as cluster formation, membership management, and adaptive load balancing. The management station, called M-Station, is a management center of the entire ALBM cluster system. All administrators commands typed through management console are received and delivered to the M-Station. Communicating with NAs, the M-Station performs cluster management operations, such as cluster creation/removal and node join/leave. The M-Station also collects current states and configurations of the entire system, and provides them to other authenticated components in the system. Using the server state information provided by NAs, it performs proactive management actions according to the predefined policies. Besides, the M-Station checks the state of the system in service-level, and carries on some actions when the monitored service-level quality value is significantly far behind with the service-level QoS objectives.

2.2 Adaptive Load Balancing Mechanism

The adaptive scheduling algorithms in the ALBM cluster adjust their schedules, taking into accounts of dynamic state information of servers and applications collected from servers. The ALBM algorithm is as follows. By collecting appropriate information of server states, the NAs customize and store the data depending on the application architecture, machine types, and expectation of a system manager. Each NA decides if the current state is overloaded or underloaded by using upper or lower thresholds of resource utilization determined by system configuration and load balancing policies of cluster management. Each cluster has a Master NA, and only the Master NA communicates with TM in

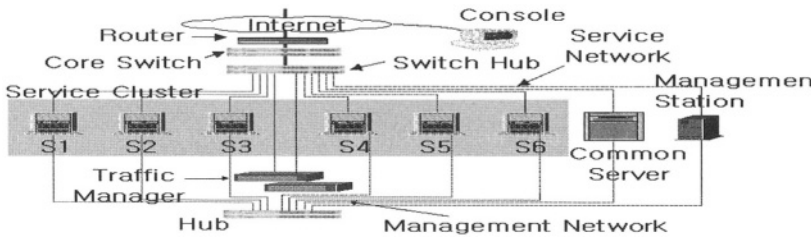


Fig. 1. ALBM Cluster Architecture

order to control the incoming TMs traffic. After collecting state data of all NAs in a cluster, the Master NA reports the state changes to the TM. Thus, real-time performance data are transferred to the M-Station, and the state data of servers are reported to the TM. By using the state information reported by Master NAs, the TM adjusts traffics of incoming requests properly to balanced server allocation. The TM does not allocate requests to overloaded servers, until the overloaded server state is back to a normal state. The scheduling algorithms are applied to the TM through the control of M-Station.

3 On Suitability of Load Scheduling Algorithms

Understanding the suitability of load scheduling algorithms to workload types is important in tuning a TM for improving performance of a site. It helps to select an appropriate scheduling algorithm to a given workload and to understand the effect of exchanging scheduling algorithms. To gain an insight into suitability, we perform a couple of experiments with various workload types. Among many kinds of workload characteristics, we use the features related to server resource utilization. The request size of an allocated resource and the amount of resource holding time are chosen. The request sizes of allocated resources and their holding times are not a priori information of a given workload in general. However, most of Web sites are classified into a service category that determines its own special workload type. For example, an e-commerce request requires small amount of resources in a short time. On the contrary, in case of audio/video stream services like Internet movie theaters, large sizes of streams take the major portion. Meanwhile, an on-line chatting session requires a small amount of resources, but for a long duration of high variance. In literatures, these kinds of Web service classification have been used for various researches [4].

Figure 2 shows the workload types considered in this paper. The x-axis shows the size of resource allocation and the y-axis shows the duration. In Figure 2-(a), each dot point indicates a workload type of a fixed size and a fixed duration (FSFD). The square containing all dots represents all possible types of workload with various sizes and various durations (VSVD). Figure 2-(b) represents another workload types that are composed of two types of workloads. The SSLD_LSSD

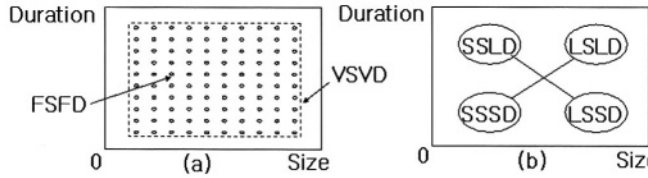


Fig. 2. Classification of Workload Types

pair contains request groups of SSLD (Small Size and Long Duration) and LSSD (Large Size and Small Duration) types. As the opposite case, the SSSD_LSLD pair has request groups of SSSD (Small Size and Small Duration) and LSLD (Large Size and Long Duration).

4 Experimental Results on Suitability

For suitability experiments, the workloads are generated by the Web Bench Tool [5] using 32 client machines that are interconnected to the cluster of 4 server nodes in the same network segment. A server node has PIII-900MHz dual CPU and 512 MB memory. Each client has PIV 1.4GHz CPU and 256MB memory. The network bandwidth is 100MB. As for workload, we use ASP requests rather than html requests since a request should hold a certain size of memory in certain duration. We use the number of ASP requests serviced per second as the performance metric.

4.1 Fixed Memory Size and Fixed Duration

The first observation is an experiment with the memory requests of fixed size and fixed duration (FSFD). Each request requires the same 10MB of memory in a server node, and holds the memory in 0 sec. Figure 3 shows the results. The x-axis shows the number of client machines generating requests and the y-axis shows the average throughputs (i.e. the number of requests served per second) of 4 server nodes. The more clients participate, the more stress is given. As we can expect, all scheduling algorithms show as good performance as RRs until the number of clients reaches 10, the peak point. This is because each request requires the fixed amount of memory in a fixed duration and thus the RR scheduling well distributes the workload in balance. In case of LC, the number of current connections is in proportion to the amount of allocated resources. Thus, LC scheduling balances the amount of allocated memory for each server node. The behaviors of ARR and ALC are similar to those of RR and LC, respectively, resulting in the similar performances up to the peak point.

4.2 Various Memory Sizes and Various Durations

Figure 4 is the experiment results with memory requests of various sizes and various durations. The required memory sizes are 5MB, 10MB, and 15MB in the

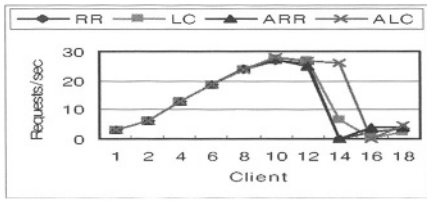


Fig. 3. Fixed Size & Fixed Duration

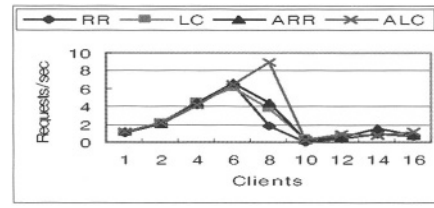


Fig. 4. Various Size & Various Durations

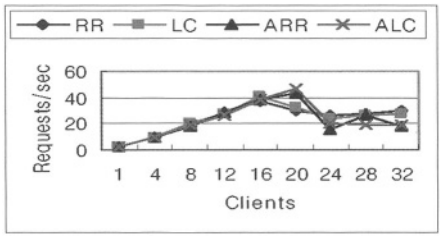


Fig. 5. SSSD versus LSLD

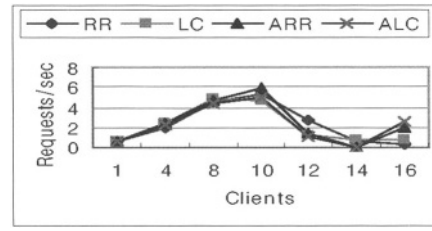


Fig. 6. E-Commerce & Memory Request Traffics

same proportion. The required durations are 0 second, 10 seconds, and 20 seconds in the same proportion. That is, one ninth of total generated requests require 5MB memory size and 0 second holding time, etc. As shown in Figure 4, LC and ARR show as poor performances as RR, compared to ALC. In this workload situation, since every request takes a portion of memory in a various size and holding time, the number of established connections is not a good indicator to represent the memory resource state. Thus, the performance of LC is not good. The best performance is achieved with the ALC algorithm. The adaptive operation of ALC could obtain balancing effect under the unbalanced situation at a peak load, increasing the throughput.

4.3 SSSD and LSLD

Most of home pages contain small sizes of html documentation containing only text and small images. These kinds of light documentation requests take small sizes of memory in a short time. Meanwhile, there are some heavy home pages containing audio and video streams, each of which takes large memory for a long duration. We mix these two extreme cases into a workload type. Half of requests require 5MB memory in 0 sec holding time, and the other half requires 15MB in 20 secs. Figure 5 shows the experimental results. At the peak stress (at 10 clients) RR-type algorithms achieve better performances than LC-type ones. This is because RR-type algorithms mix the light requests with heavy ones evenly, balancing the loads, and LC-type could not be sensitive on server state only by using the number of current request connections. Adaptive algorithms obtain better performances than non-adaptive ones.

4.4 Realistic Workload

We make an experiment with a more realistic workload. The Web Bench Tool provides a general e-commerce workload, which contains text pages, image files, and CGI requests. We make a more realistic workload that is heavy-tailed. In literature, through their experimental studies, many researchers have concluded that general Internet traffics follow heavy tail distributions[6]. In order to make heavy-tailed e-commerce traffic, we mix Web Bench e-commerce traffic and the traffic of random memory size and duration at the rate of 80% and 20%, respectively. The size and duration of a memory request are random numbers of uniform-distribution between 5MB to 15MB and between 0 to 20 seconds, respectively. As shown in Figure 6, adaptive ARR and ALC algorithms achieve better performances than non-adaptive ones. By sensitively applying server states to load scheduling, the ALC achieves the best performance.

5 Conclusion

To gain an insight into the suitability, we performed a couple of experiments with various workload types and various load scheduling algorithms on the ALBM cluster system. The workload suitability tests are applied to RR algorithm, LC algorithm, and our adaptive version of RR and LC. The adaptive algorithms, called ARR and ALC, take into account the server state information that can be measured in servers and reflect the information on adjusting their assignment decisions. Compared to non-adaptive RR and LC, the adaptive scheduling ARR, and ALC scheduling algorithms achieved better performances in various memory size and various duration, and realistic traffics with E-commerce and memory requests.

References

1. Li Xiao, Songqing Chen, Xiaodong Zhang: Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands. IEEE Transactions on Parallel and Distributed Systems, Vol.13, No.3, March (2002) 223-240
2. Wensong Zhang: Linux Virtual Server for Scalable Network Services. Linux Symposium 2000, July (2000)
3. Jeffray S. Chase: Server switching: yesterday and tomorrow. Internet Applications (2001) 114-123
4. Krishna Kant, Youjip Won: Server Capacity Planning for Web Traffic Workload. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 5, September/October (1999) 731-747
5. Web Bench Tool, http://www.etestinglabs.com
6. Martin F. Arlitt, Carey L. Williamson: Internet Web Servers: Workload Characterization and Performance Implications. IEEE/ACM Transactions on Networking, Vol. 5, No. 5, October (1997) 631-645

Minimizing Time-Dependent Total Completion Time on Parallel Identical Machines*

Stanisław Gawiejnowicz, Wiesław Kurc, and Lidia Pankowska

Adam Mickiewicz University
Faculty of Mathematics and Computer Science
Umultowska 87, 61-614 Poznań, POLAND
{stgawiej, wkurc, lpankow}@amu.edu.pl

Abstract. In the paper a problem of minimizing the total completion time for deteriorating jobs and parallel identical machines is considered. The processing time of each job is a linear function of the starting time of the job. The properties of an optimal schedule are proved and a greedy heuristic for the problem is proposed. Preliminary results of experimental evaluation of the algorithm are given.

1 Introduction

We consider a multiple machine scheduling problem with deteriorating jobs. The deterioration of the processing time of a job is modelled by an increasing function that depends on the starting time of the job. Problems of this kind appear, e.g., in fire fighting, in scheduling of learning activities and scheduling calls for medical emergency response teams. We refer the reader to the review by Alidaee and Womer [2] for more details.

Most of the research done in the area concerns a single machine case [2]. To the best of our knowledge, only a few results concerning parallel identical machines are known. Chen [3] and Kononov [8] proved that the problem of minimizing the total completion time for proportional processing times is ordinarily NP-hard. Kononov [8] and Mosheiov [11] proved that minimizing the maximum completion time for proportional processing times and for two parallel identical machines is ordinarily NP-hard. Similarly, there are known only a few results concerning dedicated machines, see Kononov [8], Kononov and Gawiejnowicz [9], Mosheiov [11], [12]. The reader is referred to the survey by Cheng et al. [4] for more details on parallel machine scheduling with deteriorating jobs.

2 Problem Formulation

There are given m parallel identical machines $M_1, M_2, \ldots, M_m$ and a set of N jobs. The jobs are nonpreemptable and independent, there are no ready times and no deadlines. All machines are available continuously, starting from time

* The research has been partially supported by KBN grant no. 4T11C 03925.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 89–96, 2004.
© Springer-Verlag Berlin Heidelberg 2004

$t = 0$. The processing time p_j of the j-th job, $j = 1, \ldots, N$, is a linear function of the starting time t of the job, $p_j = d + \alpha_j t$, where the job deterioration rate $\alpha_j > 0$ for $j = 1, \ldots, N$ and $d = O(1) > 0$. (In Section 4 we will show that w.l.o.g. we can assume that $d \equiv 1$.) The criterion of optimality of a schedule is the total completion time of all jobs, $\sum C_j$.

Let $\mathcal{J} = \{J_1, \ldots, J_N\}$ and $\mathcal{J}_i = \{J_0^i, \ldots, J_{n_i}^i\}$ denote, respectively, a set of all jobs and a set of jobs assigned to machine M_i, $i = 1, \ldots, m$, where $\mathcal{J}_i \cap \mathcal{J}_j = \emptyset$ for $i \neq j$, $\mathcal{J} = \mathcal{J}_1 \cup \ldots \cup \mathcal{J}_m$ and $m + \sum_{i=1}^m n_i = N$. Let $I = \{1, \ldots, N\}$ and $I_i = \{0, \ldots, n_i\}$ denote a set of job indices from sets $\mathcal{J}$ and $\mathcal{J}_i$, $i = 1, \ldots, m$, respectively.

We will distinguish parameters of the problem before and after scheduling by omitting (giving) upper indices. For example, α_j and α_j^i will denote the deterioration rate of the j-th job from set $\mathcal{J}$ and the deterioration rate of the job scheduled as the j-th one on machine M_i, $i = 1, \ldots, m$, respectively. Similarly, for a given schedule, C_j^i will denote the completion time of the job scheduled as the j-th one on machine M_i, $j = 0, \ldots, n_i$ and $i = 1, \ldots, m$.

Notice that since on all machines we begin scheduling at time $t = 0$, the completion times $C_0^i = 0 + p_0^i(0) = 0 + d + \alpha_0^i \cdot 0 = d$ for $i = 1, \ldots, m$. The completion times C_j^i for the remaining jobs are given by the formula

$$C_j^i = (1 + \alpha_j^i) C_{j-1}^i + d = a_j^i C_{j-1}^i + d, \tag{1}$$

where $a_j^i = 1 + \alpha_j^i$ for $j = 1, \ldots, n_i$ and $i = 1, \ldots, m$. Our aim is to find a schedule $\sigma^\star$ such that $\sum C_j(\sigma^\star) \equiv \sum_{i=1}^m \sum_{j=0}^{n_i} C_j^i$ is minimal.

Throughout the paper the above problem will be called in short the *TPTCT (Time-dependent Parallel-machine Total Completion Time)* problem.

3 Optimal Schedule Properties

Let $\mathcal{N}$ denote the set of indices of yet unscheduled jobs. (Obviously, at the beginning of scheduling we have $\mathcal{N} = I$.) Note that since job processing times are described by nondecreasing functions of time, only the schedules without idle times should be considered. Thus we can identify a schedule with a sequence of indices of jobs assigned to particular machines and a job with its index, respectively.

Lemma 1. *Let $\mathcal{N} = I$ and let $k_i = \arg\max\{\alpha_j, j \in \mathcal{N}\}$, $i = 1, \ldots, m$, denote indices of m greatest deterioration rates. Then in any optimal schedule a job J_k from the set $\{J_{k_1}, \ldots, J_{k_m}\}$ is scheduled as the first one on each machine.*

Proof. First, we will show that in any optimal schedule, the first job scheduled on machine M_i, $i = 1, \ldots, m$, is the one with the greatest deterioration rate among the jobs assigned to this machine. Indeed, let job J_k be scheduled as the first one on machine M_q, $q \in \{1, 2, \ldots, m\}$. Then the completion time of this job is $C_0^q = d$. Since in this case C_0^q does not depend on α_k, it is easy to

see that in an optimal schedule as the first job should be such that its index $k = \arg\max\{\alpha_j, j \in \mathcal{N}\}$.

To end the proof it is sufficient to note that m jobs scheduled as the first ones must have m greatest deterioration rates of all rates. Indeed, let in an optimal schedule on some machine M_q there exist a job J_s^q such that $\alpha_s^q < \alpha_0^q$ and let there exist another machine M_r such that $\alpha_0^r < \alpha_s^q$. Then, by switching jobs J_s^q and J_0^r, we obtain a new schedule, whose total completion time is lower than that of the optimal schedule. A contradiction. □

From Lemma 1 it follows that there are $m!$ possible assignments of the first m jobs to available machines. Thus an optimal schedule is not unique. Notice also that by Lemma 1 we should consider as possible optimal schedules only the schedules with $N \geq 2m$ jobs. Indeed, if $N \leq m$, we can schedule the jobs in an arbitrary way on the machines. If $m < N \leq 2m$, then, by Lemma 1, the first m greatest jobs are scheduled on the machines as the first ones, and the remaining jobs can be scheduled on available machines in an arbitrary order, since all machines are available at the same time $t = d$.

Lemma 2. *If $N \geq 2m$, then in any optimal schedule at least two jobs are scheduled on each machine.*

Proof. Assume that there are $N \geq 2m$ jobs and that there exists an optimal schedule $\sigma^\star$ such that only one job is assigned to some machine M_q. Let M_r be a machine with the largest load in schedule $\sigma^\star$, $J_{n_r}^r$ be a job assigned to M_r as the last one and let $s_{n_r}^r > d$ denote the starting time of this job. Then the completion time of the job is $C_{n_r}^r = d + (1 + \alpha_{n_r}^r)s_{n_r}^r$ and the total completion time for the schedule $\sigma^\star$ is $\sum C_j(\sigma^\star) = T + d + (1 + \alpha_{n_r}^r)s_{n_r}^r$, where T denotes the sum of completion times for jobs other than $J_{n_r}^r$.

Construct a new schedule τ by assigning job $J_{n_r}^r$ to machine M_q. Then the completion time of the job is $C_1^q = d + (1 + \alpha_1^q)d$ and the total completion time for the schedule τ is $\sum C_j(\tau) = T + d + (1 + \alpha_1^q)d$. Since $\alpha_{n_r}^r = \alpha_1^q$, we have $\sum C_j(\tau) - \sum C_j(\sigma^\star) = (1+\alpha_{n_r}^r)(d - s_{n_r}^r) < 0$. In conclusion, schedule τ is better than schedule $\sigma^\star$. A contradiction. □

Lemma 3. *The value of the total completion time of any schedule will remain unchanged, if on arbitrary machine M_q, $q = 1, \ldots, m$, the sequence of jobs assigned to the machine will be replaced by the sequence reversely ordered, starting from the second position.*

Proof. It is sufficient to note that for an arbitrary sequence and the reversed sequence, starting from the second position, the value of the criterion function is the same: $\sum_{i=1}^{m} \sum_{j=0}^{n_i} C_j^i = \sum_{i=1}^{m} \sum_{j=0}^{n_i} \sum_{k=0}^{j} \prod_{l=k+1}^{j} (1 + \alpha_l^i) = \sum_{i=1}^{m} \sum_{j=0}^{n_i} \sum_{k=0}^{j} \prod_{l=k+1}^{j} (1 + \alpha_{n-l+1}^i)$. □

By Lemma 1 and Lemma 3, there exist at least $m!2^m$ optimal schedules, since there are $m!$ ways of choosing the first m jobs to machines and the sequence of jobs assigned to each machine is symmetric starting from the second job.

Before we state the next result, recall the notion of a V-shaped sequence. A sequence $\alpha = (\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n)$ is said *to be V-shaped* (α *has a V-shape*), if there exists an index k such that for $0 \leq i \leq k$ the sequence α is nonincreasing and for $k \leq i \leq n$ the sequence is nondecreasing.

Theorem 1. *Any optimal schedule for the TPTCT problem is V-shaped on each machine.*

Proof. Assume that there exists an optimal schedule such that the sequence of jobs assigned to a machine M_q is not V-shaped. By V-shape property for a single machine (see Mosheiov [10]), there exists a V-shaped sequence for M_q such that its total completion time is not greater than that of the initial sequence. Thus, by a rearrangement of jobs assigned to M_q, we obtain a new schedule which is V-shaped on M_q and it has not worse value of the criterion function than previously. Repeating this procedure for all machines on which job sequences are not V-shaped, we will obtain an optimal schedule which is V-shaped. □

Applying the V-shape property from Theorem 1, we can establish an upper bound on the number of possible optimal schedules for the *TPTCT* problem.

Theorem 2. *For the TPTCT problem there exist at most* $(2m)^n$ *possible optimal schedules.*

Proof. (sketch) By Theorem 1, we need to consider only V-shaped sequences $a^i = (a^i_1, \ldots, a^i_{n_i})$, $i = 1, \ldots, m$. We can estimate the number of such sequences by the following formula (see Aigner [1] for details) for $\omega_1 = \cdots = \omega_m = 2$:

$$(\omega_1 + \cdots + \omega_m)^n = \sum_{n_1 + \cdots + n_m = n} \binom{n}{n_1}\binom{n-n_1}{n_2} \cdots \binom{n - \sum_{j=1}^{m-1} n_j}{n_m} \omega_1^{n_1} \cdots \omega_m^{n_m}. \qquad \square$$

4 Matrix Approach

In this section we will show that any schedule for the *TPTCT* problem can be transformed into a schedule for a single machine problem of a special type. We will apply the approach introduced by Gawiejnowicz et al. [5].

Let $N = n + m$. Each job $J_j \in \mathcal{J}$ w.l.o.g. can be identified with some element $a_j \in Q = \{a_1, \ldots, a_N\}$, where $a_j = 1 + \alpha_j$ for $j \in I$. A schedule σ^i for machine M_i is a sequence $\widetilde{a}^i \equiv (a^i_0, \ldots, a^i_{n_i})$ with $n_i + 1$ elements, where $a^i_j \in Q$. The complete schedule σ is given by the sequence $\widetilde{a} = (\widetilde{a}^1, \ldots, \widetilde{a}^m)$.

Let $\mathcal{S}(\mathcal{J})$ denote the set of all possible schedules such that on each machine is assigned at least one job, $\sigma \in \mathcal{S}(\mathcal{J})$ be a schedule given by $\widetilde{a}$ and $a = (a^1, \ldots, a^m)$, where $a^i = (a^i_1, \ldots, a^i_{n_i})$. Consider a system of linear equations $A(a)C(a) = D$ which in a block form is as follows:

$$\begin{bmatrix} A_1 & O & \cdots & O \\ O & A_2 & \cdots & O \\ \vdots & \vdots & & \vdots \\ O & O & \cdots & A_m \end{bmatrix} \begin{bmatrix} C^1 \\ C^2 \\ \vdots \\ C^m \end{bmatrix} = \begin{bmatrix} d^1 \\ d^2 \\ \vdots \\ d^m \end{bmatrix}, \tag{2}$$

where $A(a)$ is the above matrix, $C(a) = [C^1(a^1), \ldots, C^m(a^m)]^\top$ is a vector of the completion times of jobs assigned to machine M_i and $C^i(a^i) = (C_0^i, \ldots, C_{n_i}^i)$ for $i = 1, \ldots, m$. Moreover, $D = [d^1, \ldots, d^m]^\top$, $d^i = (d, \ldots, d) \in \mathbb{R}^{n_i+1}$ and

$$A_i \equiv A(a^i) = \begin{bmatrix} 1 & 0 \cdots & 0 & 0 \\ -a_1^i & 1 \cdots & 0 & 0 \\ 0 & -a_2^i \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 \cdots & -a_{n_i}^i & 1 \end{bmatrix}.$$

The system of equations (2) is equivalent to the set of matrix equations $A(a^i)C^i(a^i) = d^i$ for $i = 1, \ldots, m$. Each matrix $A(a^i)$ corresponds to a sequence of jobs assigned to machine M_i, $i = 1, \ldots, m$. Note that a_0^i, for $i = 1, \ldots, m$, never enter matrix $A(a^i)$, since a_0^i are multiplied by 0 in (1). On the other hand, the equalities $C_0^i = d$ are included in (2). Thus we will assume that the first job on each machine is taken from the set of m greatest jobs, i.e. these ones which correspond to the greatest elements in set Q. We will also assume that if no job is assigned to machine M_k, then schedule σ^k consists only of one greatest job and the corresponding equation $A(a^k)C^k(a^k) = d^k$ in (2) has the form of $[1][C_0] = [d]$ or, equivalently, that block $A(a^k)$ in matrix $A(a)$ reduces to 1. Thus, from now, the schedule $\sigma \in \mathcal{S}(\mathcal{J})$ will be identified with a sequence $a = (a^1, \ldots, a^m)$, where $a^i = (a_1^i, \ldots, a_{n_i}^i)$ for $i = 1, \ldots, m$.

Taking into account the above considerations, the *TPTCT* problem can be formulated in an alternative, matrix form as follows: *Find a schedule* $\sigma \in \mathcal{S}(\mathcal{J})$ *such that* $A(a)C(a) = D$, *and that* $||C(a)||_1$ *is minimal.* (We can use the symbol $\|\cdot\|_1$, since criteria $\|\cdot\|_1$ and $\sum C_j$ are equivalent, see Gawiejnowicz et al. [6].)

Since $\det(A(a)) = 1$, matrix $A(a)$ is nonsingular. Its inverse, in block form, is as follows:

$$A^{-1}(a) = \begin{bmatrix} A_1^{-1} & O & \cdots & O \\ O & A_2^{-1} & \cdots & O \\ \vdots & \vdots & & \vdots \\ O & O & \cdots & A_m^{-1} \end{bmatrix}, \quad A^{-1}(a^i) = \begin{bmatrix} 1 & 0 & \cdots 0 & 0 \\ a_1^i & 1 & \cdots 0 & 0 \\ a_1^i a_2^i & a_2^i & \cdots 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ a_1^i \cdots a_{n_i}^i & a_2^i \cdots a_{n_i}^i & \cdots a_{n_i}^i & 1 \end{bmatrix},$$

where A_i^{-1} is of size $(n_i + 1) \times (n_i + 1)$ and O is a zero matrix of suitable size.

From the above we obtain the formula $C_j^i = d \sum_{k=0}^{j} \prod_{l=k+1}^{j} a_l^i$ for elements of vector $C(a) = [C^1(a^1), \ldots, C^m(a^m)]^\top$, where $C^i(a^i) = (C_0^i, \ldots, C_{n_i}^i)$ for $i = 1, \ldots, m$. Since $C(a) = A^{-1}(a)D$, in view of the general form of $A^{-1}(a)$, we obtain $||C(a)||_1 = \sum_{i=1}^{m} ||C^i(a^i)||_1$, where $||C^i(a^i)||_1 = d \sum_{j=0}^{n_i} \sum_{k=0}^{j} \prod_{l=k+1}^{j} a_l^i$ is the sum of all non-zero elements in $A^{-1}(a^i)$. Thus, w.l.o.g., $d \equiv 1$.

Applying the above matrix approach, we can prove all properties of an optimal schedule for the *TPTCT* problem (see Lemmas 1-3 and Theorems 1-2) in an alternative way. Notice also that there holds the following result.

Lemma 4. *For each schedule $\sigma \in \mathcal{S}(\mathcal{J})$ there exists a unique permutation of the sequence $\widehat{a} = (a^1|0|a^2|0|\ldots|0|a^m)$.*

Proof. (sketch) First, note that for each schedule $\sigma \in \mathcal{S}(\mathcal{J})$ there exists a corresponding concatenated sequence $\widehat{a} = (a^1|0|a^2|0|\ldots|0|a^m)$ of elements laying immediately below the main diagonal in matrix $A^{-1}(a)$. Second, there exists a 1-1 correspondence between all possible schedules and all possible permutations of the sequence $\widehat{a}$. □

As a corollary from Lemma 4 we get the number of all assignments of all jobs to the machines, with m greatest jobs already assigned. This number is equal to $(n+m-1)!/(m-1)!$. Indeed, by Lemma 4 the number of all assignments of n jobs to m machines is equal to the number of all permutations of $(n+m-1)$-elements of concatenated sequence $\widehat{a} = (a^1|0|a^2|0|\ldots|0|a^m)$, since $a^i = (a^i_1, \ldots, a^i_{n_i})$ for $i = 1, \ldots, m$ and $\sum_{i=1}^{m} n_i = n$.

5 Greedy Algorithm

We can express $||C(a)||_1$ in the form $||C(a)||_1 = \bar{F}(a) + (n+m)$, where

$$\bar{F}(a) = \sum_{i=1}^{m} F(a^i) \quad \text{and} \quad F(a^i) = \sum_{j=1}^{n_i} \sum_{k=1}^{j} \prod_{l=k}^{j} a^i_l. \tag{3}$$

Notice that $F(a^i)$ is the sum of all non-zero elements in matrix $A^{-1}(a^i)$ outside its main diagonal. Let $\mu_1, \ldots, \mu_m \geq 0$ and let $\eta_1, \ldots, \eta_m \geq 0$ be given numbers. Then, for $i = 1, \ldots, m$, there holds the following formula

$$F(\mu_i|a^i|\eta_i) = F(a^i) + \mu_i C^i_{n_i}(\overline{a}^i) + \eta_i C^i_{n_i}(a^i) + \mu_i \cdot a^i \cdot \eta_i, \tag{4}$$

where $\overline{a}^i = (a^i_{n_i}, a^i_{n_i-1}, \ldots, a^i_1)$ denotes reversely ordered sequence a^i and $\mu_i \cdot a^i \cdot \eta_i$ denotes the product of all elements of a^i and the numbers μ_i, η_i.

By (4), we obtain the following lemma which is a basis for construction of a greedy algorithm for the *TPTCT* problem.

Lemma 5. *For $\mu_1, \ldots, \mu_m \geq 0$ and $\eta_1, \ldots, \eta_m \geq 0$ there holds the formula*

$$\bar{F}(\mu_1|a^1|\eta_1, \ldots, \mu_m|a^m|\eta_m) = \bar{F}(a) + \sum_{i=1}^{m} (\mu_i C^i_{n_i}(\overline{a}^i) + \eta_i C^i_{n_i}(a^i) + \mu_i \cdot a^i \cdot \eta_i). \tag{5}$$

Proof. By (3), we have $\bar{F}(\mu_1|a^1|\eta_1, \ldots, \mu_m|a^m|\eta_m) = \sum_{i=1}^{m} F(\mu_i|a^i|\eta_i)$. Applying (4), the formula (5) follows. □

The following greedy algorithm is based on Lemma 5. Its main idea is to minimize the value of the expression $\mu_i C^i_{n_i}(\overline{a}^i) + \eta_i C^i_{n_i}(a^i) + \mu_i \cdot a^i \cdot \eta_i$. The algorithm can be formulated as follows.

A greedy algorithm for the *TPTCT* problem

Input: set $Q = \{a_1, \ldots, a_{n+m}\}$
Output: suboptimal schedule $\sigma \in \mathcal{S}(\mathcal{J})$ given by $\widetilde{a} = (\widetilde{a}^1, \ldots, \widetilde{a}^m)$

Step 1. (Initialization.)
Sort set Q in the nondecreasing order, $a_{[1]} \leq \ldots \leq a_{[n]} \leq a_{[n+1]} \leq \ldots \leq a_{[n+m]}$ and put these ordered elements on list L;

Step 2. (Preprocessing.)

Step 2a. Assign m jobs corresponding to the elements $a_{[n+1]}, \ldots, a_{[n+m]}$ to machines and remove appropriate elements from list L;

Step 2b. Let $b := (b^1, \ldots, b^m)$ denote $2m$ smallest elements from list L, where $b^i = (\mu_i, \eta_i)$ for $i = 1, \ldots, m$ (if there are less than $2m$ elements, complete the missing elements by zeros). By checking all possibilities, assign the jobs corresponding to these elements to machines in such a way that the value of $\bar{F}(b)$ is minimal;

Step 2c. Remove elements of b from list L;

Step 3. (Main loop)
While list L is not empty do

Step 3a. Choose the next $2m$ elements from list L (if there are less than $2m$ elements, complete the missing elements by zeros);

Step 3b. By checking all possibilities, assign the jobs corresponding to these elements to machines in such a way that $\sum_{i=1}^{m} (\mu_i C^i(\overline{a}^i) + \eta_i C^i(a^i) + \mu_i \cdot a^i \cdot \eta_i)$ is mimimal;

Step 3c. For $i := 1$ to m do $b^i := (\mu_i | b^i | \eta_i)$;

Step 3d. Remove appropriate elements from list L.

The running time of the above algorithm is $O((n+m)\log(n+m)+n(2m-1)!)$. Note that since usually $m = O(1)$ and $N = O(n)$ with respect to n, this running time reduces to $O(N \log N)$.

6 Computational Experiment Results

A computational experiment has been conducted for the case of $m = 2$ machines. Four classes of a_j coefficients were constructed. The class A consisted of random sequences of integers, $a_j \in \{2, \ldots, 20\}$. The class B was composed of random sequences of reals, $a_j \in \{1.1, \ldots, 3.0\}$. The class C consisted of random arithmetic sequences of integers, $a_j = p + (j-1)q$, for randomly chosen $p, q \in \{2, \ldots, 10\}$. Finally, the class D was composed of random arithmetic sequences of reals, $a_j = p + (j-1)q$, for randomly chosen $p, q \in [1.1, 3.0]$.

For each class of the coefficients, 10 instances with $n = 8, 9$ or 10 jobs have been generated. (Note that each instance had, in fact, two jobs more since we assumed that two greatest jobs have been scheduled on each machine first.) Results of the experiment are summarized in Table 1. The symbols $\delta_{\min}, \delta_{\text{avg}}$ and $\delta_{\max}$ denote, respectively, the minimal error, an average error and the maximal error, calculated for each 10 instances.

Table 1. Results of computational experiment (in %)

n	Error	Class A	Class B	Class C	Class D
8	$\delta_{\min}$	0.0000	0.0000	0.0000	0.0000
8	δ_{avg}	0.0058	0.0066	0.0000	0.0006
8	$\delta_{\max}$	0.0525	0.0309	0.0000	0.0006
9	$\delta_{\min}$	90.0317	9.1504	144.5404	72.6650
9	δ_{avg}	113.1480	11.2747	224.1415	111.8880
9	$\delta_{\max}$	130.7722	14.0967	296.5700	157.1201
10	$\delta_{\min}$	0.1544	1.3979	0.1054	0.8836
10	δ_{avg}	1.1449	2.1178	0.2285	1.2995
10	$\delta_{\max}$	2.0290	2.9058	0.4025	1.7750

The above results illustrate the importance of specific construction of formula (5) for behaviour of the proposed algorithm. In particular, the bad result for $n = 9$ is caused by the fact that $n \neq 2km$. On the other hand, the algorithm behaves especially well for arithmetic sequences and $n = 2km$.

References

1. M. Aigner, *Combinatorial theory.* Springer, 1979.
2. B. Alidaee and N.K. Womer, Scheduling with time dependent processing times: Review and extensions. *J. Optl Res. Soc.* **50** (1999), 711–720.
3. Z.-L. Chen, Parallel machine scheduling with time dependent processing times. *Discr. Appl. Math.* **70** (1996), 81–93. Erratum: *Discr. Appl. Math.* **75** (1996), 103.
4. T.C.E. Cheng, Q. Ding and B.M.T. Lin, A concise survey of scheduling with time-dependent processing times. *Euro. J. Optl Res.* **152** (2004), 1-13.
5. S. Gawiejnowicz, W. Kurc and L. Pankowska, A greedy approach for a time-dependent scheduling problem. In: R. Wyrzykowski et al. (eds.), *Parallel Processing and Applied Mathematics,* Lecture Notes in Computer Science **2328.** Springer 2002, 79–86.
6. S. Gawiejnowicz, W. Kurc, L. Pankowska and C. Suwalski, Approximate solution of a time-dependent scheduling problem for l_p-norm-based criteria. In: B. Fleischmann et al. (eds.), *Operations Research Proceedings 2000.* Springer 2001, 372–377.
7. A. Kononov, Combinatorial complexity of scheduling jobs with simple linear deterioration. *Discr. Anal. Oper. Res.* **3** (1996), no. 2, 15–32 (in Russian).
8. A. Kononov, Scheduling problems with linear processing times. In: U. Zimmermann et al. (eds.), *Operations Research Proceedings 1996.* Springer 1997, 208–212.
9. A. Kononov and S. Gawiejnowicz, NP-hard cases in scheduling deteriorating jobs on dedicated machines. *J. Optl Res. Soc.* **52** (2001), 708–717.
10. G. Mosheiov, V-shaped policies to schedule deteriorating jobs. *Oper. Res.* **39** (1991), no. 6, 979–991.
11. G. Mosheiov, Multi-machine scheduling with linear deterioration, *Infor* **36** (1998), 205–214.
12. G. Mosheiov, Complexity analysis of job-scheduling with deteriorating jobs, *Discr. Appl. Math.* **117** (2002), 195–209.

Diffusion Based Scheduling in the Agent-Oriented Computing System

Marek Grochowski, Robert Schaefer, and Piotr Uhruski

Institute of Computer Science, Jagiellonian University, Kraków, Poland
{grochows, schaefer, uhruski}@ii.uj.edu.pl

Abstract. We propose a dynamic, decentralized scheduling strategy which leads to load balancing in a large distributed environment. The main activities are agent partitioning and agent migration performed according to local, diffusion-based rules. The strategy is applied to the agent-based CAE mesh generator, and the solution is tested on a network of PC desktops. We discuss the total computation time, network coverage, and different scheduling properties.

1 Introduction

Centralized task scheduling in large distributed systems is still difficult because of the huge, non-polynominal complexity of mapping algorithms (see [1]), significant communication overheads, as well as a lack of recent information about load distribution available at the managing computer.
Another possibility is to follow the selected process that appears in nature (e.g. the heat conduction) and which converges into a stable final state that may be recognized as the well balanced one (see e.g. [8]). Multi-agent system (MAS) technology seems to be suitable for such a solution [6]. We apply the local rules of this process (e.g. the analogous of the Fourier law [5]) in a small neighborhood of each agent, which can result in agent partitioning and agent migration, so no global information exchange is necessary. The novelty of our approach consists moreover in on-demand task partitioning and distant task remaping obtained by migration of a hibernated task (binary code and task context). We can expect sufficient load balancing due to the appropriate feature of the related process.

2 The Architecture

The MAS solution under consideration that allows the diffusion governed scheduling is a collection of: *a computational environment* (MAS platform) and *a computing application* composed of mobile agents. It is a part of the Smart Solid project (see e.g. [4]) supporting huge CAE computations in the distributed environments.
The computational environment is a triple $(\mathbf{N}, B_H, perf)$, where:

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 97–104, 2004.
© Springer-Verlag Berlin Heidelberg 2004

$\mathbf{N} = \{P_1, \ldots, P_n\}$, where P_i is the Virtual Computation Node (VCN). Each VCN can maintain more than one agent (the number of hardware processors usage is not relevant in our assumptions).

B_H is the connection topology $B_H = \{N_1, \ldots, N_n\}, N_i \subset \mathbf{N}$ is an immediate neighborhood of P_i (including P_i as well).

$perf = \{perf_1, \ldots, perf_n\}, perf_i : R_+ \to R_+$ is a family of functions, which describes relative performance of all VCN with respect to the total memory request M^i_{total} of all allocated agents. If M^i_{total} on P_i is small, $perf_i$ turns back the constant value, which depends only on the CPU architecture. If M^i_{total} is larger, the $perf_i$ decreases due to the intensive swap utilization.

Each *Smart Solid Agent* (SSA) is represented by the pair $A_i = (T_i, S_i)$ where: T_i is the computational task executed by agent, including all data required for computation, and S_i stands for the shell responsible for the agent's logic. The index i stands for an unambiguous agent identifier.
Each task T_i has to denominate the current requirement for computational power $(E_i, M_i)$where: E_i is the task remaining time measured in units common for all application tasks, and M_i is the RAM requirement in bytes. Another important condition we imposed for the task is that it must allow pausing and continuation of it's computation. Pausing is needed for the hibernating task in case of agent migration or partitioning, and continuation is needed to restore the paused job. In particular it can be designed in such a way that it can work from one checkpoint to the next one, and during this checkpoint operation, it saves its present state. Moreover each task T_i can be partitioned into two subtasks $T_i \to \{T_{i_1}, T_{i_2}\}$ such that $E_i > E_{i_j}$, $M_i > M_{i_j}$, $j = 1, 2$. The task partitioning rule depends strongly on the computational problem to be solved (see [9]).
The state of *the computing application* is the triple $(\mathbf{A}_t, G_t, Sch_t)$, $t \in [0, +\infty)$ where:

$\mathbf{A}_t$ is the set of application agents, $\mathbf{A}_t = \{A_{\xi_j}\}_{\xi_j \in I_t}$, I_t is the set of indices of agents active at the time t,

G_t is the tree representing agents partitioning at the time t. All agents constitute the set of nodes $\bigcup_{\xi \in \Theta} A_\xi$, $\Theta = \bigcup_{j=0}^{t} I_j$, while G_t edges show the partitioning history. All information on how to rebuilt G_t is spread among all agents such that each of them knows only its neighbours in the tree.

$\{Sch_t\}_{t \in [0, +\infty)}$ is the family of functions such that $Sch_t : \mathbf{A}_t \to \mathbf{N}$ is the current schedule of application agents among the MAS platform servers. The function is represented by the sets ω_j of agents' indices allocated on each $P_j \in \mathbf{N}$. Each of ω_j is locally stored and managed by P_j.

The shell S_i communicates with both T_i and the local server $P_j = Sch(A_i)$. It supports inter–task communication and queries task requirements for resources as well as implements the necessary logic to perform scheduling. Each server

$P_j \in \mathbf{N}$ periodically asks all local agents (allocated on P_j) for their requirements and computes the local load concentration

$$L_j = \frac{E^j_{total}}{perf_j(M^j_{total})} \text{ where } E^j_{total} = \sum_{i \in \omega_j} E_i \text{ and } M^j_{total} = \sum_{i \in \omega_j} M_i \quad (1)$$

Then P_j communicates with neighboring servers and establishes

$$\mathbf{L}_j = \{(L_\zeta, E^\zeta_{total}, M^\zeta_{total}, perf_\zeta)\} \text{ where } \zeta \text{ is such that } P_\zeta \in N_j \quad (2)$$

as well as the set of node indices Q_j such that

$$k \in Q_j \iff k \neq j,\ P_j \in N_j,\ L_j - L_k > 0 \quad (3)$$

The current values of both $\mathbf{L}_j$ and Q_j are available to the local agents.

3 Diffusion Governed Dynamics

We introduce the *binding energy* parameter $\mathbf{E}_{i,j}$ of the agent A_i allocated on VCN P_j characterized by the following conditions:

$$\mathbf{E}_{i,j} \text{ is a descending function of } E_i \text{ and not an ascending function of } L_j \quad (4)$$

We assume that the agent A_i may dynamically evaluate its binding energy for other nodes from the neighborhood N_j using the information contained in $\mathbf{L}_j$. The current value of the binding energy gradient is a vector defined by:

$$\nabla^t_{i,j} = ((j,l), \mathbf{E}_{i,l} - \mathbf{E}_{i,j}) \text{ where } P_j = Sch(A_i) \text{ and } l \in Q_j \text{ is such that } \mathbf{E}_{i,l} - \mathbf{E}_{i,j} = \max_{\zeta \in Q_j}\{\mathbf{E}_{i,\zeta} - \mathbf{E}_{i,j}\} \quad (5)$$

An agent A_i allocated on P_j migrates to P_l indicated by $\nabla^t_{i,j}$ if the binding energy $\mathbf{E}_{i,l}$ on the destination VCN exceeds the current $\mathbf{E}_{i,j}$ more than ϵ. The threshold ϵ stands for the migration parameter.
In general Smart Solid Agent $A_i = (T_i, S_i)$ currently allocated on $P_j \in \mathbf{N}$ can perform the following actions:

(a-1) Execute task T_i (solve and communicate with other agents).
(a-2) Pause T_i .
(a-3) Continue T_i .
(a-4) Denominate own load requirements (E_i, M_i).
(a-5) Compute $\nabla^t_{i,j}$ and check the condition $\mathbf{E}_{i,l} - \mathbf{E}_{i,j} > \epsilon$.
(a-6) Partition $T_i \rightarrow \{T_{i_1}, T_{i_2}\}$, create child agents $\{A_{i_j} = (S_{i_j}, T_{i_j})\}, j = 1, 2$.
(a-7) Migrate to $P_l \in \mathbf{N}, l \neq j$.
(a-8) Disappear.

These actions allow A_i to accomplish two goals:

(G-1) Perform computation of carried task by executing action (a-1) and then perform action (a-8) when the task is done.

(G-2) Find a better execution environment. We suggest following the algorithm utilizing actions (a-2) - (a-8).

If $Q_j = \emptyset$ **then** continue T_i
else
{ compute $\nabla^t_{i,j}$;
If $\mathbf{E}_{i,l} - \mathbf{E}_{i,j} > \epsilon$
then { pause T_i; migrate along the gradient $\nabla^t_{i,j}$; continue T_i }
else { Partition $T_i \rightarrow \{T_{i_1}, T_{i_2}\}$;
create $\{A_{i_j} = (S_{i_j}, T_{i_j})\}, j = 1, 2$; // G_t gets modified
disappear }
}.

The overall SSA intention is to accomplish the goal (G-1) in the shortest possible time. If the agent recognizes the local VCN resources as insufficient, it tries to accomplish the goal (G-2). On the other hand, P_j may force $\{A_i\}, i \in \omega_j$ to realize goal (G-2) when its performance is endangered.
The computational application has to be designed to start from one task T_1, so the initial state is given by $(\{A_1 = (S_1, T_1)\}, (A_1, \emptyset), (A_1, P_{init}))$, where P_{init} is the node on which A_1 is initially allocated. An application evolves over time because of agents individually trying to accomplish their goals.

4 Experiment

4.1 Case Study Description

As a case study we selected the mesh generator creating unstructured 3D computational meshes, which accomplishes the Delaunay property [2]. Task decomposition is based on the domain prepartitioning performed in both sequential and bisectional manners [3,9]. Experiments were conducted with six example sets - half created using bisection and half using sequential decomposition strategies. Each strategy was used to decompose the initial solid to 8, 16 and 32 subdomains (see Fig. 1). Although all examples were computed with the same needed computational mesh density, this gave us different computational and communication requirements. The experiment intends to show how a different amount of subdomains created for the same solid influences the time of the whole computation. The problem seems to be nontrivial, because if the number of computational nodes is bigger than the number of tasks, the total network resources are not fully utilized. On the other hand, increasing the amount of tasks (making a single task smaller) raises the amount of communication required for the whole job to be computed. The communication time is also extended by the need to migrate more agents and allow them to communicate.
As it will be presented in further sections, all these six decompositions gave different results, which point out system weak points and clarify ways in which to improve them. The next descriptions refer to these examples: BIS decompositions for bisection strategy; and SEKW decompositions for sequential strategy.

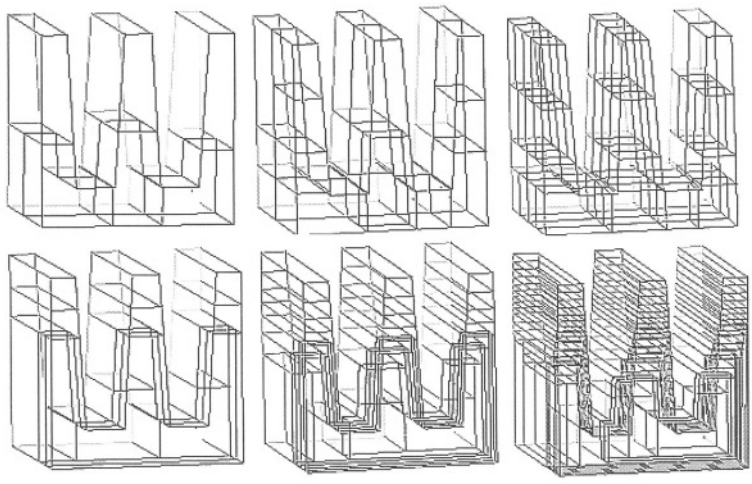


Fig. 1. Decompositions of the sample solid for 8, 16 and 32 subdomains.The upper row presents the result of bisection strategy resulting in 8, 16 and 32 subdomains.

The MAS platform was deployed within a computer network containing different amounts of PCs varying from 10 to 50 machines. The machines were ordinary desktop PCs ranging from Pentium 300Mhz up to Pentium IV with RAM from 256MB to 1GB. The machines were logically connected in the form of a 3 level deep tree.

4.2 Diffusion Rule Implementation

We utilized the following particular implementation on the agent diffusion model presented in the previous sections. All VCN nodes have the same function $perf_i(\cdot) \equiv 1$, $j = 1, \ldots, n$, which reduces L_j to E^j_{total}. The task's requirements are simplified to the case in which $M_i \equiv 0$, and E_i equals the number of subdomains processed by the agent A_i. Finally, an agent A_i currently alocated on P_j computes its binding energy coeficients using the following formula:

$$\mathbf{E}_{i,\zeta} = \frac{1}{E^{\zeta}_{total} + (1 - \delta^j_{\zeta}) E_i} \qquad \text{where } \zeta \text{ is such that } P_{\zeta} \in N_j \text{ and } \delta^j_{\zeta} = \begin{cases} 1 \text{ if } j = \zeta \\ 0 \text{ if } j \neq \zeta \end{cases} \tag{6}$$

This prevents the agent from migrating back and forth; because such binding energy also predicts what the load would be on the destination machine if an agent migrates there. Additionally, the MAS platform implements the migration as a transactional process conforming to the ACID characteristic (see [7]). A transaction guarantees that A_i binding energy remains unchanged for both source and destination VCN. Finally, the ϵ constant in the diffusion rule is set to 0, so agent A_i will migrate from P_j to P_l when $\mathbf{E}_{i,l} - \mathbf{E}_{i,j} > 0$.

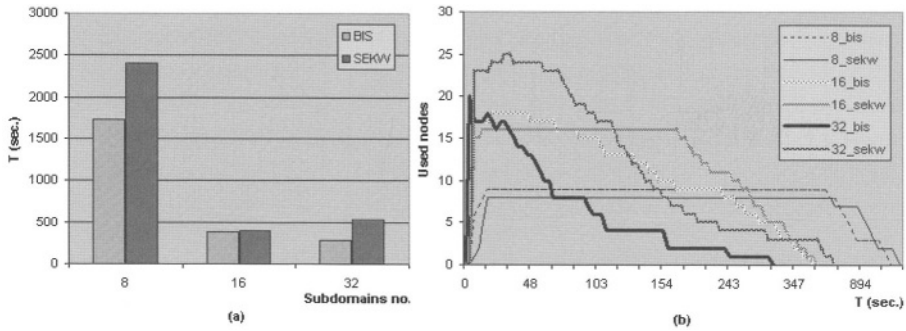


Fig. 2. (a) Total computation time grouped by decomposition types (BIS - bisection decomposition, SEKW - sequential decomposition); (b) Computer network utilization during distributed mesh generation.

4.3 Results

Figure 2 (a) presents total computation time for all six examples described previously. The optimal computation for SEKW samples has been obtained from the 16 subdomains' decomposition. The BIS strategy behaves monotonously - the bigger the amount of subdomains, the shorter computation time. The best decomposition is the 32_BIS one. Possibly having finer BIS based decompositions might lead to better results.
Figure 2 (b) shows computer network utilization vs. time during distributed mesh generation. In our experiments each computer on the network contained at most one computing agent (an agent whose load requirement was greater than zero). We may observe that:

- All cases show that diffusion based scheduling allows all agents to spread quickly among computers on the network. Each chart begins with a steep inclination. The time needed to allocate tasks is small in comparison to the whole computation time even with the actual implementation being based on high level object-oriented languages (Java) and frameworks (CORBA).
- The 32_BIS experiment shows that single agents can start generating their subdomain meshes whilst others are still diffusing among the computer network allocating themselves to less loaded computers. The chart goes up and down, because some of these first agents have such small tasks that they finish before others become allocated. In such cases, machines can be used by more than one agent. This clearly shows that the diffusion analogy allows computation and scheduling to be executed asynchronously.
- The tail at the end of charts can also be seen for experiments in which bisection decomposition was utilized. This is because of our implementation of the computational part of the Smart Solid agent, which can denominate its requirements only for CPU, but not for RAM. It caused memory swapping in some computers on the network, which consumes a lot of time during computation finalization.

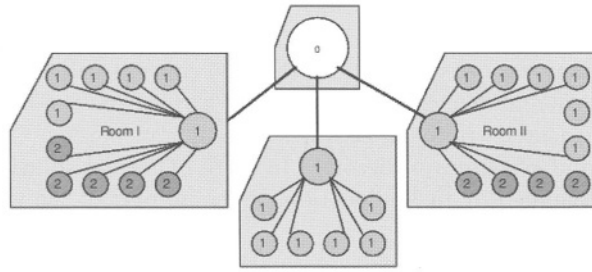


Fig. 3. Total computer network coverage by computing agents in the 32_BIS example. Each circle presents single a computing machine with the total number of computing agents inside.

Figure 3 shows how agents were allocated in the sample run of the best 32_BIS example. We notice that:

- Available resources were utilized at 96%.
- The difference between each two machines is 0 or 1; so the load is perfectly balanced according to the given local diffusion law.
- The total amount of agents created in this example was $9*2 + 20*1 = 38$. On the other hand, the peak of machines utilized for computation, as shown on Fig. 2 (b), is lower than this number. This is because Fig. 2 (b) shows agents actively computing, whereas Fig. 3 presents how agents were allocated on the network. It was checked that all machines loaded with two agents never hosted two actively computing agents at the same time. This means that when the first agent reached any of these machines, the machine load was increased at least to 1. The task was finished before the next agent asked for the machine load. This led to a short moment where each of these machines was not occupied by an active agent, and each of them was accepted as a migration destination for an active agent looking for computation power.

5 Conclusions

- Diffusion analogy as well as the MAS technology give way effectively to designing a local scheduling strategy for a distributed environment. Its effectiveness is achieved by the low complexity of local scheduling rules and the lack of intensive communication required by centralized schedulers.
- There are no synchronization points implied directly by the diffusion based scheduling solution. Each agent is independent and it carries all data required for computation. Therefore, the computation can be started by different agents at different time points. The diffusion rule implies that agents which find suitable machines will start computing whilst other may still search for free resources (machines).

- There are a couple of factors which influence the application runtime and make the scheduling to some extent unpredictable. The background computer load may vary so quickly that the load concentration computed by the MAS platform and delivered to agents may be far from the current, real one.
- The agent-oriented approach provides system transparency in terms of task allocation and resources utilization. In spite of formal solution complexity, the time consumed by task allocation is a small part of the whole computation time.
- The experiments clearly show that the current implementation needs to be extended with different server resources measurements. Such changes fit easily into the platform's design as information policy enhancements.

References

1. Agarwal R.K.: Parallel computers and large problems in industry, *Computational Methods in Applied Science.* Elsevier 1992.
2. Georg P.L.: *Automatic Mesh Generation.* John Wiley & Sons, 1991
3. Schaefer R., Toporkiewicz W., Grochowski M.: Rough partitioning of lumped structures, in *Formal Methods and Intelligent Techniques in Control, Decision Making, Multimedia and Robotics.* Polish-Japanese Institute of Information Technology Press, Warsaw, October 2000, pp. 151-166
4. Grochowski M., Schaefer R., Uhruski P.: An Agent-based Approach To a Hard Computing System - Smart Solid. *Proc. of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002),* 22-25 September 2002, Warsaw, Poland. IEEE Computer Society Press 2002, pp. 253-258.
5. Kestin, J.: *A course in thermodynamics,* Vol I, Blaisdell, 1966
6. Uhruski P., Grochowski M., Schaefer R.: Multi-agent Computing System in a Heterogeneous Network. *Proc. of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002),* 22-25 September 2002, Warsaw, Poland. IEEE Computer Society Press 2002, pp. 233-238.
7. Tanenbaum A.S.: *Distributed Operating Systems.* Prentice-Hall International, 1995
8. Luque E., Ripoll A., Cortés A., Margalef T.: A distributed diffusion method for dynamic load balancing on parallel computers. *Proc. of EUROMICRO Workshop on Parallel and Distributed Processing,* San Remo, Italy, January 1995. IEEE CS Press.
9. Schaefer R., Flasiński M., Toporkiewicz W.: Optimal Stochastic Scaling of CAE Parallel Computations. *LNAI 1424,* Springer 1998, pp.557-564

Approximation Algorithms for Scheduling Jobs with Chain Precedence Constraints

Klaus Jansen[1]* and Roberto Solis-Oba[2]**

[1] Institut für Informatik und Praktische Mathematik
Universität zu Kiel Germany
kj@informatik.uni-kiel.de
[2] Department of Computer Science
The University of Western Ontario, London, Canada
solis@csd.uwo.ca

Abstract. The problem of scheduling jobs with precedence constraints is a central problem in Scheduling Theory which arises in many industrial and scientific applications. In this paper we present a polynomial time approximation scheme for the problem of scheduling jobs with chain precedence constraints on a fixed number of uniformly related machines. Our algorithm works even if we allow "slow" machines to remain idle.

Keywords: Approximation algorithm, chains, constraints, scheduling.

1 Introduction

Scheduling problems are among the most fascinating and widely studied problems in Operations Research and Combinatorial Optimization. An important class of scheduling problems that arises in many industrial and scientific applications involves jobs that must be processed in a particular given order. These problems can be formally described as follows. A group $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of jobs needs to be processed by a set $M = \{1, 2, \ldots, m\}$ of machines. Each machine can process only one job at any given time. Every job J_i has a *length* or *processing time,* p_i, and it must be processed without interruption by one machine. Furthermore, there is a set of *precedence constraints* restricting the order in which the jobs can be processed. The goal is to schedule the jobs on the machines so that the *makespan,* schedule length, or completion time for the last job is minimized.

* Author partially supported by EU projects APPOL II, "Approximation and online algorithms", IST-2001-32007, EU project CRESCCO, "Critical resource sharing for cooperation in complex systems", IST-2001-33135, and DAAD project, "Integration of task scheduling for parallel computers into compilers and software development environment".

** Author partially supported by the Natural Sciences and Engineering Research Council of Canada grant R3050A01.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 105–112, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Scheduling jobs with precedence constraints is a fundamental problem in Scheduling Theory that has been extensively studied since the early works of Hu [7]. Despite its importance, and mainly due to its inherent complexity, there are very few results reporting algorithms to approximately solve the problem. In this paper we are interested in the case when the precedence constraints form disjoint *chains*. Every chain consists of a sequence of jobs $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(\mu)}$, that must be processed in the given order. Concerning the set of machines, we study the problem when M consists of a constant number m of uniformly related machines. This means that every machine j has a speed s_j, and so the time that it needs to process job J_i is p_i/s_j.

Using the standard three-field scheduling notation (see e.g. [9]) the above scheduling problem is denoted as $Qm|chain|C_{\max}$. Du et al. [3] showed that this problem is strongly NP-hard even with 2 machines of the same speed, while Kubiak et al. [8] proved that it is strongly NP-hard even if all jobs have unit processing time. Lenstra and Rinnooy Kan [10] showed that the problem restricted to the case of machines with the same speed cannot be approximated within a ratio smaller than 4/3 unless P=NP, even if all the jobs have unit processing times.

On the positive side, Chudak and Shmoys [2] and, independently, Chekuri and Bender [1] designed approximation algorithms with $O(\log m)$ performance guarantee for the case of arbitrary precedence constraints. For the particular case of chain precedence constraints the algorithm in [1] has a performance guarantee of 6, and a recent algorithm by Woeginger [11] achieves a performance ratio of 2. The problem without precedence constraints seems simpler, since Hochbaum and Shmoys [6] designed a polynomial time approximation scheme for it.

In this paper we present a polynomial time approximation scheme for problem $Qm|chain|C_{\max}$. Our algorithm first finds a "good" schedule for the "large" jobs. Then, every chain formed by "small" jobs is considered as a single preemptive task. This interpretation for the chains allows us to use the algorithm of Gonzalez and Sahni [4] to find an optimal schedule for the preemptive tasks. This schedule is then transformed into a feasible schedule for the original set of jobs by breaking the preemptive jobs into their constituent sets of "small" jobs. We show that a schedule of length close to the optimum can be obtained even if we allow "slow" machines to remain idle.

2 The Algorithm

Let $\mathcal{J}$ be a set of jobs with chain precedence constraints. Thus, every job J_i has at most one predecessor J_k that must be completed before the processing of J_i can start, and it also has at most one successor J_ℓ whose processing cannot start but only after J_i has been completed. The maximum number of jobs in a chain is denoted as μ.

Let s_j be the speed of machine j. For convenience, we scale the speeds and index the machines so that $1 = s_1 \leq s_2 \leq \cdots \leq s_m$. We present below a polynomial time approximation scheme for the problem $Qm|chain|C_{\max}$ restricted

to the case when $s_m/s_1 = r$, and r is a constant. Later we consider the case of machines with arbitrary speeds.

Let C^* be the makespan of an optimum schedule for the set of jobs $\mathcal{J}$, and let $D = \sum_{J_i \in \mathcal{J}} p_i$ be the total processing time of the jobs. If all jobs are processed on the fastest machine, we get a schedule of length D/r. Therefore, $D/(mr) \leq C^* \leq D/r$.

We describe below an algorithm that given a value $d \in [D/(mr), D/r]$, it either finds a schedule for $\mathcal{J}$ of length at most $d(1+O(\varepsilon))$, or it determines that no schedule of length at most d exists. By dividing the interval $[D/(mr), D/r]$ into sub-intervals of length $D\varepsilon/(mr)$, for any value $1 > \varepsilon > 0$, and using this algorithm, we can find a value $d^* \leq (1+\varepsilon)C^*$ and a schedule of length at most $(1+O(\varepsilon))C^*$.

Fix a value $d \in [D/(mr), D/r]$. Let $1 > \varepsilon > 0$, $\delta > 0$, and $\beta < 1$ be constants to be determined later. We give an overview of the algorithm below, please refer to the next sections for the details.

1. Partition $\mathcal{J}$ into 3 groups: *large,* $\mathcal{L} = \{J_i \mid p_i \geq d\beta^{\tau-1}\}$; *small,* $\mathcal{S} = \{J_i \mid p_i < d\beta^{\tau}\}$; and *medium,* $\mathcal{M} = \mathcal{J} \setminus (\mathcal{L} \cup \mathcal{S})$. The total processing time of the medium jobs must be at most δd, and $\tau \leq \frac{mr}{\delta}$.
   If such a partition does not exist, then there is no schedule for $\mathcal{J}$ of length at most d.
2. Round up the processing times of the large (medium) jobs to the nearest multiples of $\varepsilon d\beta^{\tau-1}/r$ ($\varepsilon d\beta^{\tau}/r$). Restrict the starting times of large (medium) jobs to be multiples of $\varepsilon d\beta^{\tau-1}/r$ ($\varepsilon d\beta^{\tau}/r$).
3. For each feasible schedule $\tilde{S}$ for $\mathcal{L} \cup \mathcal{M}$ of length at most $(1+3\varepsilon)d$:
   - use dynamic programming and the algorithm of Gonzalez and Sahni [4] to schedule the small jobs inside the idle intervals in $\tilde{S}$.
4. If no schedules were found in step 3, or the length of the smallest schedule is larger than $(1+(6+2r(m-1))\varepsilon)d$, then no schedule for $\mathcal{J}$ of length at most d exists. Otherwise, output the smallest length schedule found.

2.1 Grouping and Rounding

We partition the jobs into groups $G_1, G_2, \ldots, G_k$, placing in group G_i all jobs J_j of length $p_j \in (d\beta^i, d\beta^{i-1}]$. For each group G_i let $P_i = \sum_{J_j \in G_i} p_i$ be the total processing time of the group.

Let G_τ be the first group for which $P_\tau \leq \delta d$. Note that G_τ might be empty. If there is a schedule for $\mathcal{J}$ of length at most d, then $\tau \leq \frac{mr}{\delta}$ and, hence, τ is constant. To see this, note that since $d \geq C^*$, the total processing time of all the jobs $\mathcal{J}$ is at most mdr. Furthermore, since $P_i > \delta d$ for all groups G_i, $i < \tau$, then $\tau \leq \frac{mdr}{\delta d} = \frac{mr}{\delta}$. On the other hand, if each one of the $\frac{mr}{\delta}$ first groups have total processing time larger than δd, then there is no feasible schedule for $\mathcal{J}$ of length at most d. Therefore, from now on assume that $\tau \leq \frac{mr}{\delta}$.

Group G_τ partitions the set of jobs into 3 classes: the large jobs $\mathcal{L} = \{J_i \mid p_i \geq d\beta^{\tau-1}\}$, the medium jobs $\mathcal{M} = G_\tau$, and the small jobs $\mathcal{S} = \{J_i \mid p_i < d\beta^{\tau}\}$.

We simplify the problem by rounding the processing times of the jobs and constraining the set of possible starting times for the large and medium jobs. This will allow us to use enumeration and dynamic programming for constructing a good schedule for $\mathcal{J}$. Let us multiply the processing time of each large and medium job by $(1+2\varepsilon)$. This increases the length of an optimum schedule by at most the same factor. We also round the processing time of each small job up to the nearest multiple of $\varepsilon d/n$. This latter rounding might further increase the length of an optimum schedule by at most εd.

Consider a feasible schedule $\tilde{S}$ for the jobs $\mathcal{J}$ assuming these enlarged processing times. Fix the starting time of each job as determined in $\tilde{S}$ and round down, as indicated below, the processing time of each large and medium job depending on the speed of the machine that processes it. If large job J_i is scheduled on machine j, then its processing time is rounded to the smallest multiple of $\varepsilon d\beta^{\tau-1}s_j/r$ no smaller than p_i. If medium job J_k is scheduled for processing on machine j, its processing time is rounded to the smallest multiple of $\varepsilon d\beta^{\tau}s_j/r$ no smaller than p_k. Observe that this rounding does not further increase the length of an optimum schedule, and the rounded processing times are not smaller than the original processing times. Let $\tilde{p}_i$ describe the rounded processing time of job J_i.

Since the starting times of the jobs were fixed, by shrinking the processing times of the medium and large jobs we create for each one of them an idle window inside which every medium and large job can be shifted without affecting the feasibility of the schedule. Since the fastest machine has speed r, this window is of size at least $\varepsilon d\beta^{\tau-1}/r$ for the large jobs and $\varepsilon d\beta^{\tau}/r$ for the medium jobs. Thus, we can shift the large and medium jobs so that their starting times are respectively multiples of $\varepsilon d\beta^{\tau-1}/r$ and $\varepsilon d\beta^{\tau}/r$, without increasing the length of the schedule.

2.2 Restricted Schedules

For the rest of the paper we consider only *restricted schedules* in which medium (large) jobs J_i have processing times $\tilde{p}_i$ and starting times that are multiples of $\varepsilon d\beta^{\tau}/r$, $(\varepsilon d\beta^{\tau-1}/r)$. By the above discussion, if there is a schedule for $\mathcal{J}$ of length d, then an optimum restricted schedule has length at most $(1+3\varepsilon)d$. If there is no restricted schedule of length at most $(1+3\varepsilon)d$, then there is no schedule for $\mathcal{J}$ of length at most d. Let us divide the interval $[0,(1+3\varepsilon)d]$ into sub-intervals of size $\varepsilon d\beta^{\tau-1}/r$, that we call *large intervals,* and intervals of size $\beta^{\tau}\varepsilon d/r$ that we call *medium intervals.*

The number of medium jobs is $N_M \leq \lceil d\delta/(\beta^{\tau}d)\rceil = \lceil \delta/\beta^{\tau}\rceil$, and the number of large jobs is $N_L \leq \lceil D/(\beta^{\tau-1}d)\rceil \leq \lceil D/(\beta^{\tau-1}\frac{D}{mr})\rceil = \lceil \frac{mr}{\beta^{\tau-1}}\rceil$. Note that N_M and N_L are bounded by constant values. Also, the number $I_L = \lceil \left(3+\frac{1}{\varepsilon}\right) r\beta^{1-\tau}\rceil$ of large intervals and the number $I_M = \lceil \left(3+\frac{1}{\varepsilon}\right) r\beta^{-\tau}\rceil$ of medium intervals are constant values. Therefore, the number of possible restricted schedules of length at most $(1+3\varepsilon)d$ for the large and medium jobs is $(N_L)^{I_L}(N_M)^{I_M}$, which is also constant. We build all these possible schedules, and for each one of them we

use a dynamic program to assign small jobs to idle large intervals as described below.

Consider a restricted schedule $\bar{S}$ for the large and medium jobs. Note that the length of a large job is a multiple of the length of a large interval and the length of a medium job is a multiple of the length of a medium interval. Consider a large interval I. Let $\mathcal{M}_I$ be the set of medium jobs that are completely contained in interval I and have processing time strictly smaller than the length of I. Let m_I be the number of machines available in I for processing small and medium jobs $\mathcal{M}_I$.

We briefly describe the dynamic program for assigning small jobs to the large intervals of $\bar{S}$. Let v be a vector of dimension equal to the number of large intervals. Each entry of the vector stores the following $m_I + 1$ pieces of information describing the load of the small jobs and medium jobs $\mathcal{M}_I$ assigned to each large interval I: (i) the processing times of the $m_I - 1$ largest chains formed by jobs $\mathcal{M}_I$ and small jobs assigned to I, (ii) the total processing time of medium jobs $\mathcal{M}_I$ and small jobs assigned to the interval, and (iii) the total processing time of the last chain of small and medium jobs $\mathcal{M}_I$ assigned to the interval by the dynamic program (see below). The dynamic program fills up a table containing all possible vectors v. Because of the way in which we have discretized the processing times of the jobs, every entry of vector v can have $O(n^{m+1})$ possible values, and so, the aforementioned table has size $O(n^{(m+1)N_L})$, which is polynomial in n.

In the dynamic program we consider one chain at a time. For each chain, the small jobs are considered in order. We place a small job J_i in a large interval I if below condition (1) is satisfied. Let $t_1 \geq t_2 \geq \cdots \geq t_{m_I-1}$ be the lengths of the largest $m_I - 1$ maximal (sub)chains (possibly of length 1) formed by medium $\mathcal{M}_I$ and small jobs placed so far in I (some of these lengths might be zero, if the number of sub-chains assigned to I is smaller than $m_I - 1$). Let $T_j = \sum_{i=1}^{j} t_i$, and let T be the total length of small and medium jobs $\mathcal{M}_I$ placed so far in I. Let $s'_1 \geq s'_2 \geq \cdots \geq s'_{m_I}$ be the speeds of the machines available for processing small and medium jobs $\mathcal{M}_I$ in interval I, and let $S_j = \sum_{i=1}^{j} s'_i$. We place job J_i in interval I only if

$$\max\ \{\max_{1\leq j<m_I}\{T_j/S_j\}, T/S_{m_I}\} \leq \varepsilon d\beta^{\tau-1}/r + \beta^{\tau} d. \qquad (1)$$

If we think of each chain of small and medium jobs assigned to I as a single preemptive job, then condition (1) ensures, as shown in [4], that there is a preemptive schedule of length at most $\varepsilon d\beta^{\tau-1}/r$, the length of interval I. The second term in the right hand side on inequality (1) is due to the fact that in an optimum schedule a small job might cross the boundary of a large interval.

Lemma 1. *The above dynamic program produces a solution of length at most* $(1 + 3\varepsilon)d$, *if one exists.*

Proof. Consider an optimum schedule S^* for the jobs $\mathcal{J}$ and round the lengths of the jobs and their starting times as described above. The length of the schedule

increases by at most a factor of $(1+3\varepsilon)$. It is not difficult to see that the load of small and medium jobs placed in any large interval I must satisfy condition (1).

Since the processing times of small jobs are small compared to the length of a large interval, then the above preemptive schedule can be transformed into a feasible schedule for the small and medium jobs with only a slight increase in the length of a large interval as shown in the next section.

2.3 Scheduling Jobs Inside a Large Interval

Consider a large interval I and those machines that, after scheduling the medium and large jobs, have some idle time inside I. Restore each small and medium job from $\mathcal{M}_I$ to its original processing time. There are two cases that we need to consider.

Case 1: The total load of the medium jobs completely or fractionally assigned to I is at most $\frac{\delta}{\varepsilon}\Delta$ where $\Delta = \varepsilon d\beta^{\tau-1}/r$ is the length of a large interval, and $\delta = \frac{1}{2}\varepsilon\beta$.

Because of the way in which we have rounded the starting times of medium jobs, if part of a medium job is assigned to interval I then at least a fraction $\varepsilon d\beta^{\tau}/r$ of the job should be inside the interval. Therefore, the above condition on the total load of the medium jobs implies that, in this case, no medium jobs are assigned to I.

We use the algorithm of Gonzalez and Sahni [4] to find a feasible schedule for the small jobs in I. To use this algorithm we treat each task formed by small jobs as if it was a preemptive task. The algorithm schedules all the preemptive tasks inside the interval, but it might produce up to $2(m-1)$ preemptions. We identify the jobs that are preempted and for each one of them we arbitrarily select one of its preempted pieces and replace it with the entire job. This might increase the size of a large interval by at most $2(m-1)\beta^{\tau}d$.

Since there are $I_L = \lceil \left(3+\frac{1}{\varepsilon}\right) r\beta^{1-\tau} \rceil$ large intervals, scheduling the small jobs inside the large intervals might increase the overall length of the schedule by at most $2(m-1)\beta^{\tau}d\lceil \left(3+\frac{1}{\varepsilon}\right) r\beta^{1-\tau} \rceil = 2(m-1)r\beta d(1+3\varepsilon)/\varepsilon$. This increase is at most εd if we choose

$$\beta = \varepsilon^2/(2(m-1)r(1+3\varepsilon)).$$

Case 2: The load of the medium jobs in I is larger than $\frac{\delta}{\varepsilon}\Delta$. Note that since by definition the total processing time of all the medium jobs is at most δd, then this case can happen in at most $(\delta d)/(\frac{\delta}{\varepsilon}\Delta) = \varepsilon\frac{d}{\Delta}$ intervals, or in other words, this condition can happen in at most an ε fraction of the large intervals.

We split I in 3 sub-intervals. In the first subinterval we place those medium jobs that start processing before I and finish processing inside I. In the third interval we place the medium jobs that start processing inside I but finish processing after I. In the middle sub-interval we schedule the small and medium jobs from $\mathcal{M}_I$ using Gonzalez and Sahni's algorithm as described above.

The schedule produced by this algorithm increases the length of interval I by at most $(2+2(m-1)r)\Delta$, since (i) the first and third subintervals have length

at most Δ, and *(ii)* rounding the preemptive schedule of a job in the middle subinterval increases the length of the interval by at most $r\Delta$. Summing over all intervals where this case happens, we get a total increase in the length of the schedule of $(2 + 2(m-1)r)\varepsilon d$.

Adding the increases in all large intervals (cases 1 and 2), we get an overall increase in the length of the solution of at most $(3 + 2(m-1)r)\varepsilon d$.

Theorem 1. *The above algorithm runs in* $O\left(n^{m(mr/\varepsilon^2)^{m^2r^2/\varepsilon^3}}\right)$ *time and it produces a solution of length at most* $(1 + (6 + 2r(m-1))\varepsilon)d$.

2.4 Machines with Arbitrary Speeds

To deal with machines with arbitrary speeds, we find the largest speed s_ℓ, $\ell \geq 1$, such that $s_\ell \geq \frac{1}{\varepsilon}\sum_{i=1}^{\ell-1} s_i$. Observe that ℓ is always defined. Recall that machines are indexed non-decreasingly by speed. Note that if $\ell = 1$, then we can use the algorithm from the previous section because, then $s_m \leq \frac{1}{\varepsilon}\sum_{i=1}^{m-1} s_i \leq \frac{1}{\varepsilon}\left[(\frac{1}{\varepsilon}+1)\sum_{i=1}^{m-2} s_i\right] \leq \frac{1}{\varepsilon}\left(\frac{1}{\varepsilon}+1\right)^{m-2} s_1$ and, therefore, the ratio s_m/s_1 has constant value. So let us assume that $\ell > 1$.

Let S^* be an optimum schedule for the jobs $\mathcal{J}$. We show that it is possible to move all jobs to the fastest $m - \ell + 1$ machines (so that the slowest $\ell - 1$ machines remain idle) in such a way that the length of the schedule is increased by at most a factor of $(1 + \varepsilon)$.

The idea is to scan S^* starting at time zero and moving towards the end of the schedule. Every time that we find a job J_i in one of the slowest $\ell - 1$ machines, we search for job J_j belonging to the same task as J_i such that either:

- J_j is the successor of J_i placed in one of the fastest $m - \ell + 1$ machines, or
- J_j is the predecessor of J_i placed in one of the fastest $m - \ell + 1$ machines.

If job J_j does not exist, then the whole task to which J_i belongs is processed by S^* on the slowest $\ell - 1$ machines. In this case we place this task on the fastest machine, at the end of the schedule. If job J_j exists, then all jobs from the same task that appear between J_j and J_i are scheduled contiguously to J_j in the same machine where J_j is processed in S^*. Note that some jobs in other machines might need to be delayed to get a feasible schedule.

This process is repeated until all jobs scheduled by S^* in the first $\ell - 1$ machines are moved to one of the fastest machines. Let C^* be the length of schedule S^*. The total increase produced by the above algorithm in the length of the schedule is at most $\frac{s_1}{s_\ell}C^* + \cdots + \frac{s_{\ell-1}}{s_\ell}C^* \leq \frac{1}{s_\ell}\sum_{i=1}^{\ell-1} s_i C^* \leq \varepsilon C^*$.

Finally, we use the algorithm described in the previous section to schedule the jobs in the fastest $m - \ell + 1$ machines. We can do this since by the definition of ℓ, $s_m \leq \frac{1}{\varepsilon}(1 + \frac{1}{\varepsilon})^{m-\ell-1}$ and, therefore, the ratio s_m/s_ℓ has constant value. This schedule is at most $(1 + O(\varepsilon))$ times larger than the optimum.

Theorem 2. *There is a polynomial time approximation scheme for problem* $Qm|\text{chain}|C_{\max}$.

3 Conclusions

The techniques that we have presented can be used to design polynomial time approximation schemes for other scheduling problems. One of them is problem $Q|chain|C_{\max}$ for the case when the chains have fixed length and the number m of machines is arbitrary, but the fastest machine is a constant factor faster than the slowest one. Another problem is the flexible open shop problem with fixed number of machines. Here, every job consists of a sequence of operations which can be processed in any order, but at most one operation from a job can be processed at any given time. The machines used to process the jobs are unrelated, so the processing time of an operation depends on the machine that processes it.

References

1. C. Chekuri and M.A. Bender, An efficient approximation algorithm for minimizing makespan on uniformly related machines, *Journal of Algorithms,* 41 (2002), pp. 212–224.
2. F.A. Chudak and D.B. Shmoys, Approximation algorithms for precedence constrained scheduling problems on parallel machines that run at different speeds, *Journal of Algorithms* 30, 1999, pp. 323–343.
3. J. Du, J.Y.T. Leung, and G.H. Young, Scheduling chain-structured tasks to minimize makespan and mean flow time, *Information and Computation* 92, 1991, pp. 219–236.
4. T. Gonzalez and S. Sahni, *Preemptive scheduling of uniform processor systems,* Journal of the ACM, 25 (1978), pp. 92–101.
5. L.A. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk, Better approximation guarantees for job-shop scheduling, *SIAM Journal on Discrete Mathematics,* 14 (2001), pp. 67–92.
6. D.S. Hochbaum and D.B. Shmoys, A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach, *SIAM Journal on Computing,* 17 (1988), pp. 539–551.
7. T.C. Hu, Parallel sequencing and assembly line problems, *Operations Research,* 9 (1961), pp. 841–848.
8. W. Kubiak, B. Penz, and D. Trystram, Scheduling chains on uniform processors with communication delays, *Journal of Scheduling,* 5 (2002), pp. 459-476 .
9. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Sequencing and scheduling: Algorithms and complexity. In: S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.) *Logistics of Production and Inventory,* Handbooks in Operations Research and Management Science 4, North Holland, Amsterdam, pp. 445–522.
10. J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity of scheduling under precedence constraints, *Operations Research* 26, pp. 22–35.
11. G. Woeginger, A comment on scheduling uniform machines under chain-type precedence constraints, *Operations Research Letters,* 26 (3), 2000, pp. 107–109.

Combining Vector Quantization and Ant-Colony Algorithm for Mesh-Partitioning

Jurij Šilc[1], Peter Korošec[1], and Borut Robič[2]

[1] Jožef Stefan Institute, Computer Systems Department,
Jamova 19, 1000 Ljubljana, Slovenia
{peter.korosec, jurij.silc}@ijs.si
http://csd.ijs.si
[2] University of Ljubljana, Faculty of Computer and Information Science,
Tržaška 25, 1000 Ljubljana, Slovenia
borut.robic@fri.uni-lj.si

Abstract. We present heuristic mesh-partitioning method build on the ant-colony algorithm (ACA) in order to improve the quality of the mesh partitions. The method focuses on improving the initial partition that is submitted to the ACA. The method is experimentally compared with the well-known mesh-partitioning programs pMETIS 4.0 and Chaco 2.0.

1 Introduction

Parallel computer systems are suitable for solving computationally intensive numerical methods, such as finite-element methods (FEMs) for solving partial differential equations. This type of a program and the associated data are distributed among the processors. The aim is to find a distribution where the communication is as low as possible. This leads to the mesh-partitioning optimization problem, which is known to be *NP*-hard. One of the new, competing heuristics for solving this problem is the so-called ant-colony algorithm (ACA).

In this paper we present heuristic mesh-partitioning method build on the ant-colony algorithm (ACA) in order to improve the quality of the mesh partitions. The method focuses on improving the initial partition that is submitted to the ACA.

2 Mesh-Partitioning Problem

The *mesh-partitioning problem* for a non-overlapping partition of an unstructured FEM mesh can be formulated as a graph-partitioning problem. The FEM mesh is associated with a graph $G(V, E)$ that consists of vertices and edges connecting them, where every vertex v has weight 1 and corresponds to an element of the mesh. An edge between two vertices indicates that the two corresponding elements are neighbors. More precisely, let $G(V, E)$ be an undirected graph consisting of a non-empty set V of vertices and a set $E \subseteq V \times V$ of edges. A k-partition D of G comprises k mutually disjointed subsets $D_1, D_2, \ldots, D_k$ (called domains)

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 113–118, 2004.
© Springer-Verlag Berlin Heidelberg 2004

of V whose union is V. The set of edges that connect the different domains of a partition D is called an edge-cut. A partition D is balanced if the sizes of the domains are roughly the same, i.e., if $b(D) = \max_{1 \leq i \leq k} |D_i| - \min_{1 \leq i \leq k} |D_i| \approx 0$. The graph-partitioning problem is to find a balanced partition with a minimum edge-cut $\zeta(D)$.

3 Ant-Colony Algorithm

The *Ant-Colony Algorithm* (ACA) [4] uses a metaheuristic approach for solving hard combinatorial optimization problems. The idea for the ACA came from observing real ants while they gathered food and communicated via pheromone trails. The *artificial ants* used in the ACA are stochastic constructive procedures that build solutions with the help of probability. While building solutions they consider the heuristic information of the problem and the pheromone trails that are being dynamically changed.

```
while termination condition not satisfied do
   Schedule_Activities
      Manage_Ants_Activity();
      Evaporate_Pheromone();
      Daemon_Actions();
   end Schedule_Activities
endwhile
```

Fig. 1. The pseudocode of the ACA

The concept of foraging and gathering food (see Fig.) is simulated by the `Manage-Ants_Activity` procedure. `Evaporate_Pheromone` simulates the reduction of pheromone intensity. This is needed so that we avoid a too rapid convergence of the algorithm towards a sub-optimal solution. The optional procedure `Daemon_Actions` can be used to implement centralized actions that cannot be performed by ants. The construct `Schedule_Activities` does not specify how the activities in its body should be scheduled or synchronized; instead, it is up to the programmer to specify their interaction (e.g., parallel, independent).

4 Vector Quantization

The *Vector Quantization* (VQ) method [3] is a stochastic approximation method that uses the basic structure of the input vectors for solving a specific problem (for example, data compression). In other words, the input space is divided into a finite number of distinct regions (domains) and for each region there is a representative vector. When a mapping function (device) receives a new input vector it maps it into a region with which this vector is represented best. This is a

simple example of some sort of compression. Of course this is only one possibility of the use of this method. In our case we used it as a mapping device for our mesh-partitioning problem. The mesh vertices are usually locally connected to their neighbors. Now we can treat the position of each mesh vertex as an input vector and each domain in our partition as a region in input space. We try to divide our "mesh" space into domains, so that the size (number of vertices) of each domain is approximately the same, with as few as possible connections between the domains.

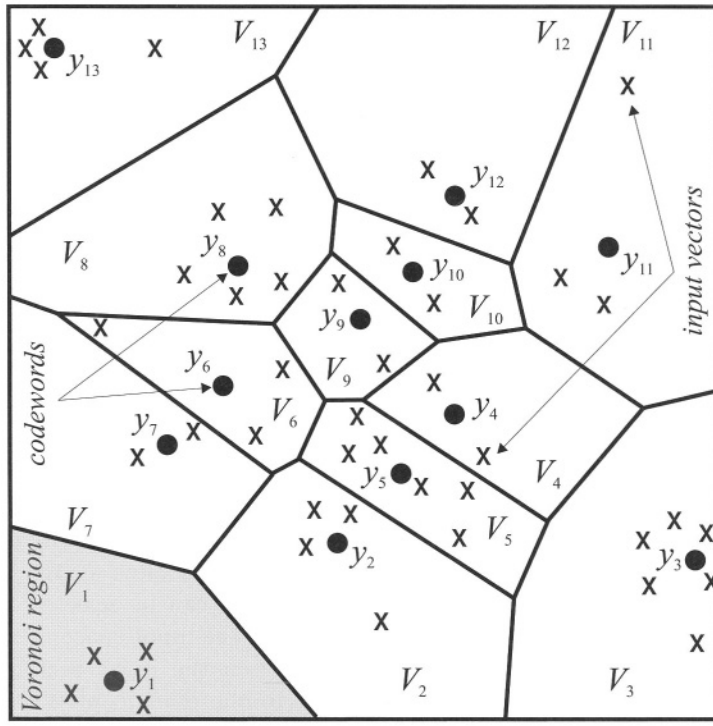


Fig. 2. Vector quantization

A vector quantizer maps ℓ-dimensional vectors in the vector space $\mathcal{R}^\ell$ into a finite set of vectors $Y = \{y_i : i = 1, 2, \ldots, k\}$. Each vector y_i is called a *codeword* and the set of all the codewords is called a *codebook.* Associated with each codeword y_i is a nearest-neighbor region called the *Voronoi region,* and it is defined by: $V_i = \{x \in \mathcal{R}^\ell : \| x - y_i \| \leq \| x - y_j \|, \forall j \neq i\}$. The set of Voronoi regions partition the entire space $\mathcal{R}^\ell$ such that: $\left(\bigcup_{i=1}^{k} V_i = \mathcal{R}^\ell\right) \wedge \left(\bigcap_{i=1}^{k} V_i = \emptyset\right)$.

The VQ consists of the following six steps:

Step 1: Determine the number of domains k.

Step 2: Read the input graph and its coordinates. The input vector consists of three elements. The first two are the vertex coordinates and the third

Table 1. Comparison of the algorithms where the meshes are partitioned into two domains ($k = 2$)

			Chaco 2.0	pMetis 4.0	VQ+ACA		
Graph	$\|V\|$	$\|E\|$	ζ	ζ	ζ	$\bar{\zeta}$	$\pm\sigma$
3elt	4720	13722	124	98	90	90.0	0.0
3elt_dual	9000	13287	70	70	45	69.5	10.7
airfoil1	4253	12289	82	85	81	82.1	0.6
airfoil1_dual	8034	11813	60	40	40	51.0	4.4
big	15606	45878	242	165	139	142.4	4.5
big_dual	30269	44929	92	92	77	98.7	9.6
crack	10240	30380	209	206	196	205.8	13.8
crack_dual	20141	30043	130	101	88	93.7	2.3
grid1	252	476	26	20	18	18.8	0.0
grid1_dual	224	420	16	16	16	16.0	0.0
grid2	3296	6432	38	37	34	34.0	0.0
grid2_dual	3136	6112	35	32	32	32.0	0.0
netz4504	1961	2578	25	26	24	24.0	0.0
netz4504_dual	615	1171	21	21	19	19.0	0.0
U1000.5	1000	2394	10	1	2	8.0	3.2
U1000.10	1000	4696	115	56	39	56.1	10.0
U1000.20	1000	9339	294	253	221	261.6	15.6
ukerbel	5981	7852	30	28	28	28.0	0.0
ukerbel_dual	1866	3538	25	25	22	22.0	0.0
whitaker3	9800	28989	135	128	127	128.1	0.5
whitaker3_dual	19190	28581	82	74	65	66.5	0.8

is the vertex density. By density we mean how close together its connected neighbors are. The closer they are, the higher is the value for the density.

Step 3: Select k codewords. The initial codewords can be selected randomly from the input vectors or as random points in the input space.

Step 4: Calculate the Euclidian distance between the input vector and the codewords. The input vector is assigned to the domain of the codeword that returns the minimum value according to the function: $f(x, y_i) := \varepsilon(x, y_i) - \widetilde{\zeta}_i - \widetilde{\beta}_i$, where $\varepsilon(x, y_i)$ represents a function that calculates the Euclidian distance between x and y_i, $\widetilde{\zeta}_i$ represents the change in the edge-cut if x belonged to the i-th domain and $\widetilde{\beta}_i$ represented the difference between the number of vertices in the largest and the i-th domains.

Step 5: Compute the new set of codewords. We add up all the vectors in the domain and divide the summation by the number of vectors in the domain: $y_i = \frac{1}{m} \sum_{j=1}^{m} x_{ij}$, where m represents the number of input vectors in the i-th domain.

Step 6: Repeat steps 4 and 5 until the values of the codewords converge, usually to a suboptimal solution.

For easier understanding an example of this is shown in Fig. 1. Here we used a two-dimensional graph ($\ell = 2$), but it can easily be expanded to any

Table 2. Comparison of the algorithms where the meshes are partitioned into four domains ($k = 4$)

			Chaco 2.0	pMetis 4.0	VQ+ACA		
Graph	$\|V\|$	$\|E\|$	ζ	ζ	ζ	$\bar{\zeta}$	$\pm\sigma$
3elt	4720	13722	258	252	212	253.0	33.8
3elt_dual	9000	13287	130	120	154	180.4	11.3
airfoil1	4253	12289	182	179	190	204.0	9.2
airfoil1_dual	8034	11813	111	84	110	117.5	4.6
big	15606	45878	416	405	382	428.7	4.6
big_dual	30269	44929	219	196	222	238.1	4.5
crack	10240	30380	457	458	371	379.5	3.7
crack_dual	20141	30043	228	201	164	169.5	3.1
grid1	252	476	48	40	38	38.2	0.4
grid1_dual	224	420	37	35	35	35.0	0.0
grid2	3296	6432	106	121	92	101.6	1.9
grid2_dual	3136	6112	99	91	96	97.0	0.2
netz4504	1961	2578	66	62	49	51.2	1.7
netz4504_dual	615	1171	54	49	44	46.2	1.4
U1000.5	1000	2394	20	6	12	19.1	2.0
U1000.10	1000	4696	200	108	107	123.5	4.8
U1000.20	1000	9339	664	515	497	535.5	15.6
ukerbel	5981	7852	82	64	61	61.9	1.2
ukerbel_dual	1866	3538	56	51	48	48.0	0.2
whitaker3	9800	28989	439	424	383	388.0	2.2
whitaker3_dual	19190	28581	251	210	195	199.1	2.3

other number of dimensions. We can see 45 input vectors that are divided into $k = 13$ domains (Voronoi regions $V_1, V_2, \ldots, V_{13}$) and are represented with the codewords $y_1, y_2, \ldots, y_{13}$.

5 Combining VQ and ACA

5.1 Basic Idea

So what we have done is a merger of the VQ and the ACA into a single algorithm (VQ+ACA). With the VQ we compute a partition, which is then used as a starting partition for the ACA. With the ACA we refine this partition so that the best possible result is obtained.

5.2 Experimental Results

In this section we present and discuss the results of the experimental evaluation of our VQ+ACA algorithm in comparison to the well-known partitioning programs pMetis 4.0 [2] and Chaco 2.0 [1].

The test graphs used in our experiment were acquired from the Graph Collection Web page at the University of Paderborn (`www.uni-paderborn.de/cs/-ag-monien/RESEARCH/PART/graphs.html`) and are described in Tables 1 and 2.

We partitioned each of the graphs into two and four domains ($k = 2, 4$). Each score is described with the best obtained edge-cut ζ. Here we must mention that the balance b was kept inside 0.2% of the $|V|$.

Because the VQ+ACA is stochastic algorithm we ran it 32 times on each graph and calculated the average edge-cut value $\bar{\zeta}$ and its standard deviation ($\pm\sigma$). The results of our experiment are shown in Tables 1 and 2.

Tables 1 and 2 show that in most cases the best partition was obtained with the VQ+ACA algorithm.

6 Conclusion and Future Work

The graph-partitioning problem is an important component for mesh partitioning in the domain-decomposition method. The ACA uses a metaheuristic approach for solving hard combinatorial optimization problems. The goals of this paper were to investigate the VQ and the ACA for mesh partitioning. We merge of the VQ and the ACA into a single algorithm VQ+ACA.

An obvious improvement of our algorithm would be to merge the two methods into one. To do this one could apply VQ to produce a starting partition, then coarsen the graph to some extent (to a much smaller extend than in the original multilevel method), and then use the multilevel method to refine the previously (with VQ) obtained partition.

References

1. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. Proc. Supercomputing (1985)
2. Karypis, G., Kumar, V.: Multilevel k-way partioning scheme for irregular graphs. J. Parallel Distr. Com. **48** (1998) 96–129
3. Linde, Y., Buzo, A., Gray, R. M.: An algorithm for vector quantizer design. IEEE Trans. Commun. **28** (1980) 84–95
4. Meuleau, N., Dorigo, M.: Ant colony optimization and stochastic gradient descent. Artif. Life **8** (2002) 103–121

Wavelet-Neuronal Resource Load Prediction for Multiprocessor Environment

Paweł Hajto and Marcin Skrzypek

Institute of Computer Science,
Jagiellonian University,
Poland, Nawojki 11, 30-072 Cracow
{hajto,skrzypek}@ii.uj.edu.pl

Abstract. The paper presents a wavelet-neuronal method for prediction of resource load in multiprocessor environment. Research was performed on multiprocessor NUMA *(Non-Uniform Memory Access)* machine, but prediction algorithm can be successfully used also in virtual multiprocessor system. Prediction can improve dynamic load balancing methods.

1 Introduction

Prediction of resource load in multiprocessor environment is highly motivated by large scale computations scheduling that are both: time and memory consuming. Allocation of small tasks can be much more productive with greedy heuristic.

Dynamic load balancing applied to multiprocessor system can considerably improve its performance [7],[8]. It is valid not only for Virtual Multiprocessor Systems (e.g. Beowulf clusters), but also for mainframe NUMA units. Prediction of system resources usage can make load balancing more efficient - performance can be optimized before it gets worse by communication overhead, slower RAM or use of swap files.

Efficient task scheduling politics has to be based on the knowledge of system parameters and their time variability. We propose the following parameters to be tracked: CPU (utilization of the system idle process), RAM (size of unallocated Random Access Memory), swap file activity (number of pages swapped in and out) and Network Communication (volume of data transferred in and out on all network interfaces).

Memory demanding parallel computations like CAD *(Computer Aided Design)* solvers [2] that use Finite Element Method need also RAM and swap forecast to be perfectly scheduled. In case of slower distant RAM memory or especially swap files, performance dramatically decreases. In case of high demand for memory, new processes could be queued for execution. They would start their execution later, but in overall the system would increase its performance. Swap file memory is usually about 3 levels of magnitude slower than RAM.

The last proposed parameter, Network Communication is of key importance in case of distributed applications that need to exchange large amount of data (this parameter is rather unimportant for NUMA units).

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 119–124, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Forecast of environment parameters can be used for many purposes. An example would be high level OS *(Operating System)* task scheduling or allocation/dynamic allocation of user tasks. It can be applied also to assist information policy in MAS'es *(Multi-Agent Systems).* Framework of the MAS can provide this information to agents and allow them to make decisions on possible migrations.

This paper presents method for forecasting resource parameters. Research was performed on NUMA supercomputer SGI 2800 and PC desktop computer. CPU load of the SGI was close to 100% which is why we focused on RAM and swap file usage forecasts. However, the presented method can be successfully applied to other needs.

2 Wavelets and Artificial Neural Networks Based Load Analysis

In this chapter we will describe a deterministic approach to time series prediction and a computer system's load balance. Running processes have influence on computer's resources. Their impact can be measured by sampling at a particular period. Some of these effects are, as shown in this paper, not of a random nature, which makes a prediction possible.

The data used in this work were collected from two computers. The first one was SGI 2800 named "Grizzly" from the Academic Computer Center "Cyfronet" in Cracow, Poland. The machine has 128 CPUs, 45 GB RAM, over 300GB disk space installed and driven by IRIX 6.5. The second one was a PC with one Intel Pentium III 1.1 GHz processor and 256 MB RAM under the control of Windows 2000.

The forecasting method is based on wavelet time series analysis and an autoregressive neuronal model that serves as a prediction function. It was formerly used by the author for the economic time series forecasting (see [6]).

2.1 Wavelet-Neuronal Prediction Model

Wavelet theory evolved in the mid-1980s. It can be considered a Fourier analysis extension, especially in the scope of signal processing. In this work wavelet filters are a tool for separating variations from a signal $s = [x_1, \ldots, x_N]$. This is achieved through convolution of s and low- or high-pass wavelet filter coefficients, which yields two new signals, containing low and high frequency changes from s. This process can be repeated to extract desired frequency intervals, which is the basic concept of Discrete Wavelet Transform (DWT) and Mallat's pyramid algorithm. If the filters are orthogonal or biorthogonal an inverse procedure (iDWT) is possible (i.e. s can be perfectly reconstructed from all of its descendent filtrations). The reader can find these issues discussed in [1] or [5]. A good source for algorithms and implementation details is [4].

We define the problem of value forecasting relying on the nonlinear autoregression model. Given data samples $x_1, \ldots, x_N$ at consecutive time points (e.g. history of network traffic in Mbps) we assume that x_i depends on previous k

samples: $x_{i-1}, \ldots, x_{i-k}$, $k > 0$, $i = k+1 \ldots N$. In other words, we would like to have a function $f : \mathbb{R}^k \to \mathbb{R}$, such as: $x_i = f(x_{i-1}, \ldots, x_{i-k})$.

If f existed, it could be used to estimate forthcoming, unknown samples. Of course, the future is generally unpredictable, but one can try to observe regularity in the data and find an approximation of f. It is the moment when artificial neural networks (ANNs) come into focus with their generalization capabilities. The problem of an appropriate k choice can be solved through BDS determinism test (see [9]).

The obvious solution would be to construct and train a neural network to approximate f, unfortunately a simple one-network neuronal model does not work well approximating a complex phenomenon. Our wavelet approach consists of three steps. The first, preprocessing step bases on applying wavelet transform techniques (Mallat's algorithm) to input data, which splits it into separated frequency bands. During the second step, oscillations in each band are neural forecasted. Finally, predicted values of each frequency range are added which builds the desired forecast.

This process is shown on fig. 1. Let us notice a possible optimization. When forecasting only a few (e.g. 1-5) samples there is no need to predict data in low frequency bands. For example, if the input signal's length is 2048 samples, the next (predicted) 5 values cannot generate oscillations 512-1024 samples long. They will have influence on high, but none or very little on low and mid frequency coefficients. This allows to reduce amount of networks and so time needed to train them.

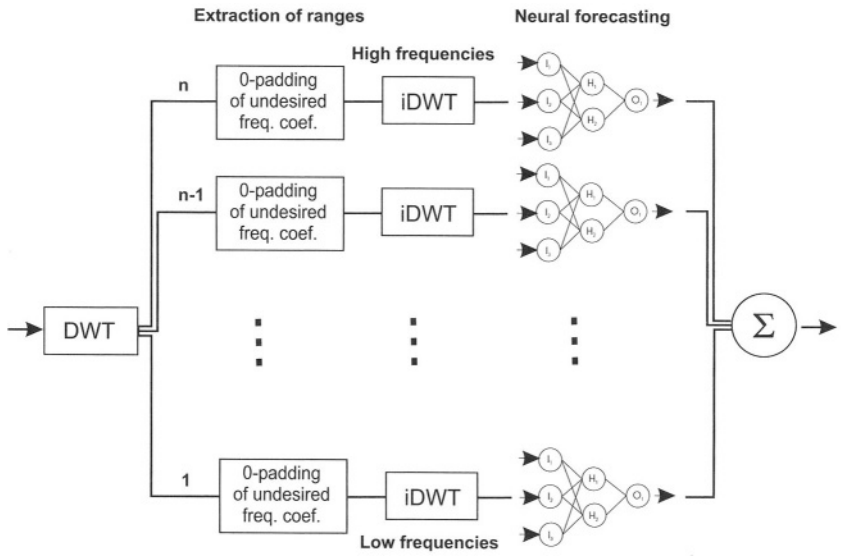


Fig. 1. Wavelet-neuronal model, $n = \lceil log_2 N \rceil$ due to properties of DWT.

2.2 Forecast Model and Test Configuration

The described prediction model was applied to data acquired from a SGI 2800 "Grizzly" and a Windows 2000 workstation. During nearly a week, 591000 data samples were collected in 1 second intervals from both machines. The samples contained load information, memory, swap, OS and network activity. In case of Grizzly CPU load information was collected from 4 randomly chosen processors.

Two different tests were prepared. First, all of the 591000 values of every variable were averaged in consecutive groups of 600 samples. This resulted in 985 numbers representing mean load, activity etc. in 10 minutes intervals. For the second test 86400 sequential samples from the initial 591000 were averaged in 85 seconds intervals. The purpose of such a proceeding was to remove noise, some random-like peaks and check influence on model's forecasting ability.

The prediction method's test consisted of forecasting average load for the next 10 minutes (85 s.). This test was repeated 500 times for each variable, each time predicting a subsequent sample and basing on different input data. For example: sample s_{890} was forecasted from $s_1, \ldots, s_{889}$; s_{891} from $s_1, \ldots, s_{890}$, etc.

There were four multilayer ANNs used, modeling highest four frequency ranges. They had 6 input neurons. The number of hidden neurons depended on each frequency band and ranged from 1 (lowest of the 4 frequency bands) to 6 (highest frequencies). These architectures were estimated using JavaNNS and Optimal Brain Surgeon algorithm (see [3]).

An orthogonal spline wavelet filter (58 coefficients) was applied to transform the data. This wavelet family gave the highest prediction accuracy. Also Daubechies 4, 8, 12 and 20 filters were tested of which D20 worked best.

The following error measures were used:

1. *DIR* - the percentage of correctly guessed *directions* of value alteration. A *DIR* = 50% indicates no forecasting abilities; a significantly higher value is very advisable.
2. Mean Absolute Percentage Error, $MAPE = \frac{1}{P}\sum_{i=1}^{P}\left|\frac{s_i - \tilde{s}_i}{s_i}\right| * 100$, where s_i is the real value, $\tilde{s}_i$ its prediction, $P = 500$ the number of forecasts.

2.3 Results

A brief examination of log files demonstrated no difficulty in forecasting Grizzly's CPUs load. The machine performs modeling of biological molecules, medical projects and crystallography applications. That is why CPUs power is used up to 100% most of the time. On the contrary, processor usage of the PC was very low. The computer is used by a software developer about 8 hours a day.

Statistics of other forecasts are presented in tables 1 - 3. Columns *Data 1* present results obtained from original samples averaged in 10 minutes intervals, while columns *Data 2* results obtained from 86400 original samples averaged in 85 seconds intervals. Network Activity in Table 2 represents volume of incoming and outgoing data on all network interfaces, whereas Swap Activity is the number of pages swapped in and out.

Table 1. Memory usage, SGI 2800

	Free		Kernel		User	
Error	Data 1	Data 2	Data 1	Data 2	Data 1	Data 2
MAPE	5,809%	4,443%	0,527%	0,233%	1,216%	0,824%
DIR	62,292%	55,625%	51,667%	53,125%	49,792%	58,125%

Table 2. Network and swap activity, SGI 2800

	Network		Swap	
Error	Data 1	Data 2	Data 1	Data 2
MAPE	10,608%	12,831%	10,375%	19,876%
DIR	55,625%	58,75%	58,337%	65,833%

Table 3. OS activity, SGI 2800

	No. of fork calls		No. of exec calls		No. of file operations	
Error	Data 1	Data 2	Data 1	Data 2	Data 1	Data 2
MAPE	6,569%	15,22%	10,019%	22,942%	70,791%	54,439%
DIR	60%	57,5%	49,792%	58,125%	58,337%	55,417%

Table 4. CPU and memory usage, PC

	CPU		Memory	
Error	Data 1	Data 2	Data 1	Data 2
MAPE	12,286%	21,494%	2,462%	1,783%
DIR	51,25%	55,417%	58,75%	60,625%

Table 5. Network and swap activity, PC

	Network		Swap	
Error	Data 1	Data 2	Data 1	Data 2
MAPE	6,425%	26,314%	33,062%	43,803%
DIR	54,167%	55,833%	63,958%	58,542%

Table 6. OS activity, PC

	No. of processes		No. of threads		No. of file operations	
Error	Data 1	Data 2	Data 1	Data 2	Data 1	Data 2
MAPE	0,696%	0,727%	0,446%	0,84%	2,462%	1,783%
DIR	58,333%	62,083%	56,875%	53,542%	58,75%	60,625%

3 Concluding Remarks

The above results indicate that the prediction algorithm forecasts changes in an amount of free memory or swap activity with accuracy *DIR* > 60%. Changes in network traffic, memory allocated by users and OS activity are predicted with slightly less accuracy (*DIRs* approx. 58,3%). Obviously, these are statistical results determined by particular environment.

Considering these facts: computing DWT of 2048 samples is almost immediate; training 4 ANNs used in this work lasts for approx. 20 s on a 1.5GHz Pentium processor; they can generate several forecasts with the above described accuracies - it seems that the wavelet-neuronal method can take part in an optimization of task allocation in a Multi-Agent System as well as NUMA units. Our model's computational costs are low compared with benefits. Moreover our model is able to predict significant changes in a machine's load.

Further research of the correlation of wavelet shapes, neural networks and forecasting abilities is intended.

Acknowledgements. We would like to thank prof. Robert Schaefer for comments and suggestions, ACK "Cyfronet" for granting access to "Grizzly" and our friend Mike Moore for grammar revision.

References

1. Aboufadel E., Schlicker S.: Discovering Wavelets, *John Wiley & Sons* (1999).
2. Grochowski M., Schaefer R., Toporkiewicz W., Uhruski P.: An Agent-based Approach To a Hard Computing System - Smart Solid, *Proc. of the International Conference on Parallel Computing in Electrical Engineering,* Warsaw, Poland 22-25 September 2002 253-258.
3. Hassibi B. and Stork D.G., Second order derivatives for network pruning: Optimal brain surgeon, in Advances in Neural Information Processing Systems, Vol 5, S.J. Hanson, J.D. Cowan and C.L. Giles, Eds., pp. 164-172, *Morgan-Kaufmann,* Apr 1993.
4. Jensen A., Cour-Harbo A.: Ripples in Mathematics. The Discrete Wavelet Transform, *Springer-Verlag* Berlin Heidelberg (2001).
5. Mallat S.: Multiresolution Approximation and Wavelet Orthonormal Bases of $L_2(\mathbb{R})$, *Trans. Am. Math. Soc.,* 315 (1989), 69-88.
6. Hajto P.: A Neural Economic Time Series Prediction With The Use of Wavelet Analysis, *Schedae Informaticae* No. 11, Jagiellonian University Press, Cracow (2002), 115-132.
7. Onderka Z., Schaefer R.: Markov Chain Based Management of Large Scale Distributed Computations of Earthen Dam Leakages, *Lecture Notes in Computer Science 1215,* Springer-Verlag, 1997, pp. 49-64.
8. Williams R.D.: Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations, *Concurrency: Practice and Experience,* vol. 3(5), 457-481 (October 1991).
9. W. A. Brock, W. D. Dechert, J. A. Scheinkman: A test for independence based on the correlation integral, *Technical report. University of Wisconsin-Madison* (1986).

Fault-Tolerant Scheduling in Distributed Real-Time Systems*

Nguyen Duc Thai

Institute of Informatics, Slovak Academy of Sciences, Dubravska cesta 9, 845 07 Bratislava, Slovakia
thai.ui@savba.sk

Abstract. Enhancing the reliability of a system executing real-time jobs is, in many cases, one of the most important design goals. A dynamic reconfigurable system offers an approach for improvement of reliability. To achieve high reliability, the most suitable recovery action must be used when a fault occurs, which means that some kind of optimal recovery strategy should be followed. In this paper, we suggest a method to analyze the schedulability of a task set in distributed systems and enhance the reliability of the systems.

1 Introduction

Real-time systems are those systems in which the correctness of the systems depends not only on the logical results but also on the time at which the results are produced [1]. Real-time tasks, which have more deterministic properties when compared to tasks in general computer systems [5] [6] [7] are characterized by deadline, release time and execution time. The design of a real-time system needs to ensure high levels of reliability and fault tolerance. The system should be able to tolerate various kinds of failures and still finish the mission successfully. High levels of reliability can be achieved using massive hardware redundancy, fault tolerant scheduling schemes and incorporation recovery strategies, in order to minimize the probability of failure. The reliability $R(t)$ of a system is the probability that a system will provide the specified service until time t, given that the system was operational at $t = t_0$ [2]. If a system has a constant failure rate of failures/hour, then the reliability at time t is given by $R(t) = e^{-\lambda(t-t_0)}$

This paper is concerned with providing a method to improve reliability and productivity of real-time systems. Section 2 describes the system model and assumptions used in the paper. In section 3 we propose a method to improve reliability of the system in order to get better performance. Section 4 shows experimental results and Section 5 concludes the paper.

* This work was supported by the Slovak Scientific Grant Agency within Research Project No. 2/3132/23

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 125–130, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 System Model

The computational model is described as follows. Each node of the system represents a computing element on which the tasks run and it exchanges messages with other nodes via the network. The nodes run in a master-slave configuration. All specified system tasks arrive at the master node and are allocated to the slave nodes; these tasks are executed locally on each node according to the scheduling algorithms, for example Rate Monotonic (RM), Earliest Deadline First (EDF) [3] [4], etc. The master is also responsible for detecting faults and overseeing the entire system recovery process. Each node maintains a list of tasks that are assigned to it, a queue of running tasks, and an event queue. Each of the computing nodes in the system can be modeled as a single entity that is self sufficient in itself. It has private memory, its own scheduling algorithm, checkpointing scheme, etc. Each of these can be individually changed without affecting the others. Once we have such a collection of nodes, we need to have some mechanism by which they work together. One of the most common solutions is to have a master-slave relationship among the nodes. The tasks in our system are characterized by their execution time, deadline, release times, periodicity, message passing capacities, etc.

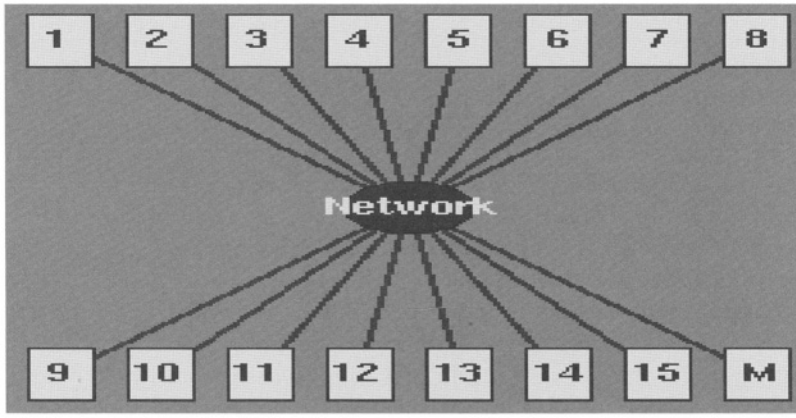


Fig. 1. An example: A system with 16 nodes

The performance of a dynamic real-time system can be improved through dynamic system reconfiguration and recovery selection under the control of the proper policies. Different actions (recovery or reconfiguration) have different short-term efficiencies (or reliability) and long-term effects on the system. A selected policy should take into account both of these factors to achieve better overall system behavior. A real-time system can fail due to the missing of a single critical task's deadline. Therefore, the selected control policies should be able to take care of every critical task during the system mission. This means

that a decision is made not only on the average system behavior values, but also instantaneous system operational information such as workload.

In order to analyze real-time systems, we use Markovian Decision Process (MDP). The MDP, provides a general method, which models a dynamic system and evaluates the optimal control policies. The decision state for our considered system has to reflect not only the average system behavior values but also the instantaneous system operational status such as workload. This will result in a prohibitively large decision state space, which prevents any practical analysis.

Markovian Decision Processes are stochastic processes that describe the evolution dynamic systems controlled by sequences of decisions or actions.

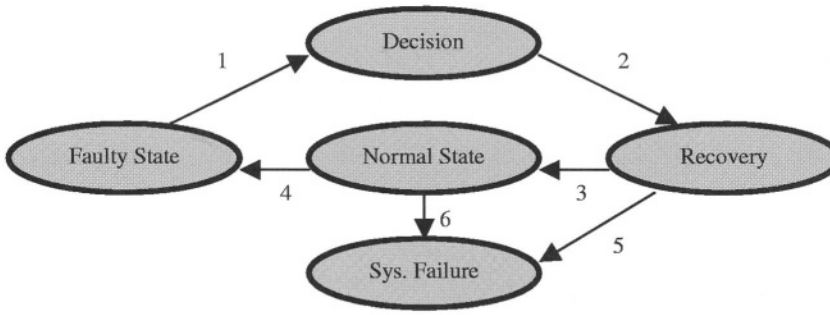


Fig. 2. The Time Phases of A State Equation in MDP

Figure 2 describes behavior of the system. Every state equation starts with transition 1 (decision) and is ended by transition 4 (fault arrival). Let:

- S denotes the set of all possible system configuration states,
- $LP(s), s \in S$ denote all possible workload patterns related to each configuration state. The workload pattern indicates the quantity and service status of the tasks currently in the system.
- $FP(s)$ denote the set of all the fault-patterns associated with s. A fault pattern indicates the system fault status and the fault location. The system state space can be expressed as:

$$Z = \{z\} = \{(s, l_p, f_p) | l_p \in LP(s), f_p \in FP(s), s \in S\} \quad (1)$$

And the associated MDP state space is:

$$Z^{MDP} = \{z\} = \{(s, l_p, f_p) | l_p \neq f_{p0}, f_p \in FP(s), l_p \in LP(s), s \in S\} \quad (2)$$

With the consideration of the time discretized backward introduction, the computational complexity is very high. Therefore, the direct MDP approach is not practical for the analysis of any medium or large sized system.

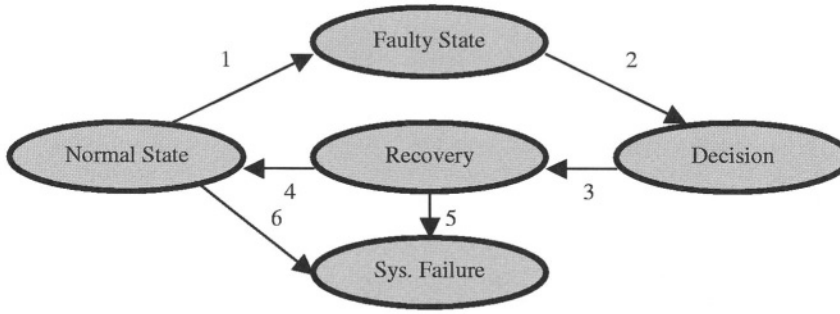


Fig. 3. The Time Phases of A State Equation in MDP'

3 Scheduling Algorithms

According to the result above, it is the good idea to propose a new method in order to get better complexity and enhance the reliability and productivity of the systems.

We consider a real-time system with dynamic recovery and reconfiguration strategies starts in a normal system state. During the normal operation, the system may possess variable workload due to the random arrivals of aperiodic tasks and variation in task service time. The deadline-miss rate will vary with the workload. During system operation, high workload can be dealt with by employing some spare resources, thus reducing the system fault tolerance. Upon the occurrence of a fault, a decision is made on the selection of the recovery procedure to be applied. The recovery duration, task-missing rate during recovery, and the state after recovery are functions of the system faulty state, pre-fault workload and the selected action. Due to the timing constraint of the real-time task service and disastrous consequence of missing critical task deadlines, a policy must take care of all the individual critical tasks in a system mission. Therefore, a decision has to take the instantaneous workload information into account as well as other average values.

Our attempt: to get a successful recovery that will bring the system back to a new normal configuration state with a certain initial workload, which depends only on this system configuration state. We call this strategy MDP'. According to Figure 2, to describe the transition at the decision epochs from one state z_0 to another state $z_1, z_0, z_1 \in Z^{MDP}$ in the MDP model, we need

$$|MDP| = |Z|^2.|A| = (|S|.|FP|)^2.|A| \tag{3}$$

transition expression Q-functions, which are given as

$$Q(u, z_1|z_0, a) = Q_2(u, z_1.s|z_0, a).Q_1(u, z_1); z_0, z_1 \in Z^{MDP} \tag{4}$$

In MDP' model, we need $|S|^2$ state transition expressions. Within each of the expression, there are

$$|LP|.|FP|.|A| \tag{5}$$

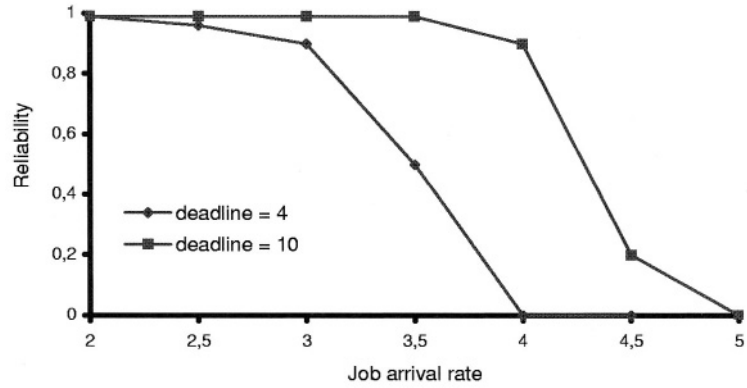


Fig. 4. System Reliability vs. Job Arrival Rate

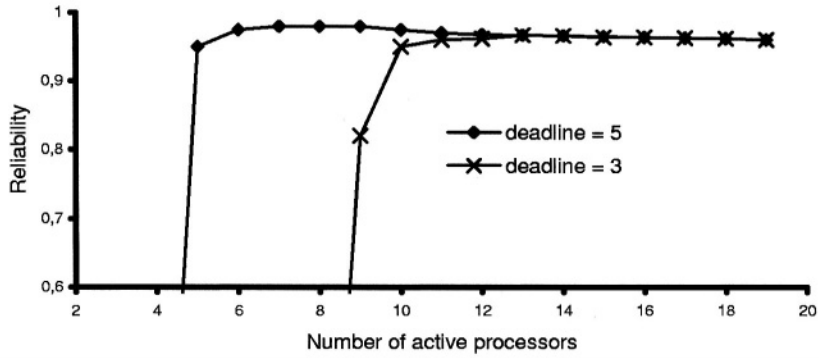


Fig. 5. The System Reliability

items to sum up. Hence, the complexity of our model is $|MDP'| = |S|^2.|LP|.|FP|.|A|$ Therefore, the MDP' reduces the complexity of the MDP by

$$\frac{|MDP|}{|MDP'|} = \frac{|Z|^2.|A|}{|S|^2.|LP|.|FP|.|A|} = |LP|.|FP| \tag{6}$$

The MDP' model reduces the complexity of the decision problem by $|LP|.|FP|$ times. For a system with ten different workload patterns and ten different possible fault patterns in a con-figuration state, our model will reduce the complexity of the decision problem by $|LP|.|FP| = 100$ times.

4 Experimental Results

All the experiments were carried out with simulator. Tasks are scheduled and run in the environment specified by user. To support all the experi-ments we have run PVM process for each node.

Figure 4 shows how the reliability depends on job arrival with different dead-lines (4 and 10 time units). The system reliability decreases when job arrival rate increases, and we can see two different curves with different deadlines of the tasks. With larger deadline, the system reliability is better even with high job arrival rates. In Figure 5, we calculate the reliability of the system when the number of nodes varies from 4 to 18 nodes with a single spare node. The job arrival rate is 2 per a unit of time and the dead-lines Δ for all tasks are 3 and 5. We can see that for different deadlines job arrival and the number of proces-sors the reliability of the system is changed. When the job arrival increases the reliability decreases rapidly and when the number of processors increases the reliability decreases.

5 Conclusions

We have shown how to enhance reliability of the system by using Markovian Decision Processes with choosing normal state as the first state. When we start with normal state, the improvement is directly proportional to workload patterns and fault patterns.

References

1. Stankovic J. A: Distributed Real-Time Computing: The Next Generation, Invited paper for Journal of the Society of Instrument and Control Engineers of Japan. (1992)
2. Hermann Kopetz: Real-Time Systems – Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers (1997)
3. C. L. Liu and J. W. Layland: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM. **20(1)** (1973) 46–61
4. H. Chetto and M. Chetto: Some Results of the Earliest Deadline Scheduling Algorithm. IEEE Transactions on Software Engineering. **15(10)**, (October 1989).
5. Hluchy L., Dobrucky M. Viet T.D. and Astalos J.: The Mapping, Scheduling and Load Balancing Tools of GRADE. Parallel Program Development For Cluster Computing, NOVA Science Publishers, Inc. USA. ISBN 1-56072-865-5. Advances in Computation: Theory and Practice Volume 5 (2001)
6. Hluchy L., Nguyen G., Halada L., Tran V.: Parallel Flood Modeling. In: Parallel Computing, Parco (2001), London: Imperial College Press.
7. Hluchy L., Dobrucky M., Astalos J.: Hybrid Approach to Task Allocation in Distributed Systems. In Lecture Notes in Computer Science 1277, Springer, 1997. pp 210-216.

Online Scheduling of Multiprocessor Jobs with Idle Regulation*

Andrei Tchernykh[1]** and Denis Trystram[2]

[1] CICESE, Ensenada, Baja California, México
chernykh@cicese.mx
[2] ID-IMAG, 38330 Montbonnot St. Martin, France
trystram@imag.fr

Abstract. In this paper, we focus on on-line scheduling of multiprocessor jobs with emphasis on the regulation of idle periods in the frame of general list policies. We consider a new family of scheduling strategies based on two phases which successively combine sequential and parallel executions of the jobs. These strategies are part of a more generic scheme introduced in [6]. The main result is to demonstrate that it is possible to estimate the amount of resources that should remain idle for a better regulation of the load and to obtain approximation bounds.

1 Introduction

Different types of parallel applications, depending on their dynamic and static characteristics, introduce different constraints on the scheduling policies. Unfortunately, usually there is no sufficient knowledge about applications in order to schedule them efficiently based on models of the classical theory. Moreover, real application characteristics are too complex to be used as formal scheduling constraints. If the application constraints are unpredictable or, even, undefined, the resource allocation for computing jobs could be done by a scheduler, based on available information as parameters of the processors or the operating system. However, they are very hard to define formally too. For the case of sequential jobs, scheduling problems have been studied for decades [1,3], much less is known about the efficient on-line parallel jobs' scheduling solutions [23,4].

1.1 Applications

Parallelism inside applications adds a new dimension to the scheduling problem. The manner in which jobs partitioning can be done (when a job requires more than one processor at a time) depends on their divisible property. According to the classification of Feitelson et al. [5,11], several types of job processor requirements in two general classes named rigid and flexible are distinguished. Rigid

* This work was supported in part by CONACYT under grant #32989-A

** Part of the work was done during a research stay of Andrei Tchernykh at ID-IMAG, France

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 131–144, 2004.
© Springer-Verlag Berlin Heidelberg 2004

jobs include multiprocessor jobs with a fixed number of processors requested for parallel execution that is not changed until the job is completed. They can also have moldable property, and, hence, they can be run on any number of processors, but once the number of processors has been allotted to the job it remains the same throughout its execution [10]. Malleable parallel jobs have flexible property so that they can be divided into any number of segments of any desired fractional size. The number of processors allotted to the job is not imposed in advance and depends on the number of processors available at the moment of allotment, on change in requirements or load. At any moment, when more processors are available, the *same* job can be preempted, redistributed and resumed on a different number of processors. Evolving parallel jobs require different numbers of processors during its execution [9].

1.2 Framework for Resource Management

Big variety of job scheduling policies for parallel computers that have been studied in the literature makes it clear that practical scheduling solutions are different and require a variety of models and techniques. Managing the performance of systems through direct administrator or user adjustment of scheduling policies is impractical. One possible way to handle realistic scheduling problems is to use a framework that can support different solutions and be adapted to them on-line.

In [6], a generic adaptive scheduling strategy, called *(a,b,c)-Scheme,* has been introduced. This framework appears to be a good starting point for understanding the unification of different dynamic scheduling strategies. Preliminary results show that it is possible to design good approximation algorithms for scheduling parallel jobs. The scheme unifies the scheduling of usual sequential jobs and jobs that may require more than one processor for their execution, and automatically adapts to the right granularity of the applications by using a set of three parameters (a, b and c referring respectively to system, application and scheduling strategies characteristics).

1.3 Penalty Factor

To propose a trade-off between the complexity of parallel systems, applications, and desired simplicity of their models for a theoretical study, we consider the model of parallel job based on a *penalty factor.* Such a model has been used in various actual codes [2,7]. The idea is to add an overhead into the parallel execution time, which includes the time lost for communication, synchronization, preemption or any extra factors that come from the management of the parallel execution. The penalty factor implicitly takes into account some constraints, when they are unknown, very hard to be defined formally, or hides real application characteristics or computer parameters, though known, but too complex to be used as formal scheduling constraints. The penalty of a single job execution or workload processing can be estimated based on an empirical analysis, benchmarking, counting, profiling, performance evaluation through modelling or measurement, or based on the information provided by a user.

Several qualitative shapes of the penalty as a function of the number of processors allotted to the job have been discussed in the literature [2]. We usually distinguish three broad classes of job penalty factor: namely, constant, linear, and convex. They are the most common classes while parallelizing actual numerical applications. The constant shape of the penalty functions corresponds to the system where applications achieve near linear increasing speed-up with increasing number of processors. The linear penalty functions correspond to the logarithmic shape of a job speed-up function that exhibits speed-up that mostly rises monotonically up to the certain threshold and slowdown beyond a certain number of allocated processors. The convex penalty functions correspond to the small start-up overhead and the addition of extra processors up to the certain threshold costs minimum. After the threshold with greater number of processors, the cost of management of parallelism is increased causing degradation of the speed-up. It correlates with the concave processing speed-up function.

We assume that the penalty factor depends non-decreasingly on the number of processors: $\mu_k^i \leq \mu_{k+1}^i$ for any $1 \leq k \leq m$. For a job T_i allocated to k processors, $p_k^i = \frac{\mu_k^i p_1^i}{k}$, where p_k^i is the processing time of job T_i executed on k processors. We consider that p_k^i depends non-increasingly on k, at least for a reasonable number of processors, where $k_{\min} \leq k \leq k_{\max}$. Calculating the speed-up S_k^i of the execution of job T_i on k processors as p_1^i/p_k^i, the penalty μ_k^i is considered as the ratio of the ideal speed-up over achieved one k/S_k^i.

In this paper, we exclude from analysis the applications with super linear speed-up that may be obtained for some big applications exceeding cache or available memory, and applications with speed-up value less than one (that contain not enough parallelism or large parallelization overhead for being executed by several processors).

Let $\mu_{\max}$ be $\max_{1,n}\{\mu_k^i\}$, $\mu_{\min}$ be $\min_{1,n}\{\mu_k^i\}$, $p_i = p_1^i$, $p_{\max}$ be $\max_{1,n}\{p_i\}$, $p_{\max}^{\text{par}}$ be $\max_{1,n}\left\{\frac{\mu_k^i p_i}{k_i}\right\}$, $k_{\min}$ be $\min_{1,n}\{k_i\}$, and $k_{\max}$ be $\max_{1,n}\{k_i\}$.

1.4 Idle Regulation

The natural principle of the on-line list scheduling is a greedy method such that a processor cannot be idle if there exist some jobs ready to be executed. Scheduling strategies used a dynamic idle regulation approach in order to improve system behavior has received some interest recently. Job partitioning with keeping some processors idle was suggested in [20]. An analysis has shown its effectiveness for non-scalable workloads, possibly with high variance in arrival rate and execution. In [14], the authors presented the design and implementation of a scheduling policy that prevents the degradation in performance appeared from scheduling idle processors too early. It keeps some processors idle during some periods of time before scheduling them. Exploiting unused time slots in list scheduling was also considered in [25].

We put emphasize in this work on how to parametrize easily a simple policy based on list scheduling. We consider the general framework introduced in [6],

the *(a,b,c)-Scheme,* where the strategies are described by a set of parameters. The value of the first parameter a represents the number of processors allowed to be idle in the system. This parameter can be selected in accordance to the current system load, performance of the system, provided by the scheduler or administrator. It depends on the penalty factor of the system and applications, and can be adapted to the change of quality of the workload on-line, based on runtime measuring of workload characteristics. The variation of parameter a gives the possibility to find a trade-off between avoiding idle processors by starting the parallelization sooner when more jobs are parallelized (with a larger overhead) and delaying their parallelization (causing processors to be left idle) until a smaller number of jobs is available.

Many models of the workload assumed that information about its performance (speed-up, inefficiency) is available a priori. However, it is clear that the choice of appropriate policies must be guided at least in part by the behaviour of the actual parallel workload that is diverse and can be changed at runtime. *(a,b,c)-Scheme* uses the idea of the runtime tuning of the system parameter a in order to optimize the system behaviour, measuring of workload characteristics (post-mortem analysis of jobs characteristics during a certain time interval) to estimate the penalty factors. The runtime measuring of workload characteristics to improve job scheduling has received considerable attention in recent years. In [19] on-line measuring of job efficiency and speedup in making production schedulers decisions were discussed. In [24], the use of runtime measurements to improve job scheduling on a parallel machine with emphasis on gang scheduling based strategies was investigated.

1.5 Processor Allocation Regulation

The idea of allocation of fewer and fewer processors to each job under heavy load, thus increasing efficiency, was proposed in [12]. The authors considered a processor allocation scheme based on the number of processors, for which the ratio of execution time to efficiency is minimized. The maximizing of application speedup through runtime selection of an appropriate number of processors on which to run was discussed in [18]. The use of a runtime system that dynamically measures job efficiencies at different allocations and automatically adjusts a job's processor allocation to maximize its speedup was proposed. In [17], the dynamic scheduler that uses runtime idleness information to dynamically adjust processor allocations to improve shared memory system utilization was proposed. Self-adaptive scheduling strategy in a master-worker paradigm that dynamically adjusts the number of processors based on performance measures gathered during its execution was considered in [13]. In [8], it is shown that of several strategies with equivalent turnaround times, the strategy that reduces allocations when load is high yields the lowest slowdowns. A general virtualization technique that simulates a virtual machine of p processors on $p' < p$ processors and allows the execution of a parallel job that requests p processors while only p' processors are available can be found in [22]. The technique yields

good results for different network topologies. The two-phases strategy for the processor allocation regulation was introduced in [21].

The objective of this paper is to study the impact of idle regulation and processor allocation regulation policies solving on-line rigid multiprocessor jobs scheduling problems in the framework of the two-phases strategy.

The rest of paper is organized as follows: in section 2 we present the notation of scheduling problem and we describe the two-phases strategy for on-line multiprocessor job scheduling. The analysis of the performance guarantee for some cases is presented in section 3. Finally, further research directions are discussed.

2 Preliminaries

In this work, we focus on on-line batch style of scheduling. That means that a set of available ready jobs will be executed up to the completion of the last one. All jobs which arrive in the system during this time will be processed in the next batch.

2.1 Scheduling Problem

We consider a set of independent parallel multiprocessor jobs with the objective to minimize the total execution time (makespan) in the frame of the two-phases list scheduling strategy described in the next section. We restrict the analysis to the scheduling systems where all the jobs are given at time 0 and are processed into the same batch. A relation between this scheme and the scheme where jobs released over time, either at their release time, according to the precedence constraints, or released by different users is known and studied for different scheduling strategies for general or restricted cases [26].

The following problem is studied. Given a parallel machine with m identical processors and a set of n independent multiprocessor jobs $T = \{T_1, \ldots, T_n\}$. The main on-line feature is the fact that the processing times $p_1, \ldots, p_n$ are not known until their completion. The number of processors k_i needed for the execution of job T_i is fixed and known as soon as it becomes available. We assume that the job irrespective of k_i can be scheduled either on the single processor or requested k_i processors. The job can be preempted, assigned to the required number of processors, then it cannot be preempted and/or continue to run on a different set and/or different number of processors. A parallel job T_i is characterized by a triple $T_i = (p_i, k_i, \mu_k^i)$: namely, its execution time on a single processor (the total work done by the job), the number of processors k_i, and the penalty factor of the execution on these processors.

All the strategies will be analysed according to their competitive ratio. Let C_{opt}^{par} denote the makespan of an optimal schedule, where the parallelization of the jobs is allowed, and C_{opt}^{seq} denote the makespan of the optimal schedule, when the parallelization of the jobs is not allowed. For a strategy A, let C_A be the makespan of corresponding schedule. Let $\rho_A^{\text{par}} = \frac{C_A}{C_{opt}^{\text{par}}}$ and $\rho_A^{\text{seq}} = \frac{C_A}{C_{opt}^{\text{seq}}}$ denote

the parallel and sequential competitive ratios of strategy A. The sequential competitive ratio will highlight the possible gain to allow the multiprocessing of jobs compared to the classical approach of general list scheduling where jobs are purely sequential.

2.2 The Two-Phases Algorithm

In this work, we are interested in studying the influence of idle regulation on a general strategy based on list scheduling. This strategy have been introduced in [21] and consider two successive phases.

When the number of jobs is large, it allocates processors without idle times and communications. When enough jobs have been completed, we switch to a second phase with multiprocessor execution. In the following analysis, we assume that in the first phase each processor has one job, each job is assigned to one processor, and the efficiency is 1. Inefficiency appears when less than m jobs remain. Then, when the number of idle processors become larger than a, where $0 \leq a \leq m$, all remaining jobs are preempted and another strategy is applied to avoid too many idle processors. Figure 1 illustrates this algorithm.

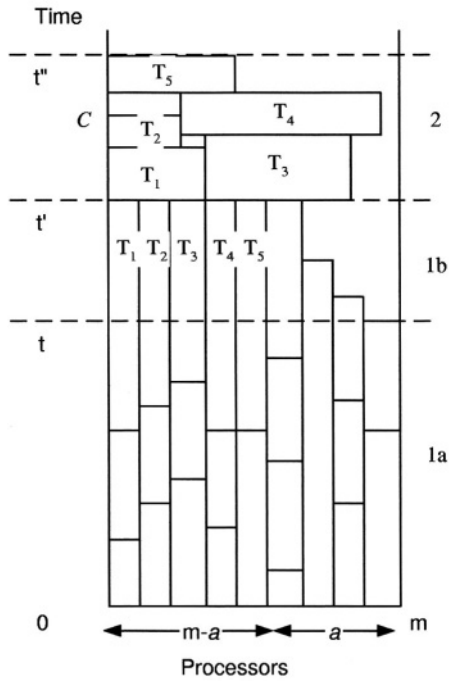


Fig. 1. The two-phases strategy for $a = 4$ and $m = 9$

The successive phases are shown: phase 1a (time interval $[0, t]$), when all the processors are busy; phase 1b (time interval $[t, t']$), when at least $m - a + 1$ processors work (both phases use the well-known Graham's strategy); and phase 2 (time interval $[t', t'']$), when a or more processors become idle, and, hence, turn to a second strategy with multiprocessor jobs.

Parameter a determines the balance between the number of jobs processed by the first and the second strategies. The execution of jobs sequentially in the first phase under heavy load is a best strategy as there is no extra overhead. We assume that all parallel jobs can be executed sequentially irrespective of their type and the number of processors needed for the execution is imposed or not. Even rigid jobs that carefully tuned-up to achieve a large degree of latency hiding and cannot cope with the reducing or increasing the number of allotted processors are executed efficiently on one processor because of the locality of communications. It is suitable assumption for many distributed and shared memory applications, in which parallel processes can be executed sequentially (in particular, message passing processes executed by one processor), but excludes from our consideration some class of parallel jobs, for example, synchronous shared memory applications, where parallelism cannot be transformed to the time sequence.

On the second phase, when the load is reduced and the number of idle processors becomes equal to or more than a, the system changes the strategy to avoid too many idle times.

Such an idle regulation gives a possibility to find a trade-off between starting parallelization sooner, when a is smaller (hence more jobs are parallelized causing bigger parallelization overhead), and delaying their parallelization, when a is bigger (hence causing more processors to be left idle), until a smaller number of jobs is available. This scheme balances the needs of the user (job) with those the computer system. The number of processors for the job execution selected by users is typically suitable for light load conditions, but it leads to unacceptably low processor system efficiency at heavy load [11]. Since users cannot practically know the load on the system, one possible way to optimize system behavior is to support different solutions and be adapted to them on-line.

3 Analysis

We provide now an analysis of the two-phases algorithm with idle regulation for the case $0 \leq a \leq m$. We study the performance in regard to the number of processors allotted to a job for its execution and its penalty factor. Before deriving the general bound, we study the restricted case of $a = 0$ (scheduling rigid jobs by a list algorithm). The case $a = 1$ for a specific allocation of processors in gang (to m processors) has been studied in [21]. The case $a = m$ corresponds simply to list scheduling for sequential jobs (only first phase).

3.1 $a = 0$

We should first notice that multiprocessor rigid jobs strategy corresponds to a 2 dimensional packing problem. Only the second phase of the general algorithm is considered for this case.

Lemma 1. *The sequential and parallel performance guarantees of any list algorithm for scheduling of independent multiprocessor jobs are [15,16]*

$$\rho^{\text{seq}} \leq \frac{1}{m - k_{\max} + 1}\left(\mu_{\max} m + \frac{\mu_{\min}}{k_{\min}}(m - k_{\max})\right) \quad \textit{and}$$
$$\rho^{\text{par}} \leq \frac{2m - k_{\max}}{m - k_{\max} + 1} \ .$$

Proof. Let us consider that in the schedule, there are two kind of time slots which can be combined conceptually into two successive intervals, namely C_1 and C_2. Both intervals correspond respectively to the time slots when at most $k_{max} - 1$ processors are idle, and strictly more than $k_{\max} - 1$ processors are idle.

Let $W_{\text{seq}} = \sum_{i=1}^{n} p_i$ be the total work of all jobs, $W_{\text{par}} = \sum_{i=1}^{n} \mu_k^i p_i \leq \mu_{\max} W_{\text{seq}}$ be the work performed in the parallel execution, and W_{par}^1 be a the work performed in C_1.

Noticing that the number of idles in the interval C_2 is limited by $(m - k_{\max})C_2$, and $C_2 \leq p_{\max}^{\text{par}}$, we obtain an upper bound of the total completion time

$$C = C_1 + C_2 \leq \frac{W_{\text{par}}^1}{m - k_{\max} + 1} + \frac{(W_{\text{par}} - W_{\text{par}}^1) + (m - k_{\max})p_{\max}^{\text{par}}}{m}$$
$$C \leq \frac{W_{\text{par}} + (m - k_{\max})p_{\max}^{\text{par}}}{m - k_{\max} + 1}$$

Noticing also that $\frac{W_{\text{seq}}}{m}$ and $p_{\max}$ are lower bounds of C_{opt}^{seq}, and $\frac{W_{\text{par}}}{m}$ and $p_{\max}^{\text{par}}$ are lower bounds for C_{opt}^{par}, it follows that

$$\rho^{\text{seq}} \leq \frac{\mu_{\max} m}{m - k_{\max} + 1} + \frac{\mu_{\min}(m - k_{\max})}{k_{\min}(m - k_{\max} + 1)}$$
$$\leq \frac{1}{m - k_{\max} + 1}\left(\mu_{\max} m + \frac{\mu_{\min}}{k_{\min}}(m - k_{\max})\right) \quad \text{and}$$
$$\rho^{\text{par}} \leq \frac{2m - k_{\max}}{m - k_{\max} + 1} \qquad \square$$

Remark 1. Let us check briefly what happens for sequential jobs: we obtain the same bound as Graham (no penalty, $\mu_{\min} = \mu_{\max} = 1$ and $k_{\min} = k_{\max} = 1$) and both competitive ratio coincide.

3.2 Idle Regulation with $a > 0$

Let us consider now that the strategy switches from the *Graham's* strategy to multiprocessor job scheduling when a processors become idle (Figure 1).

Theorem 1 (Sequential competitivity). *Given a set of n independent multiprocessor jobs with variation of the penalty factors from $\mu_{\min}$ to $\mu_{\max}$ when allotted on fixed number of processors from $k_{\min}$ to $k_{\max}$, the sequential performance guarantee can be estimated by $\rho^{\text{seq}} = \max\{\rho^1, \rho^2\}$, with*

$$\rho^1 \leq 1 + \frac{a-1}{m} \quad \textit{and}$$
$$\rho^2 \leq \frac{a}{m} + \frac{1}{m - k_{\max} + 1}\left(\mu_{\max}(m-a) + \frac{\mu_{\min}}{k_{\min}}(m - k_{\max})\right) .$$

Proof. Let denote by W_{seq} the total work, and by W^1_{seq} the work executed until t. Each job not finished until t' is executed necessarily in phase 2. Then, assuming that part of these jobs have not been processed in phase 1a or 1b the remaining work is less than $W^2_{\text{seq}} \leq \sum_{i=1}^{h}(p_i - \delta) \leq h(p_{\max} - \delta)$, where $h \leq m - a$, and $\delta = t' - t$. Let $p^{\text{par}}_{\text{rem}}$ be $\max_{1,n}\left\{\frac{\mu^i_k(p_i - \delta)}{k_i}\right\}$. Following Lemma 1 for the completion time of the phase 2, the total completion time is

$$C = \frac{W^1_{\text{seq}}}{m} + \delta + \frac{\mu_{\max} W^2_{\text{seq}} + (m - k_{\max})p^{\text{par}}_{\text{rem}}}{m - k_{\max} + 1} ,$$

$W_{\text{seq}} \geq W^1_{\text{seq}} + (m - a + 1)\delta + W^2_{\text{seq}}$. Hence

$$C \leq \frac{W_{\text{seq}}}{m} - \frac{m - a + 1}{m}\delta + \delta + W^2_{\text{seq}}\left(\frac{\mu_{\max}}{m - k_{\max} + 1} - \frac{1}{m}\right) + \frac{(m - k_{\max})p^{\text{par}}_{\text{rem}}}{m - k_{\max} + 1} .$$

Considering h jobs executed after t', and $p^{\text{par}}_{\text{rem}} \leq \frac{\mu_{\min}(p_{\max} - \delta)}{k_{\min}}$ it follows that

$$C \leq \frac{W_{\text{seq}}}{m} + \left(\frac{a-1}{m} - \left(\frac{h\mu_{\max}}{m - k_{\max} + 1} + \frac{(m - k_{\max})\mu_{\min}}{k_{\min}(m - k_{\max} + 1)} - \frac{h}{m}\right)\right)\delta$$
$$+ \left(\frac{h\mu_{\max}}{m - k_{\max} + 1} + \frac{(m - k_{\max})\mu_{\min}}{k_{\min}(m - k_{\max} + 1)} - \frac{h}{m}\right)p_{\max} .$$

Let denote $B(a) = \frac{a-1}{m}$, $A(h) = \frac{h\mu_{\max}}{m-d} + \frac{(m - k_{\max})\mu_{\min}}{k_{\min}(m-d)} - \frac{h}{m}$, and $J(a,h) = \big(B(a) - A(h)\big)\delta + A(h)p_{\max}$. Hence, $C \leq \frac{W_{\text{seq}}}{m} + J(a,h)$.

We consider three different possibilities for values of $A(h)$ and $B(a)$.
If $A(h) = B(a)$ then $J(a,h) \leq A(h)p_{\max}$,

$$C \leq \frac{W_{\text{seq}}}{m} + \left(\frac{h\mu_{\max}}{m-k_{\max}+1} + \frac{(m-k_{\max})\mu_{\min}}{k_{\min}(m-k_{\max}+1)} - \frac{h}{m}\right)p_{\max} \quad \text{and}$$

$$\rho^{\text{seq}} \leq \frac{a}{m} + \frac{1}{m-k_{\max}+1}\left(\mu_{\max}(m-a) + \frac{\mu_{\min}}{k_{\min}}(m-k_{\max})\right) \ .$$

If $A(h) < B(a)$ then $J(a,h) \leq \big(B(a) - A(h)\big)p_{\max} + A(h)p_{\max} \leq B(a)p_{\max}$,

$$C \leq \frac{W_{\text{seq}}}{m} + \frac{a-1}{m}p_{\max} \quad \text{and} \quad \rho^{\text{seq}} \leq 1 + \frac{a-1}{m} \ .$$

If $A(h) > B(a)$ then $J(a,h) \leq A(h)p_{\max}$ and

$$\rho^{\text{seq}} \leq \frac{a}{m} + \frac{1}{m-k_{\max}+1}\left(\mu_{\max}(m-a) + \frac{\mu_{\min}}{k_{\min}}(m-k_{\max})\right) \ . \qquad \square$$

Remark 2. For $a = 0$, $\rho^{\text{seq}} \leq \dfrac{1}{m-k_{\max}+1}\left(\mu_{\max}m + \dfrac{\mu_{\min}}{k_{\min}}(m-k_{\max})\right)$ that corresponds to a sequential performance guarantee of any list algorithm for scheduling multiprocessor jobs (Lemma 1). For $a = m$, and $k_{\max} = k_{\min} = 1$, $\rho^{\text{seq}} \leq 2 - \dfrac{1}{m}$, only the *Graham* strategy (phase 1) is applied.

Figure 2 shows the sequential competitive ratio ρ^{seq} when varying the number of processors with fixed $a, k_{\max}, k_{\min}$, linear μ_k (logarithmic shape of a job speed-up function of k), and constant μ_k (linear $m/2$ shape of a job speed-up function of k). We assume here that the number of unfinished jobs on phase 2 is $m - a$. Figure 3 presents the performance when varying the parameter a with fixed number of processors, $k_{\max}, k_{\min}$, linear and constant μ_k. It shows that tuning of the idle times by the parameter a allows to get the minimum of the worst case error bound.

Theorem 2 (Parallel competitivity). *Given a set of n independent multiprocessor jobs, the parallel performance guarantee can be estimated by:*

$$\rho^{\text{par}} \leq \frac{(2m-k_{\max})}{m-k_{\max}+1} - a\left(\frac{1}{m-k_{\max}+1} - \frac{1}{m}\right) \ .$$

Proof. As shown in Theorem 1, the completion time of the schedule is

$$C \leq \frac{W^1_{\text{seq}}}{m} + \delta + \frac{W^2_{\text{par}} + (m-k_{\max})p^{\text{par}}_{\text{rem}}}{m-k_{\max}+1} \ .$$

Notice that

$$\begin{aligned} W_{\text{par}} &\geq \mu_{\min}W^1_{\text{seq}} + (m-a+1)\delta\mu_{\min} + W^2_{\text{par}} \\ &\geq W^1_{\text{seq}} + (m-a+1)\delta + W^2_{\text{par}} \end{aligned}$$

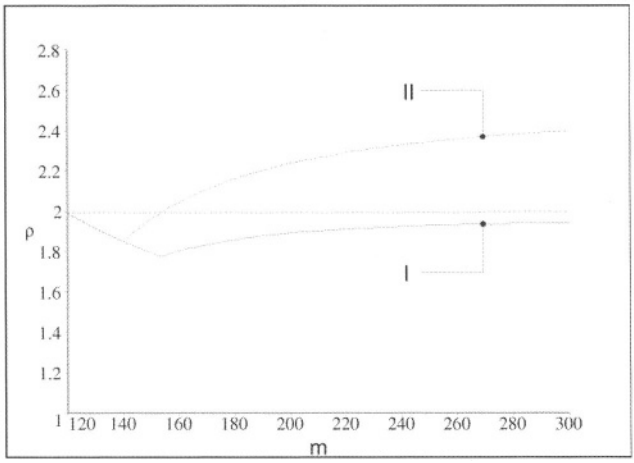


Fig. 2. ρ^{seq} when varying the number of processors; with $a = 120$, $k_{\max} = 60$, $k_{\min} = k_{\max}/30$. (I) linear μ_k (logarithmic shape of a job speed-up function), and (II) constant μ_k (linear $m/2$ shape of a job speed-up function of k).

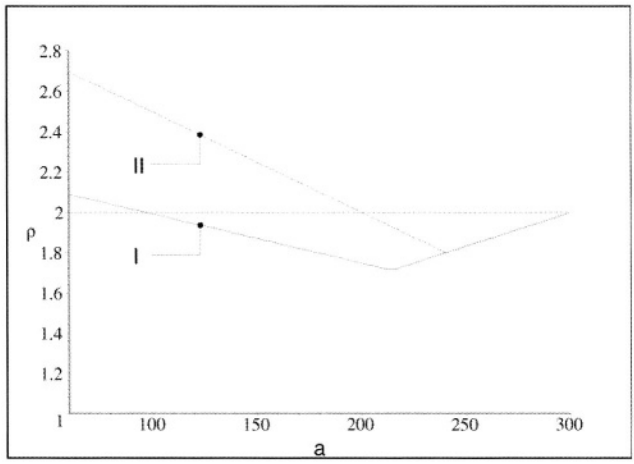


Fig. 3. ρ^{seq} when varying a, with $m = 300$; (I) linear μ_k (logarithmic shape of a job speed-up function) and (II) constant μ_k (linear $m/2$ shape of a job speed-up function of k), $k_{\max} = 60$, $k_{\min} = k_{\max}/30$.

It follows $C \leq \dfrac{W_{\text{par}}}{m} + \dfrac{a-1}{m}\delta + W_{\text{par}}^2 \left(\dfrac{1}{m - k_{\max} + 1} - \dfrac{1}{m} \right) + \dfrac{(m - k_{\max}) p_{\text{rem}}^{\text{par}}}{m - k_{\max} + 1}.$

Considering $W_{\text{par}}^2 \leq h p_{\text{rem}}^{\text{par}}$, where $p_{\text{rem}}^{\text{par}} = \dfrac{\mu'(p' - \delta)}{k'}$ be $\max\limits_{1,n} \left\{ \dfrac{\mu_k^i (p_i - \delta)}{k_i} \right\}$, it follows that

$$C \leq \frac{W_{\text{par}}}{m} + \left(\frac{a-1}{m} - \left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{m-k_{\max}}{m-k_{\max}+1} \right) \frac{\mu'}{k'} \right) \delta + \left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{m-k_{\max}}{m-k_{\max}+1} \right) \frac{\mu' p'}{k'} \ .$$

Let denote $B(a) = \frac{a-1}{m}$, $A(h) = \left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{m-k_{\max}}{m-k_{\max}+1} \right) \frac{\mu'}{k'}$, and $J(a,h) = (B(a) - A(h))\,\delta + A(h)p'$. Hence $C \leq \frac{W_{\text{par}}}{m} + J(a,h)$.

Noticing that $A(h) \geq B(a)$, it follows $J(a,h) \leq A(h)p'$.

$$C \leq \frac{W_{\text{par}}}{m} + \left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{m-k_{\max}}{m-k_{\max}+1} \right) \frac{\mu'}{k'} p' \ .$$

Lower bounds of an optimal schedule are

$$C^{\text{par}}_{opt} \geq \frac{W_{\text{par}}}{m} \quad \text{and} \quad C^{\text{par}}_{opt} \geq p^{\text{par}}_{\text{rem}} + \frac{\mu'}{k'}\delta = \frac{\mu'}{k'} p' \ .$$

Thus $$\rho^{\text{par}} \leq \frac{2m - k_{\max}}{m - k_{\max} + 1} - a \left(\frac{1}{m-k_{\max}+1} - \frac{1}{m} \right) \ . \qquad \square$$

Remark 3. For $a = 0$, $\rho^{\text{par}} \leq \frac{2m - k_{\max}}{m - k_{\max} + 1}$ that corresponds to a performance guarantee of any list algorithm for scheduling multiprocessor jobs. For $k_{\max} = 1$, $\rho^{\text{par}} \leq 2 - \frac{1}{m}$, only the *Graham* strategy is applied.

4 Conclusion and Future Work

In this paper, we focused on on-line scheduling of multiprocessor rigid jobs with emphasis on the idle regulation. We considered non-clairvoyant scheduler that have no information about the jobs other than the number of unfinished jobs in the system and their processor requirements. We proposed to solve on-line scheduling problems using batch scheduling under a generic framework of two-phases of list scheduling (for sequential jobs and rigid multiprocessor jobs).

Scheduling strategies used a dynamic idle regulation approach in order to improve system behavior has receive attention recently. In this paper, we showed that tuning of the parameter a that regulates idle times gives a possibility to find a tradeoff between avoiding idle processors by starting parallelization sooner when more tasks are parallelized and delaying their parallelization until a smaller number of jobs is available. A tradeoff is a minimum of the performance guarantee for a parallel computer system with fixed number of processors and known

overhead. The minimum depends on the penalty factor of the system and applications, and can be adapted to the change of quality of the workload on-line, based on runtime measuring of workload characteristics.

Several important and interesting questions still remain unanswered. Firstly, analysis and simulations of the strategy with more variations of other features like moldable or malleable jobs, and different scheduling strategies (FCFS, largest job first, etc.). Secondly, the analysis of the system to be able to support scheduling dependent jobs, or when new jobs can arrive in any moment that is more practical considering that workloads vary in a day cycle. There is considerable scope to obtain results by using the scheme to the scheduling of malleable jobs that can be scheduled on any available number of processors. The issues related to the implementation of the strategy on real parallel or distributed computing systems are also important.

References

1. J. Błażewicz, M. Drozdowski. K. Ecker, "Management of resources in parallel systems", in the Handbook on Parallel and Distributed Processing, Springer Verlag, 2000, pp.263-341
2. E. Blayo, L. Debreu, G. Mounie, D. Trystram. Dynamic Load Balansing for Adaptive Mesh Ocean Circulation Model. Engineering Simulation. V22, 2, pp. 8-23, 2000
3. J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz, "Scheduling Computer and Manufacturing Processes", Springer Verlag, Berlin, New York, 2001.
4. V. Bharadwaj, D. Ghose, V. Mani, T. Robertazzi. Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamos, USA, 1996
5. S. Chapin, W. Cirne, D. Feitelson, J. Patton Jones, S.T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers". In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999, LNCS. vol. 1659, pp. 66-89.
6. A. Thernykh, C. Rapine, D. Trystram. Adaptive scheme for on-line scheduling of parallel jobs. Technical report at IMAG, 2003.
7. T. Decker, W. Krandick, Parallel real root isolation using the Descartes method, Proceedings of the 6th High Performance Conference (HiPC99), 261-268, LNCS, vol. 1745, Springer-Verlag, 1999
8. A. Downey. A parallel workload model and its implications for processor allocation. In Proc. the 6th International Symposium of High Performance Distributed Computing, pp 112–123, 1997.
9. D. Feitelson and L. Rudolph. "Toward convergence in job schedulers for parallel supercomputers" In Job Scheduling Strategies for Parallel Processing, D. Feitelson and L. Rudolph (Eds.), pp. 1-26, Springer-Verlag, 1996. LNCS 1162.
10. D. Feitelson and L. Rudolph. Metrics and Benchmarking for Parallel Job Scheduling. D.G. Feitelson, L. Rudolph (Eds.): JSSPP, IPPS/SPDP'98 Workshop, Orlando, Florida, USA, March 1998. Proceedings, LNCS 1459, p. 1-24.

11. D. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling". In Job Scheduling Strategies for Parallel Processing D. G. Feitelson and L. Rudolph (Eds.), pp. 1-34, Springer-Verlag, 1997. LNCS, Vol. 1291.
12. D. Ghosal, G. Serazii, and S. Tripathi. "The processor working set and its use in scheduling multiprocessor system", IEEE Trans. Soft. Eng. 17(5), pp. 443-453, May 1991
13. E. Heymann, M. Senar, E. Luque, and M. Livny. Self-Adjusting Scheduling of Master-Worker Applications on Distributed Clusters. R. Sakellariou et al. (Eds.): Euro-Par 2001, Manchester, UK August 28-31, 2001, Proceedings, LNCS 2150, p. 742-751.
14. S. Iyer, P. Druschel, Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O, Symposium on Operating Systems Principles, 117-130, 2001
15. E. Lloyd Concurrent task systems, Operational Research 29/1, 1981, pp. 189-201
16. R. Lepere, G. Mounie, D. Trystram. An Approximation algorithm for scheduling Trees of Malleable Tasks. EJOR, 2002
17. C. McCann, R. Vaswani, and J. Zahorjan. A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors. ACM Transactions on Computer System, 11(2): 146-178, May 1993
18. T. Nguyen, R. Vaswani, and J. Zahorjan. Parallel Application Characterization for Multiprocessor Scheduling Policy Design. In JSSPP, D.G. Feitelson and L. Rudolph (eds.), Volume 1162 of LNCS, Springer-Verlag, 1996.
19. T. Nguyen, R. Vaswani, and J. Zahorjan. Using Runtime Measured Workload Characteristics in Parallel Processor Scheduling. In JSSPP, D. G. Feitelson and L. Rudolph (editors), vol. 1162, LNCS. Springer-Verlag, 1996.
20. E. Rosti, E. Smirni, G. Serazzi, L. Dowdy. "Analysis of non-work-conserving processor partitioning policies". In Job Scheduling Strategies for Parallel Processing, D. Feitelson and L. Rudolph (Eds.), pp. 165-181, Springer-Verlag, 1995. LNCS 949.
21. C. Rapine, I. Scherson, D. Trystram. On-Line Scheduling of Parallelizable Jobs. Proceedings of EUROPAR'98 Conference, Springer-Verlag, LNCS, Vol. 1470, pp. 322-327, 1998
22. J. Sgall, A. Feldmann, M. Kao, and S. Teng. Optimal online scheduling of parallel jobs with dependencies J. of Combinatorial Optimization, 1(4):393-411, 1998
23. J. Sgall. On-line scheduling-A Survey, In Online Algorithms: The State of the Art, eds.A. Fiat and G. J. Woeginger, LNCS 1442, pp 196-231, Springer-Verlag, 1998.
24. F. Silva and I. Scherson. Improving Parallel Job Scheduling Using Runtime Measurements. In D.G. Feitelson and L. Rudolph (Eds.): JSSPP 2000, LNCS 1911, Springer-Verlag, pp. 18–39, 2000
25. O. Sinnen and L. Sousa. Exploiting Unused Time Slots in List Scheduling, Considering Communication Contention, R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 166–170, Springer-Verlag, 2001
26. D. Shmoys, J. Wein, D. Williamson. Scheduling parallel machines on-line. SIAM J. Comput., 24:1313-1331, 1995.

Predicting the Response Time of a New Task on a Beowulf Cluster

Marta Beltrán[1] and Jose L. Bosque[1,2]

[1] ESCET, Universidad Rey Juan Carlos, 28933 Móstoles, Madrid, Spain
[2] Universidad San Pablo CEU, Madrid, Spain
{mbeltran,jbosque}@escet.urjc.es

Abstract. In this paper the problem of making predictions of incoming tasks response times in a cluster node is focused. These predictions have a significant effect in areas such as dynamic load balancing, scalability analysis or parallel systems modelling. This paper presents two new response time prediction models. The first one is a *mixed model* based on two widely used models, *CPU availability* and *Round Robin models.* The second one, called *Response Time Prediction (RTP) model,* is a completely new model based on a detailed study of different kinds of tasks and their CPU time consuming. The predictive power of these models is evaluated by running a large set of tests and the predictions obtained with the RTP model exhibit an error of less than 2 % in all these experiments.

1 Introduction

Beowulf clusters are becoming very popular due to their good price-performance ratio, scalability and flexibility. And predicting the amount of work on the different cluster nodes is a basic problem that arises in a lot of contexts such as cluster modelling, performance analysis or dynamic load balancing ([1], [2]).

In this paper the problem of predicting the response time of a new task in a cluster node is examined. The *CPU assignment* concept (A) is proposed to measure and compare the load of different cluster nodes, in other words, the response time of a new task in each of these nodes. The assignment is defined as the percentage of CPU time that would be available to a newly created task. CPU availability has been successfully used before, for example, to schedule programs in distributed systems ([3], [4]).

The contributions of this paper are an analytical and experimental study of two well-known response time prediction models, two new static models for CPU assignment computation, a verification of these models through their application in a complete set of experiments and a comparison between all the obtained results. In contrast to the other approaches, the response time predictions with the second proposed model exhibits an error of less than 2%, so the experimental results indicate that this new model is accurate enough for all the mentioned contexts.

The rest of the paper is organized as follows. Section 2 discusses related work on predicting processor workload. Section 3 presents two existing CPU assignment models and proposes two improved new models. Experimental results on comparing the four discussed models are reported in Sect. 4. And finally, Sect. 5 with conclusions and suggestions for future work.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 145–152, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Background

Research that is closely related to this paper falls under two different models: based on predicting the future from past or based on the task queue model.

As an example of the first kind of models, [5] focused on making short and medium term predictions of CPU availability on time-shared Unix systems. On the other hand, [6] presented a method based on neural networks for automatically learning to predict CPU load. And finally [7] and [8] evaluated linear models for load prediction and implemented a system that could predict the running time of a compute-bound task on a host.

Queueing models have been widely used for processors due to their simplicity, so the other kind of models is more extended. In a highly influential paper, Kunz ([9]) showed the influence of workload descriptors on the load balancing performance and concluded that the best workload descriptor is the number of tasks in the run queue. In [10] the CPU queue length is used too as an indication of processors load. And this load index is used again, for example, in [11], [12] and [13]. Finally, in [14], the number of tasks in the run queue is presented as the basis of good load indices but an improvement is proposed by averaging this length over a period of one to four seconds.

3 Response Time Prediction Models

In this paper the *CPU assignment (A)* is defined as the percentage of CPU time that would be available to a new incoming task in a cluster node. This parameter is proposed to analyse prediction models because the response time of a task is directly related to the average CPU assignment it has during its execution. If a process is able to obtain 50% of CPU time slices, it is expected to take twice as long to execute as it would if the CPU was completely unloaded. So a response time prediction for a new task will require a prediction of the CPU assignment for this task during its execution. The assignment concept tries to consider the effects of executing new tasks on CPU availability.

3.1 Previous Models Analysis

The most simple approach is to consider the less loaded node as the node with more free or idle CPU. Analysing this model from the CPU assignment point of view, it considers the CPU assignment as the available CPU at a given instant:

$$A = Available\ CPU \quad . \tag{1}$$

Thus, the predicted assignment for a new task is the percentage of CPU idle time. This model, called *'CPU availability model'* in the rest of this paper, has one important drawback: it does not take into account processor time-sharing between tasks.

In most of computer systems tasks share the processor time using a Round Robin scheduling policy. The other well-known response time prediction model is based on this scheduling and takes the node with less tasks in the run queue as the less loaded cluster node. So the assignment is predicted as the percentage of CPU time that corresponds to a new task with this scheduling policy. If the number of tasks in the run queue is *N,* the assignment prediction with the *'Round Robin model'* is:

$$A = \frac{1}{N+1} \quad . \tag{2}$$

Because the processor time will be shared in a balanced way between *N+1* tasks.

This model is widely used but it only considers CPU-bound tasks. These tasks are computing intensive CPU operations all the time but do not make memory swapping or I/O operations (with disks or network). Indeed, a node executing one CPU-bound task could give less assignment to a new task than a node executing several no CPU-bound tasks. But this model always predicts more assignment for a new task in the first case.

3.2 Proposed Models

To overcome these limitations and take into account all kind of tasks without monitoring other resources like memory or network, a *'mixed model'is* proposed, combining the two previous prediction models.

Let *U* denote the CPU utilization (percentage of CPU time used for the execution of all the tasks in the run queue). The CPU assignment prediction for a new task with this model is:

$$A = \begin{cases} 1/(N+1) \text{ if } U \geq 1/N \\ 1-U \qquad \text{otherwise} \end{cases} . \tag{3}$$

Therefore, if there are only CPU-bound tasks executing on a processor, assignment is obtained applying the *Round Robin model.* But when there are no CPU-bound tasks, they are not taking advantage of all their corresponding CPU time and the CPU assignment for an incoming task will be all the available CPU, of course, greater than $\frac{1}{N+1}$.

Finally, an improvement for this model is proposed, with a more sophisticated explanation about how CPU time is shared between different tasks. This model is called *Response Time Prediction (RTP) model.*

Considering the Round Robin scheduling, CPU-bound tasks always run until the end of their time slices while no CPU-bound tasks sometimes leave the processor without finishing their quantums. The remaining time of these slices is consumed by CPU-bound tasks, always ready to execute CPU intensive operations.

The aim is to take into account this situation, so let t_{CPU} denote the CPU time consumed by a no CPU-bound task when the CPU is completely unloaded and *t* denote the response time for this task in the same environment. The fraction of time spent in CPU intensive operations for this task is:

$$X = \frac{t_{CPU}}{t} \quad . \tag{4}$$

Suppose that there are *n* CPU-bound tasks in the run queue and *m* no CPU-bound tasks. Therefore, $N = n+m$ and the proposed model predicts the following assignment for the *i 'th* task when there is a new incoming task:

$$A = \begin{cases} \frac{X_i}{(n+1)+\sum_{j=1}^{m} X_j} & \text{i=1,...,m} \\ \frac{1-\sum_{j=1}^{m} A_j}{n+1} & \text{i=m+1,...,m+n} \end{cases} . \tag{5}$$

Using the predicted assignments for all no CPU-bound tasks, the assignment for a new task can be computed as all the CPU time that is not consumed by no CPU-bound tasks shared with Round Robin policy between CPU-bound tasks. The new task is supposed to be CPU-bound because it would be the worst case, when the new task would consume all its CPU slices. So, with the new incoming task there will be *n+1* CPU-bound tasks in the run queue.

4 Experimental Results

To determine the accuracy of these four models, a set of experiments has been developed in order to compare measured and predicted response times. The criteria used to evaluate the assignment models is the relative error of their predictions.

All the experiments take place on a 550 MHz Pentium III PC with 128 MB of RAM memory. The operating system installed in this PC is Debian Linux kernel version 2.2.19 and it uses a Round Robin scheduling policy with a 100 millisecond time slice (q = 100 ms).

Different synthetic workloads have been generated due to the lack of appropriate trace data and to their simplicity. The CPU-bound task is named t0 and the no CPU-bound tasks are t1, t2, t3, t4, t5, t6 and t7. These test loads have X=0.66, 0.5, 0.4, 0.3, 0.2, 0.1 and 0.05 respectively.

The first set of experiments to evaluate the models validity and accuracy is performed statically. Thus different sets of test loads are executed simultaneously in our system, beginning and ending at the same time. In each experiment, CPU and response times are measured for all test loads. In order to determine the most accurate model, assignment predictions are made for the task called t0 in each experiment. If A_P is the predicted assignment for this task, the predicted response time is:

$$t_P = \frac{t}{A_P} \quad . \tag{6}$$

where t is the response time for the task called t0 when it is executed on the unloaded CPU.

The results obtained with all these experiments are detailed in tables 1 and 2, and Figs. 1,2 and 3. In the tables the response time measured for t0(t_m) is presented together with the predicted response time for t0(t_P) and the percentage of relative prediction error (e). All the time values are measured in seconds. There are four predicted times and prediction errors because the four presented models are evaluated: *CPU availability model* (model A), *Round Robin model* (model B), *mixed model* (model C) and *RTP model* (model D).

Results in table 1 are reported in Figs. 1, 2 and 3. Figure 1 corresponds to a set of experiments with one CPU-bound task (t0) and one variable no CPU-bound task. This task increases its *X* value from t7 (*X* = *5* %) to t1 (*X* = 66%) and some of these results are not in tables for space restrictions. The prediction error for the CPU-bound task response time is plotted against the fraction of CPU time for the no CPU-bound task (defined as X in the previous section). This curve is plotted for the four discussed models. Figures 2 and 3 present the results for the same kind of experiments but with two and three

Table 1. Results with the four discussed models. Sets of experiments reported in figures.

Exp.	t_m	t_{P_A}	$e_A(\%)$	t_{P_B}	$e_B(\%)$	t_{P_C}	$e_C(\%)$	t_{P_D}	$e_D(\%)$
t0, t0	24.02	240.20	899.58	24.02	0.00	24.02	0.00	24.02	0.04
t0, t1	20.01	35.38	76.82	24.06	20.24	24.06	20.24	19.94	0.12
t0, t2	18.19	24.16	32.85	24.16	32.82	24.16	32.75	18.14	0.36
t0, t4	15.80	17.16	8.59	24.02	52.02	17.16	8.59	15.63	1.10
t0, t6	13.38	13.41	0.23	24.14	80.42	13.41	0.23	13.29	0.76
t0, t0, t0	36.03	480.40	1211.14	36.03	0.00	36.03	0.00	36.03	0.00
t0, t0, t1	32.14	70.65	119.81	36.03	12.10	36.03	12.10	32.00	0.53
t0, t0, t3	29.16	40.07	37.40	36.06	23.66	36.06	23.66	28.85	1.00
t0, t0, t5	26.80	30.20	12.69	36.24	35.22	30.20	12.69	26.60	0.72
t0, t0, t7	24.96	25.37	1.64	36.15	44.83	25.37	1.64	24.78	0.78
t0, t0, t0, t0	48.09	720.60	1398.44	48.04	0.10	48.04	0.10	48.04	0.10
t0, t0, t0, t1	40.20	106.24	139.00	48.16	8.35	48.16	8.34	44.29	0.21
t0, t0, t0, t2	40.20	72.06	68.96	48.04	12.64	48.04	12.64	42.28	0.68
t0, t0, t0, t4	40.19	51.51	28.18	48.06	19.63	48.08	19.63	39.63	0.21
t0, t0, t0, t6	37.80	40.20	6.35	48.24	27.62	40.20	6.35	37.45	0.83

Table 2. Results with the four discussed models. Remaining experiments

Exp.	t_m	t_{P_A}	$e_A(\%)$	t_{P_B}	$e_B(\%)$	t_{P_C}	$e_C(\%)$	t_{P_D}	$e_D(\%)$
t0, t5, t6	15.86	17.17	8.27	36.06	127.36	17.20	9.69	15.65	0.18
t0, t2, t7	18.67	26.69	42.95	36.03	92.98	26.84	43.71	18.72	0.24
t0, t1, t4	23.53	301.75	1182.41	36.21	53.89	36.21	54.02	23.66	0.54
t0, t3, t6	18.03	24.04	33.33	36.06	100.00	24.04	33.33	18.05	0.14
t0, t0, t6, t7	26.04	14.16	45.62	48.12	84.79	28.31	8.70	25.89	0.63
t0, t0, t5, t6	27.88	17.24	37.98	48.12	72.45	34.34	23.18	27.76	0.14
t0, t0, t4, t7	28.44	18.55	34.72	48.04	68.92	36.95	29.94	28.34	0.28

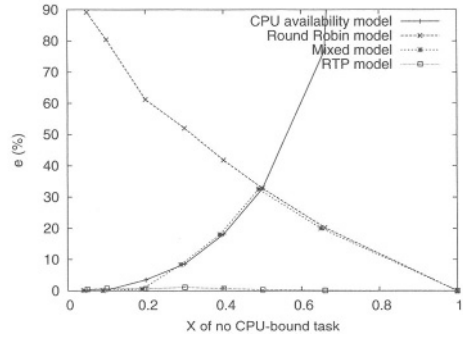


Fig. 1. Prediction error for the discussed models with one CPU-bound task and other varying task

CPU-bound tasks respectively. The remaining experiments, with other combinations of tasks, are showed in table 2.

From both, the tables and the figures, it is clear that large reductions in prediction errors are experienced in all the experiments using the *RTP model.* Indeed the prediction

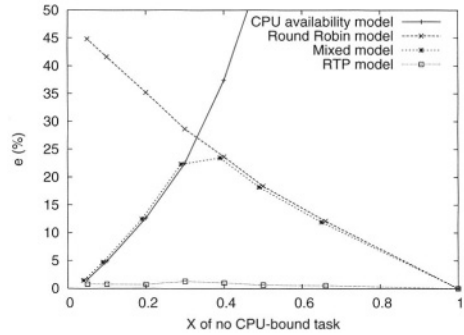


Fig. 2. Prediction error for the discussed models with two CPU-bound tasks and other varying task

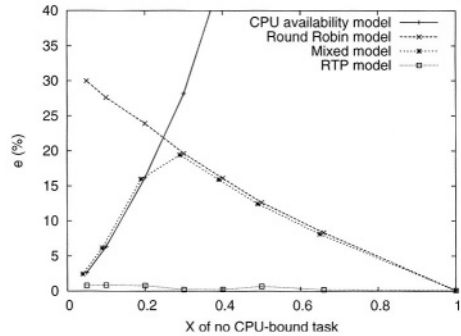


Fig. 3. Prediction error for the discussed models with three CPU-bound tasks and other varying task

error with this model is always less than 2 % . There are not instances in which one of the others models perform better than *RTP model.*

Figures 1, 2 and 3 show how the prediction error with the *CPU availability model* varies with the X value for the no CPU-bound task. For low X values the prediction error is low too, but as this value increases, the prediction error is sharply increasing. As it is said before, this can be attributed to the model, which ignores the possibility of time-sharing between tasks. A task with X around 50 % (near CPU-bound tasks) would share the processor time in a balanced way with the tasks in the run queue and it is opposed to the prediction made with this model, which predicts a very low CPU assignment for this task.

In contrast to this approach, the *Round Robin model* performs very well with large X values (near CPU-bound tasks) but the prediction error increases dramatically when X decreases. This was expected because this model does not take into consideration the remaining time of the CPU time slices left by no CPU-bound tasks. So, the assignment prediction when there are this kind of tasks in the run queue is always less than its real value.

Finally, for the *mixed model,* the value of *e* falls with low and large values of X. The previous results give some insight into why the error varies in this way. This model is proposed to take the best of the *CPU availability* and *Round Robin models.* Thus, the

mixed model curve converges with the *CPU availability model* curve at low values of X and with the *Round Robin* model curve at large values. Notice that the error increases for medium values of X, and this is the disadvantage of this model, although it supposes a considerable improvement over the two previous models because the prediction error does not increases indefinitely.

Still, even this last model is not superior to the *RTP model.* Besides the low prediction error values obtained with this model in all the experiments, figures and tables show the imperceptible dependence of this error on the kind of tasks considered.

5 Conclusions

The selection of a response time prediction model is nontrivial when minimal prediction errors are required. The main contribution of this paper is a detailed analysis of two existing and two new prediction models for all kind of tasks in a real computer system.

The *CPU assignment* concept has been introduced to predict response times. Using previous prediction models, the greatest source of error in making predictions comes from considering only the current system load. Thus, CPU assignment is introduced to consider the current system load and the effects of executing a new task on the CPU availability.

A wide variety of experiments has been performed to analyse, in terms of CPU assignment prediction, the *CPU availability* and *Round Robin models* accuracy. The results presented in the previous section reveal that these models perform very well in certain contexts but fail their predictions in others. In these cases error can reach more than a 100 %. These results suggest that it may not be unreasonable to combine these two models for improving their predictions. So, the first proposed model *(mixed model)* is a simple combination of these two discussed models. And the experimental results indicate that an important improvement over these models can be obtained.

Finally, an optimized and relatively simple model is proposed. The *RTP* model is based on a study of the CPU time sharing and the scheduling policies used by the operating system. This model takes into consideration the influence of a new task execution on the set of tasks in the run queue. Experimental results demonstrate the validity and accuracy of this model, the prediction error is always less than 2 %. Thus the *RTP* model has been shown to be effective, simple and very accurate under static conditions.

In the context of Beowulf clusters these results are encouraging. A very interesting line for future research is to extend the *RTP* model to dynamic environments. This may require some changes in the model to avoid using a priori information about tasks such as the tasks percentage of CPU utilization *(X).* But it would provide us with a dynamic model, very useful for predicting the response time of new incoming tasks on cluster nodes.

Acknowledgements. This work has been partially funded by the Spanish Commission for Science and Technology (grants CYCIT TIC2001-0175-03-03 and TIC2002-04486-C02-02).

References

1. Bell G. and Gray J. : What's next in high-performance computing? *Communications of the ACM,* 45(2):91–95, (2002).
2. Pfister G.F. : *In search of clusters: The Ongoing Battle in Lowly Parallel Computing, 2nd edn.* Prentice Hall, (1998).
3. Berman F.D. et al. : Application-level scheduling on distributed heterogeneous networks. In *Proceedings of the International Conference on Supercomputing,* (1996).
4. Spring N.T. and Wolski R. : Application level scheduling of gene sequence comparison on metacomputers. In *Proceedings of the International Conference on Supercomputing,* pages 141–148, (1998).
5. Wolski R., Spring N., and Hayes J. : Predicting the cpu availability of time-shared unix systems on the computational grid. In *Proceedings of the Eighth International Symposium on High Performance Distributed Computing,* pages 105–112. IEEE, (1999).
6. Mehra P. and Wah B.W. : Automated learning of workload measures for load balancing on a distributed system. In *Proceedings of the International Conference on Parallel Processing. Volume 3: Algorithms and Applications,* pages 263–270, (1993).
7. Dinda P.A. : Online prediction of the running time of tasks. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing,* pages 336–337, (2001).
8. Dinda P.A. : A prediction-based real-time scheduling advisor. In *Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium,* (2002).
9. Kunz T. : The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering,* 17(7):725–730, (1991).
10. Zhou S. : A trace-driven simulation study of dynamic load balancing. In *IEEE Transactions on Software Engineering,* pages 1327–1341, (1988).
11. Benmohammed-Mahieddine K., Dew P.M., and Kara M. : A periodic symmetrically-initiated load balancing algorithm for distributed systems. In *Proceedings of the 14th International Conference on Distributed Computing Systems,* (1994).
12. Lee G.H., Woo W.D., and Yoon B.N. : An adaptive load balancing algorithm using simple prediction mechanism. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications,* pages 496–501, (1998).
13. Shen K., Yang T., and Chu L. : Cluster load balancing for fine-grain network services. In *Proceedings of the International Parallel and Distributed Processing Symposium,* pages 51–58, (2002).
14. Ferrari D. and Zhou S. : An empirical investigation of load indices for load balancing applications. In *Proceedings of the 12th IFIP International Symposium on Computer Performance Modelling, Measurement and Evaluation.* Elsevier Science Publishers, (1987).

Space Decomposition Solvers and Their Performance in PC-Based Parallel Computing Environments

Radim Blaheta[1,2], Ondřej Jakl[1,2], and Jiří Starý[1]

[1] Institute of Geonics, Czech Academy of Sciences,
Studentská 1768, 708 00 Ostrava–Poruba, Czech Republic
{blaheta, jakl,stary}@ugn.cas. cz
[2] VŠB – Technical University Ostrava,
ul. 17. listopadu, 708 00 Ostrava–Poruba, Czech Republic

Abstract. The paper summarizes progress we have made since PPAM 2001 [1] in the parallel solution of large-scale FE systems arising from mathematical modelling in geomechanics. It tracks both the development of the mathematical methods implemented in the solvers, where the overlapping domain decomposition is contraposed to the displacement decomposition covered in [1], and also the performance advantage provided by a self-made Beowulf cluster for practical parallel computing.

1 Introduction

At PPAM 2001 we presented a paper [1] dealing with the solution of large-scale linear systems arising from the *finite element* (FE) analysis of elasticity problems. The systems were solved by the *conjugate gradient* (CG) method preconditioned by techniques based on the *displacement decomposition* (DiD). This means that nodal displacements in the coordinate directions x, y, z are separated and the positive definite *stiffness matrix* A is preconditioned by a block diagonal matrix B,

$$A = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{xy}^T & A_{yy} & A_{yz} \\ A_{xz}^T & A_{yz}^T & A_{zz} \end{bmatrix}, \qquad B = \begin{bmatrix} A_{xx} & & \\ & A_{yy} & \\ & & A_{zz} \end{bmatrix},$$

where the blocks correspond to the displacement decomposition.

The preconditioner B was realized in two different ways: First, the blocks A_{xx}, A_{yy}, A_{zz} were replaced by their *incomplete factorization,* which provides a *fixed preconditioner* (FP). Second, the subsystems with blocks A_{xx}, A_{yy}, A_{zz} were solved by a low-precision inner preconditioned CG method, controlled by the relative accuracy ε_0. This approach was called *variable preconditioner* (VP). In this case however, the use of outer CG method is not justified. The correct method can be constructed by a combination of the variable preconditioning with generalized CG method. More details can be found in [1,3,5].

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 153–160, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The DiD technique has a straightforward parallelization: It can be naturally implemented on three processors (each of the displacement directions x, y, z is processed by a separate *worker* process) or four processors (in case of a stand-alone *master* process, with minimal demands on computing resources), which is appropriate for small parallel systems. A higher level of parallelization could be achieved by applying an alternative technique for the parallelization of the solution of the subsystems with the blocks A_{xx}, A_{yy}, A_{zz}. For example, a combination of DiD and domain decomposition is tested in [5].

In this paper, the DiD approach is not further developed, but the code is still used as a benchmark for performance comparison of various parallel machines, here with a special emphasis on clusters of PC's (*Beowulf* clusters). This is the platform that we recently could set up ourselves. The details can be found in Sect. 2.

Beside DiD, we tested the overlapping *domain decomposition* (DD). The exploited two-level Schwarz preconditioner with a coarse space created by aggregation is described in Sect. 3. Section 4 is devoted to large-scale computations employing this method and gives comparison between our DiD and DD solvers on a commercial shared-memory machine and on our Beowulf cluster. Finally, some conclusions are given.

2 The Thea Cluster

Positive references and first experience, mentioned in [1] already, encouraged the decision to renew our in-house system for scientific computations in form of a cluster of personal computers. This cluster, named *Thea,* was assembled in 2001/02 from nine identical low-cost boxes (nodes) powered by the AMD Athlon 1.4 GHz processors and interconnected by a dual switched Fast Ethernet (100 MBit/s) network. The total cost of this hardware was about 13 000 €, i.e. less than a half in comparison with our previous system (an IBM workstation). In the considered configuration, one of the nodes operates as a provisional fileserver and the front-end simultaneously. This bottleneck will be eased by a dedicated two-processor fileserver with a Gigabit Ethernet channel to the switch.

The cluster runs Debian Linux, which we configured, having more or less user knowledge of Linux/Unix systems only, according to the recommendations found in literature, e.g. [2], and advice of more experienced administrators, making use of e.g. XFS filesystem, Cfengine and some other useful tools and packages.

After assembling the cluster, we wanted to compare its performance with other more professional computer systems. The systems that we could approach, including their relevant technical characteristics, are given in Tab. 1.[1]

For performance measurements, we employed our standard *Square Footing* benchmark (FOOT), a simple 3-D elasticity problem of soil mechanics, introduced in [1] already. The discretization accomplished by a rectangular grid of $41 \times 41 \times 41$ nodes, with a grid refinement under the footing, provided a linear

[1] Unfortunately, in most cases we do not know the price (price per processor) of the non-local systems to calculate the interesting price/performance ratio.

Table 1. Technical characteristics of systems under consideration

System	*Manuf.*	*Type*	*Processor/Frequency*	*Memory*	*Commun.*	*OS*
Thea	beowulf	cluster	8×AMD Athlon/1.4	8×768 MB	2×FastEth.	Linux
Beowulf	beowulf	cluster	42×AMD Athlon/1.2	42×1 GB	2×FastEth.	Linux
Lomond1	SUN	SMP	18×UltraSparcII/400	18 GB	shared mem.	Solaris
Lomond2	SUN	SMP	24×UltraSparcIII/750	48 GB	shared mem.	Solaris
Abacus	IBM	SMP	4×POWER3/375	4 GB	shared mem.	AIX
Parmac	beowulf	cl./SMP	4×2×PPC G4/450	4x512 MB	Fast Eth.	Linux

system of 206 763 equations. This relatively small-sized problem permits to test almost any computer platform, including memory constrained machines.

The results of our sequential solver PCG-S on this benchmark predicate the floating-point performance of the processor. Table 2 left gives the times of the FOOT/PCG-S combination on the systems introduced above, for the fixed *(FP)* and variable (*VP*) preconditioning. The first value *(Iter.)* is the time (in seconds) of the iterative phase of the computation, measured by the solver itself. For our purposes, it tells more then the subsequent *Total,* which as overall execution time includes also e.g. initial data reading, dependent on the disk subsystem performance (NFS vs. local), or the system dependent overhead related to the start of the application (e.g. through a batch queueing system).

Table 2. The FOOT benchmark, timings (in seconds) of the sequential PCG-S solver (*left*) and parallel PCG-DiD (*right*) solver on various platforms

	FP (51 it.)		*VP (11 it.)*	
Platform	*Iter.*	*Total*	*Iter.*	*Total*
Thea	23.2	23.6	44.8	45.3
Beowulf	27.6	31.9	53.3	57.9
Lomond1	92.7	97.8	136.0	139.9
Lomond2	57.2	58.9	93.7	95.4
Abacus	24.0	25.9	52.0	53.0
Parmac	78.8	79.7	121.1	122.0

		FP (51 it.)			*VP (11 it.)*		
Platform	*Par.lib.*	*Iter.*	*S-up*	*Total*	*Iter.*	*S-up*	*Total*
Thea	PVM	13.0	1.8	14.7	12.2	3.7	13.9
Beowulf	PVM	14.1	2.0	17.0	12.8	4.2	15.9
Lomond1	MPI	19.1	4.9	25.5	17.5	7.8	22.2
Lomond2	MPI	14.1	4.1	16.7	12.1	7.7	14.6
Abacus	PVM	9.5	2.5	13.8	8.1	6.4	12.7
Parmac	PVM	24.2	3.2	33.2	28.0	4.3	37.0

Table 2 left confirms the premier floating-point performance of the Athlon Thunderbird processors. Note the different performance ratios in FP and VP, most likely due to different data locality in the preconditioning operation. VP, although it has dramatically reduced the number of iterations (from 51 to 11), in the sequential case makes the solver even more than twice slower.

Let us now consider the parallel PCG-DiD solver, which with its four concurrent processes can serve as a basic comparison tool for any parallel environment. Its timings are given in Tab. 2 right, including the applied parallel library (PVM

or MPI – *Par.lib.*) and the relative speedup of the iterative phase of the computation *(S-up).*

From the figures, we can derive the following observations:

1. With parallel DiD solver, Thea lost its leadership in favour of Abacus. Its shared memory architecture is the base for an outstanding communication performance, which outweighs the lower processor performance.
2. Certainly for the same reason, the greatest leap in performance has been achieved by Lomond1 and its successor Lomond2, other commercial symmetric multiprocessors, even with superlinear speedup.
3. Superlinear speedup is characteristic for all VP measurements, so that VP became faster then FP. We try to explain this by better data locality, because VP concentrates the computation on smaller data arrays, where the cache hierarchy may work much more efficient. However, we cannot rule out some retarding bottleneck in the sequential version of VP, too.
4. Nevertheless the iteration phase on Thea was again superior to most other machines. The worse total times are due to the missing fileserver (see above).
5. We can demonstrate the importance of a fast communication subsystem for our parallel solvers by the following experiment: We substituted temporarily the Fast Ethernet switch (100 Mbit/s) by a 10Mbit/s Ethernet hub. Then the run times of PCG-DiD solver on Thea were as follows (Iter. (Total), in seconds): FP: 109.4 (163.2), VP: 32.8 (84.9). With data on local disks: FP: 110.3 (110.7), VP: 32.9 (33.3). Poor performance, even in comparison with the sequential results. On this slow network however, variable preconditioning amplifies its advantage.

This is to summarize the benefits we experienced so far with the Thea cluster:

- stable computing environment
- very good performance
- relatively low hardware costs
- minimal software purchases (the Portland Group Fortran compilers only)
- easy to expand (possibly with loss of homogeneity)
- its front-end used also as a Linux application server
- usable per node as personal computers after dismantlement

If some drawbacks should be mentioned, we must admit the non-trivial administration necessary to make the cluster run and keep it in good shape, if no professional support is available. As a loosely coupled system, the cluster is also not appropriate for large sequential jobs with great memory requirements.

3 Two-Level Schwarz Preconditioner

In this section, we shall use domain decomposition for partitioning of data and construction of the preconditioner and the parallel solver. More exactly, we shall consider the *additive Schwarz preconditioner,* see e.g. [7,3,5] for details. For simplicity, we restrict to the case of linear triangular or tetrahedral finite elements.

Let $\mathcal{T}_h(\Omega)$ be a FE division of Ω, $\mathcal{N}_h$ be the corresponding set of nodes, $\mathcal{N}_h = \mathcal{N}_1^0 \cup \ldots \cup \mathcal{N}_m^0$ be a splitting of $\mathcal{N}_h$ into disjunct subsets $\mathcal{N}_i^0$ and $\mathcal{N}_i \supset \mathcal{N}_i^0$ define another (overlapping) splitting. Let $\Omega_i^0 = \Omega(\mathcal{N}_i^0)$ and $\Omega_i = \Omega(\mathcal{N}_i)$ be subdomains of Ω, which are unions of elements with at least one vertex in the set $\mathcal{N}_i^0$ and $\mathcal{N}_i$, respectively. Then the splitting into $\mathcal{N}_i^0$ can be used for a decomposition of data and parallelization of the CG method and the splitting into $\mathcal{N}_i$ can be used for the construction of preconditioners.

If A is the stiffness matrix of the dimension n, then $\mathcal{N}_i$ defines a restriction operator $R^n \to R^{n_i}$ given by an $n \times n_i$ matrix R_i. Further, $A_i = R_i A R_i^T$ is a stiffness matrix corresponding to the subproblem on Ω_i with homogeneous Dirichlet boundary conditions on the inner boundary $\partial\Omega_i \backslash \partial\Omega$. The *(one-level) additive Schwarz preconditioner* (AS) is then given by the operator $B \sim A$,

$$B^{-1} = G_{\text{1l-AS}} = \sum_{i=1}^{m} R_i^T A_i^{-1} R_i. \tag{1}$$

The efficiency of this preconditioner increases with the amount of overlap, but decreases with the number of subdomains. However, greater overlap increases the size of subproblems and requires more (local) data transfers between subproblems.

The efficiency of the one-level preconditioner can be substantially increased by introducing correction from an auxiliary coarse grid problem. Similarly to the one-level case, we can define a coarse grid restriction operator R_0 and the subproblem matrix $A_0 = R_0 A R_0^T$. Then the *two-level additive Schwarz preconditioner* has the form

$$B^{-1} = G_{\text{2l-AS}} = R_0^T A_0^{-1} R_0 + \sum_{i=1}^{m} R_i^T A_i^{-1} R_i. \tag{2}$$

It is also possible to consider *hybrid preconditioners* with the subdomain and coarse grid corrections performed successively, e.g.

$$B^{-1} = G_{\text{2l-HS}} = R_0^T A_0^{-1} R_0 + \sum_{i=1}^{m} R_i^T A_i^{-1} R_i (I - A R_0^T A_0^{-1} R_0). \tag{3}$$

For a more complicated problem, it may be laborious to generate a separate coarse mesh, define the restriction R_0 and assemble the stiffness matrix A_0. For this reason, we use a coarse grid problem given by *aggregation.* It means that we divide the set of nodes $\mathcal{N}_h$ into nonoverlapping groups $\mathcal{G}_i$ and create coarse grid basis functions by summing the basis functions corresponding to all nodes in $\mathcal{G}_i$. The restriction operator R_0 is then represented by a Boolean matrix: The one's in its i-th row indicate positions of nodes belonging to $\mathcal{G}_i$. The matrix A_0 can be easily obtained by summing the elements of A in rows and columns belonging to the same groups.

In the next section, we shall consider applications of the additive Schwarz preconditioner with coarse grid created by aggregations. The efficiency of this method depends on the

- size of the overlap,
- amount of the aggregation,
- choice of the subproblem solvers.

What the overlap concerns, the optimal choice seems to be $\mathcal{N}_i$ consisting of all nodes from $\mathcal{N}_i^0$ and their neighbours. But the role of overlap is much less important for the two-level AS preconditioner then for the one-level one.

The amount of the aggregation is a more subtle issue. We used regular grids and groups of $k \times k \times k$ nodes with $k = 3, \ldots, 6$. This enables to reduce substantially the number of iterations and keep the coarse problem balanced with the subdomain problems. On the other hand even much stronger aggregation with $\mathcal{G}_i = \mathcal{N}_i^0$ can work and guarantee the numerical scalability of the algorithm, see [6].

The choice of the subproblem solver is another very important point. From our experience, the subdomain problems can be solved very roughly and we solve them only by means of incomplete factorization of matrices $A_1, \ldots, A_m$. On the other hand, the coarse grid problem deserves more effort: We solve it using inner CG iterations preconditioned again by incomplete factorization. The inner iterations are controlled by the relative accuracy $\varepsilon_0 = 10^{-1}$. This type of a nonlinear preconditioner as well as a nonsymmetric hybrid preconditioner can be used together with the generalized preconditioned CG method [3], [5].

In applications, we use one-dimensional DD along the z axis.

4 Large-Scale Computations

The *Dolní Rožínka* (DR) problem is an example of a large-scale practical modelling in geomechanics, dealing with underground mining in a uranium ore deposit. This problem gave rise to a benchmark, whose FE discretization by a regular structured grid of 124 × 137 × 76 nodes generates a linear system of 3 873 264 degrees of freedom. The solution parameters include 10^{-4} as the relative accuracy of the result and the null vector as its initial guess.

This large problem is an appropriate object for the solution by the domain decomposition solver PCG-DD, which is capable to utilize increasing number of processors. When experimenting with various run-time parameters, PCG-DD achieved its best results on DR when applied with the additive Schwarz preconditioner, small overlap (one grid layer) of the subdomains and with a global subproblem (coarse grid) generated by aggregation of six adjoining nodes in each direction (the coarse grid task solved 22176 equations). The diagram in Fig. 1 shows the times of the solution (the iterative phase only) for two platforms: Lomond2, i.e. SUN Fire 6800, declared by the producer as a midframe server, and Thea. For comparison, the times of the solution without a global subproblem are given, too. Moreover, the best PCG-DiD results (VP, with additional orthogonalization) are added as well.

We can observe that:

- The PCG-DD solver scales well with increasing number of processor nodes (up to eight) on both platforms, reaching its time minimum with 7 worker

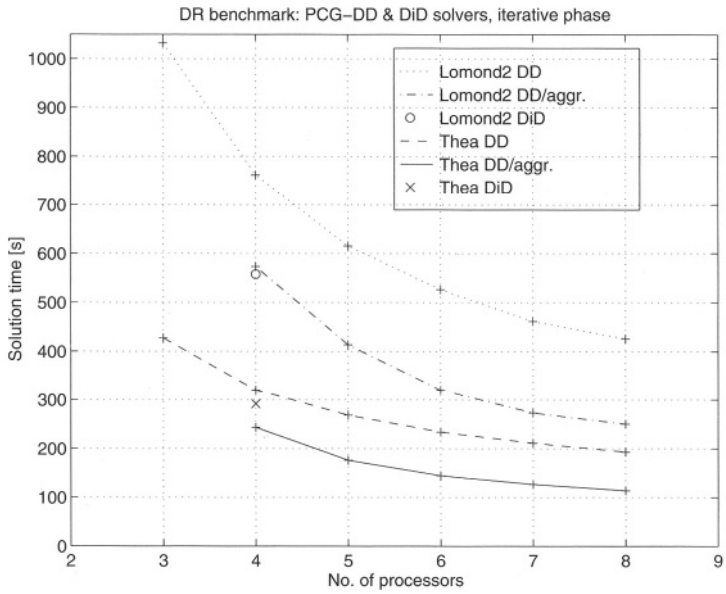


Fig. 1. The DR benchmark: timings of the parallel PCG-DD solver without/with aggregation on Lomond2 (SUN Fire) *(dotted/dashdot)* and Thea (Beowulf) *(dashed/solid)* on increasing number of processors. Compare with the PCG-DiD/VP solver on Lomond2 *(circle)* and Thea $(x\text{-}mark)$ on four processors.

processes (besides the master process running on a separate node). On Lomond2, the potential to scale with even more processors is greater, due to the communication subsystem taking advantage of its shared memory, by order of magnitude faster.[2]

- On both platforms, PCG-DD solver that makes use of the global subproblem generated by aggregation is faster than PCG-DD without it. Its advantage grows with increasing number of nodes, resulting in a time ratio greater than 2:3 with eight processors.
- The Thea performance is again superior to that one of Lomond2, at least twice in all cases.
- The PCG-DiD solver keeps pace with the best PCG-DD results on the same number of processors (four). Note that the times of the PCG-DD solver with aggregation on eight processors are at least twice shorter.
- The best time we could achieve for the solution of the DR benchmark on Thea was 114 s. It is about five times shorter than our best result (557 s) reported in [1]. This speedup, highly valuable for applied mathematical modelling, has been achieved by progress in both available hardware and employed algorithms.

[2] According to our measurements, the older Lomond1, certainly not faster than Lomond2, boasts of a bandwidth of 120 MByte/s and a latency about 11 μs (MPI communication), whereas Thea offers respectively 10.5 MByte/s and 125 μs (PVM).

5 Conclusions

In this paper we described two types of parallel solvers based on space decomposition – subspace correction idea. Both types, displacement decomposition solver and domain decomposition solver, were implemented using the message passing model and tested on selected benchmarks including large-scale real-life problems.

The paper showed the efficiency of the adopted methods and the remarkable capabilities which cheap clusters of PC's offer for their processing. Nevertheless, some further ideas as the use of other domain partitions, hybrid schemes or improved correction from the aggregated coarse space are under investigation.

Comparing with [1], the overall expansion of our technical capacities to solve large geomechanical problems benefits from both the development in algorithms and the progress in available computing resources.

Acknowledgements. The authors acknowledge the support of the Academy of Sciences of the Czech Republic through grant No. S3086102. The tests carried out outside the Institute of Geonics were possible thanks to SARA Amsterdam (Beowulf), EPCC Edinburgh (Lomond), CLPP BAS Sofia (Parmac) and TU Ostrava (Abacus).

References

1. Blaheta, R., Jakl, O., Starý, J.: Parallel displacement decomposition solvers for elasticity problems. In: Wyrzykowski, R. et al. (eds.): Parallel Processing and Applied Mathematics PPAM 2001. LNCS, Vol. 2328. Springer-Verlag, Berlin (2002) 395–403
2. Baker M. (ed.): Cluster Computing White Paper, December 2000. http://www.dcs.port.ac.uk/ mab/tfcc/WhitePaper, 10/2002
3. Blaheta, R.: GPCG – generalized preconditioned CG method and its use with nonlinear and nonsymmetric displacement decomposition preconditioners. Num. Linear. Alg. Appl., Volume 9, Issue 6-7 (2002) 527–550
4. Blaheta, R.: A multilevel method with overcorrection by aggregation for solving discrete elliptic problems. J. Comp. Appl. Math. 24 (1988) 227–239
5. Blaheta, R., Byczanski, P., Jakl, O., Starý, J.: Space decomposition preconditioners and their application in geomechanics. Mathematics and Computers in Simulation, 61 (2003) 409–420
6. Jenkins, E. W., Kelley, C. T., Miller, C. T., Kees, C. E.: An aggregation-based domain decomposition preconditioner for groundwater flow. SIAM Journal on Scientific Computing 23 (2001), 430–441
7. Smith, B. F., Bjørstad, P. E., Gropp, W. D.: Domain Decomposition. Parallel multilevel methods for elliptic partial differential equations. Cambridge University Press (1996)

Evaluation of Execution Time of Mathematical Library Functions Based on Historical Performance Information

Maciej Brzezniak and Norbert Meyer

Poznan Supercomputing and Networking Center
61-704 Poznan, Noskowskiego 12/14, Poland
{maciekb,meyer}@man.poznan.pl

Abstract. The paper presents the concept of a mechanism for predicting the execution time of mathematical library functions. The technique bases on the analysis of historical information concerning the performance of previous executions. The main application of the proposed mechanism is the prediction of the execution time of mathematical library functions in the Grid RPC systems. It can also be used in other heterogeneous and distributed environments where some parts of code are often executed and the prediction of execution time is crucial for high performance of computations and/or load balancing of the resources.

1 Introduction

The performance of computations led in cluster and distributed environments strongly depends on the accuracy of estimation of task execution time on particular nodes. While such evaluation is relatively simple in the dedicated laboratory installations, it is very difficult in real production systems. The difficulty results from the distributed, heterogeneous, shared and dynamic nature of these environments.

It is impossible to design and implement a universal task execution time prediction technique that would be efficient and accurate for various architectures of computing systems. It would require *a priori* knowledge of performance models of all kinds of computing systems, characteristics of particular subsystems and patterns of other (coexisting in the environment) applications' behaviour. Moreover, the dynamic character of production environments (changes of configuration, structure, and management policy) as well as continuous development of computing systems' architectures would require periodic updates of this knowledge. It is also difficult to design the execution time prediction technique for general type of tasks. It would demand knowledge of the task's algorithm, its resource requirements, communication patterns etc. Moreover, the course of a given execution of the algorithm may depend on the task's input parameters and many unpredictable conditions. In general, such information is not available *a priori.*

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 161–168, 2004.
© Springer-Verlag Berlin Heidelberg 2004

However, in several applications and environments some parts of code are often repeated. Moreover, several parameters of the algorithm realised by these parts of code are known (e.g. computational complexity). Our idea is to use the information concerning the performance of previous runs of these parts of code to predict the time that is needed for their future executions.

The main application of the mechanism is the prediction of the execution time of mathematical library functions in the Grid RPC [1][2] systems. Grid RPC allows the end-users to launch functions of mathematical libraries installed on remote machines from inside of their applications. The processing model is similar to the traditional RPC. The programmer calls the function by its name followed by a list of input and output arguments. The resources for computations (i.e. the computational node and the mathematical library) are assigned by the central module of the Grid RPC system.

The important feature of processing in the Grid RPC environments is that the particular library functions are repeatedly invoked. Although, the execution time prediction techniques currently used in the Grid RPC systems do not take advantage of that fact [3] [4] [5].

In our opinion, this feautre of the Grid RPC systems can be exploited. The paper presents the concept of the mechanism that automatically recognises the relation between the mathematical problem parameters, the computing system's state, architecture of the system and the task execution time basing on analysis of the historical performance information.

2 Definition of the Problem

Let us define *the mathematical function execution task* T_n. It is an instance of *mathematical problem P*. n is the *size* of this instance. Instance T_n of problem *P* can be solved using one of the mathematical libraries that are installed on the computing nodes. Each of these libraries implements *the algorithm A* that solves problem *P*. Evaluation of the execution time on particular machines is necessary, in order to realise a given scheduling policy in the environment.

The problem is how to evaluate the value of *execution time* ($time_{CA}(T_n)$) that is needed to solve instance T_n of mathematical problem *P* on the given machine *C* using algorithm *A*.

3 The Existing Solutions

The computational complexity theory defines *the computational complexity function* as the relation between size n of instance T_n of problem *P* and *the number of elementary operations* (N_{eo}) that are needed to solve this instance using algorithm *A*. Let us call this relation in short *the computational complexity function* f_{ccA}.

$$N_{eoCA}(T_n) = f_{ccA}(n) \tag{1}$$

The execution time prediction mechanisms used in Grid RPC systems exploit the $f_{ccA}(n)$ function. Suppliers of many mathematical libraries specify the f_{ccA} function of the algorithm implementing particular functions. That makes evaluating the number of elementary operations $N_{eoA}(T_n)$ that are needed to solve the instance T_n of problem P using a given version of the mathematical library (implementing the algotihm A) possible.

Evaluating the time that is needed to solve instance T_n of problem P on machine C (using algorithm A) requires calculating the time that is needed to perform a given number of elementary operations using the library installed on machine C:

$$time_{CA}(T_n) = time_C(N_{eoCA}(T_n)). \tag{2}$$

In the existing solutions the time that is needed to perform a given number (N_{eo}) of elementary operations $(time_C\ (N_{eo}))$ on the computing machine C is computed as:

$$time_C(N_{eo}) = \frac{N_{eo}}{V_{Creal}}, \tag{3}$$

where V_{Creal} is the real processing speed of machine C, i.e. the number of elementary operation that machine C is able to perform in a time unit. Determining V_{Creal} in shared environments is not trivial. It depends on the state (S_C) of computing system C. In the existing solutions, the relation between the state (S_C) of computing system C and its real processing speed V_{Creal} is reflected by *the performance model.* Let the f_{pmC} function represent this model. Then:

$$V_{Creal} = f_{pmC}(S_C). \tag{4}$$

The state (S_C) of computing machine C is a set of parameters of the computing system: the number of processors N, load L, the amount of available memory M and other performance-related information.

$$S_C = \{P, L, M, ...\}$$

The time that is needed to perform a given number of elementary operations (N_{eo}) that are needed to solve a given instance T_n of problem P on machine C using algorithm A can be expressed as (from (2) and (3)):

$$time_{CA}(T_n) = time_{CA}(N_{eoCA}(T_n)) = \frac{N_{eoCA}(T_n)}{V_{Creal}}. \tag{5}$$

Substituting in (5) $time_C(T_n)$ for $time_C\ (N_{eoC}(T_n))$ (see (2)), the right side of (1) for $N_{eoC}(T_n)$ and the right side of (4) for V_{Creal} we receive:

$$time_C(T_n) = \frac{f_{ccA}(n)}{f_{pmC}(S_C)}. \tag{6}$$

For a given instance of problem T_n value of n is known. Values of elements of the S_C set are predicted (e.g. using the Network Weather Service [8] [9]).

3.1 Limitations of the Existing Solutions

The presented method of evaluating the execution time of remotely invoked functions is used in the Grid RPC systems and other distributed environments, e.g. RCS [6] [7]. However, it has the following limitations.

Firstly, the computational complexity functions of some algorithms used in mathematical libraries may be unknown *a priori.* The form of the function or its coefficients may be unspecified by the library supplier. Therefore, it might be impossible to use this function to evaluate the number of elementary operations that are needed to solve instance T_n of problem P using some of the mathematical libraries.

Secondly, the number of elementary operations that is determined using the computational complexity function is a very high-level measure. It allows to evaluate the number of the algorithm steps, but it does not determine the number of machine instructions that implements (on the low level) the high-level algorithm steps. The execution of the same algorithm may require different number processor time, depending on processor architecture, level of binary code optimisation and other unpredictable factors.

Thirdly, the performance models of some computing systems are unknown *a priori,* i.e. at the moment of designing the prediction techniques. Therefore, the usage of these models is limited to the architectures that are known at the moment of designing the prediction mechanisms.

Fourthly, the performance models of computing systems that are used in the Grid RPC systems are relatively simple. They cannot reflect the features of complicated computing systems such as parallel machines, cluster systems etc.

Finally, the method of evaluating the execution time used so far in the Grid RPC systems does not take advantage of the fact that several mathematical library functions may be often invoked.

4 Proposed Solution

We propose another approach to the problem of predicting task T_n execution time. Seeing the limitations of the existing solutions, we desist from the usage of the computational complexity function (f_{ccA}) and the performance models of computing systems (f_{pmC}). Instead, we propose another model.

4.1 Definitions

Let us define *instance* $T_{n,OP}$ *of problem* P as the mathematical function execution task. n_T is the *size* of this instance. OP_T includes *other run-time parameters of the remote function call* that can be specified by the programmer (e.g. algorithm version).

Let us also define *work* W_{CT} that has to be done to solve this instance of the problem on *computing machine C.* This work is defined as the amount of time that is spent by the process solving the problem in the user and system mode

on machine *C*. It is not the real time. It can be measured e.g. using standard *Unix* system mechanisms.

Let us also define *function of work* f_{CT} that reflects the relation between size n_T and other parameters OP_T of instance $T_{n,OP}$ of problem *P* and work W_{CT} that must be done on computing system *C* to solve this instance of the problem.

$$W_{CT} = f_{CT}(n_T, OP_T) \tag{7}$$

We decided to use the function of work instead of the computational complexity function as the measure of the work that must be done to solve a given mathematical problem on a given computing system. The main motivation is the fact that the time spent by the process in the user and system mode can be measured. On the contrary, it is not possible to measure the number of elementary operations that have been done by the process when solving a given problem on the computing system. We also observed that the system and user time that is needed to execute several parts of code (e.g. mathematical library function) depends not only on the number of elementary operations of the algorithm. It also depends on the other factors, e.g. the degree of binary code optimisation and the quality of the compiler used to prepare the binary code.

Let us define *machine performance model function* g_C. It reflects the relation between the state S_C of the machine *C,* the amount of work *W* to be done on the machine and the amount of real time *(timeW)* that is needed to do this work on *C*.

$$time_W = g_C(W, S_C) \tag{8}$$

The g_C function reflects the scheduling and resource assignment policies in a given computing system.

Let us also extend (in comparison to the existing solutions) the set of parameters considered as the elements of the state (S_C) of computing system *C*. Let it include: number of processors *N,* load *L,* amount of available memory *M*, number of processes in the run queue *R,* intensity of I/O operations *I* and other performance-related information. All this data can be acquired in *Unix* operating systems e.g. using simple user-level commands as *ps, uptime, sar.*

$$S_C = N, L, M, R, I, ...$$

The extension aim in providing the prediction mechanisms the ability to consider the widest possible range of system state' parameters.

Substituting in (8) W_{CT} for *W* we receive the formula for computing the real time that is needed to solve the instance $T_{n,OP}$ of problem *P:*

$$time_{W_{CT}} = g_C(W_{CT}, S_C). \tag{9}$$

For a given instance $(T_{n,OP})$ of the problem, values of n_T and OP_T are known. Values of elements of the S_C set can be predicted using the external prediction systems, e.g. Network Weather Service.

The evaluation of the time that is needed to solve a given instance $(T_{n,OP})$ of the problem *P* requires knowledge of the form and coefficients of f_{CT} and g_C functions.

4.2 Statements

The form and coefficients of the function of work f_{CT} may be reconstructed by using the experimental data. It requires collecting measurements of works W_{CT} done to solve particular instances $T_{n,OP}$ of problem P with the usage of a mathematical library installed on machine C as well as measurements of the parameters' values (n_T and OP_T) that are describing these task instances. Having collected enough measurements of values of the corresponding n_T, OP_T and W_{CT} quantities makes the reconstruction of the f_{CT} function possible.

The form and coefficients *of the machine performance model function g_C can be reconstructed from the experimental data.* That requires collecting measurements of the values of the work W for particular tasks as well as the values of real times ($time_W$) that are needed to do these works on system C. The parameters' values that are describing state S_C of system C while doing these portions of work should also be gathered. Having gathered enough measurements of values of the corresponding $time_W$, W, and S_C parameters, makes the reconstruction of the g_C function possible.

4.3 Detailed Considerations

The reconstruction of the f_{CT} and g_C functions requires running a large number of test computations. In practice, the cost of the experimental reconstruction of the functions of work (f_{CT}) for all the implementations of a given mathematical functionality that exist in the environment may be unacceptable. Similarly, the cost of reconstructing the performance model functions (g_C) of all computing systems in the environment may be unacceptable.

The designed mechanism is intended to be used for evaluating future execution times of these parts of code that are often used. Therefore, the functions of work f_{CT} of the implementations of mathematical functions that arc often invoked must be analysed. Similarly, performance models g_C of the computing systems that are used intensively have to be reconstructed. Therefore, we assume that the reconstruction is done continuously during the Grid RPC system work.

The mechanism has two phases of work. In the initial state – *gathering phase* – prediction mechanism uses Grid RPC-native methods for evaluating the execution time. During these executions, for each implementation of mathematical function that has been used, values of the problem's parameters (n_T, OP_T) and values of the corresponding works (W_{CT}) are collected. Similarly, for each computing system exploited by the computations, values of works W, values of state parameters (S_C) and the quantities of real times that have been needed to do the relevant works are gathered. When the sufficient number of "probes" is collected, the f_{CT} and g_C functions are reconstructed. Then they can be used to predict the future execution times of mathematical library functions, that is the *exploitation phase* begins.

In the *exploitation phase* the execution time prediction is performed with use of reconstructed f_{CT} and g_C functions. In order to discover possible changes of the features of the mathematical libraries and computing systems, the analysis

of performance data and reconstruction of the f_{CT} and g_C functions are continuously performed. In addition, the accuracy of the predictions done by using the reconstructed f_{CT} and g_C functions is periodically verified. If it degrades, the mechanism returns to the *gathering phase.*

The processes of reconstructing the f_{CT} and g_C functions have two stages. In case of the f_{CT} function, in the first stage, the subsets of parameters of the problem OP_T that may be significant for the form of the f_{CT} function are selected. Similarly, in the first stage of reconstruction of the g_C function, subsets of parameters of computing system's state that may be significant for the form of the g_C function are selected. It is known that the size (n) of a given instance $T_{n,OP}$ of problem P has influence on the amount of work that should be done in order to solve this instance. However, it is necessary to evaluate the influence of other elements of the parameter set (OP_T) describing the problem instance $T_{n,OP}$ on the value of the work that should be done. The evaluation requires the analysis of the correlation between the parameters of particular problem's instance and values of W_{CT} for the given (constant) size of the problem (n). It is also necessary to recognise what parameters of state S_C of system C have influence of the value of $time_W$ for the given amount of work *(W).* The evaluation requires the analysis of the correlation between particular elements of the S_C set and values of $time_W$ for the given (constant) work *W.* As the result of the evaluation appropriate weights are assigned to elements of the OP_T set and the S_C set. These weights can be useful in reconstructing the form and coefficients of the f_{CT} and g_C functions (the second stage of the reconstruction).

The details of methods for reconstructing the f_{CT} and g_C functions are subject of present work. Example considered method for reconstructing the f_{CT} function tries to fit some templates of the f_{CT} function to the experimental data. Templates can include linear and non-linear functions, polynomial and exponential functions as well as the known form of the f_{ccA} function of the algorithm implementing the given mathematical functionality (if specified by the library supplier). The mechanism tries to fit coefficients of template functions to the experimental data concerning executions of the given mathematical library function. Other methods of reconstruction are also taken into consideration.

5 Summary

The paper provides the concept of the execution time prediction mechanism. The proposed solution copes with the limitations of the existing methods. It takes advantage of the fact that particular mathematical library functions are often executed in the Grid RPC systems.

The mechanism is able to recognise the performance-related features of the mathematical functions' implementations even if computational complexity of the algorithm is unknown *a priori.* The mechanism can also recognise the performance characteristics of computing systems. The recognition is based on analysis of historical information concerning the performance of previous executions of mathematical functions on particular computing systems. Additionally, the

mechanism is able to discover possible changes of performance-related features of the computing resources (mathematical library functions and computing machines) dynamically as the prediction module works continuously.

In our opinion, the proposed technique may improve the accuracy of prediction of execution time of mathematical library functions in Grid RPC systems. It may be also exploited in cluster and distributed environments, where several parts of code are often executed and the performance characteristics of code parts and computing systems are *a priori* unknown and may change dynamically. We believe that the technique is cost-efficient, since only the features of often-used computing resources are analysed.

The current focus is to work out the methods for reconstructing f_{CT} and g_C functions. The accuracy, efficiency and costs of the mechanism are going to be evaluated in real production environments.

We plan to exploit the results of this work in order to optimise the usage of mathematical libraries in Grid environment. These efforts are broadly discussed in [10].

References

1. Grid RPC: A Remote Procedure Call API for Grid Computing. K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, H. Casanova, ICL Technical Report, ICL-UT-02-06, June, 2002.
2. Overview of Grid RPC: A Remote Procedure Call API for Grid Computing. K. Seymour, H .Nakada, S. Matsuoka, J. Dongarra, C. Lee and H. Casanova. Grid Computing – Grid 2002, LNCS 2536, pp. 274-278, November, 2002.
3. NetSolve: A Network Server for Solving Computational Science Problems. H. Casanova and J. Dongarra. The International Journal of Supercomputer Applications and High Performance Computing, Vol. 11, Number 3, pp. 212-223, 1997.
4. Utilizing the Metaserver Architecture in the Ninf Global Computing System. H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, S. Sekiguchi. High-Performance Computing and Networking '98, LNCS 1401, pp. 607-616
5. Innovation of the NetSolve Grid Computing System. D. Arnold, H. Casanova, J. Dongarra. To appear in Concurrency and Computation: Practice and Experience, 2002.
6. The Remote Computation System. P. Arbenz, W. Gander, M. Oettli. Parallel Computing (23): 1421-1428, 1997.
7. The Remote Computation System. P. Arbenz, W. Gander, M. Oettli. High.Performance Computing and Networking, H. Liddell, A. Colbrook, B. Hertzberger and P. Sloot (eds.). Springer-Verlag, Berlin, 1996, pp. 820-825. (Lecture Notes in Computer Science, 1067).
8. Dynamically Forecasting Network Performance Using the Network Weather Service. R. Wolski, Journal of Cluster Computing, Vol. 1, pp. 119-132, January, 1998.
9. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. R. Wolski, N. Spring, and J. Hayes, Journal of Future Generation Computing Systems, Vol. 15, Numbers 5-6, pp. 757-768, October, 1999.
10. Optimisation of Usage of Mathematical Libraries in the Grid Environment. M. Brzezniak, N. Meyer. Proceedings of Second Cracow Grid Workshop, pp. 74-86, Cracow, Poland, 2003.

Empirical Modelling of Parallel Linear Algebra Routines*

Javier Cuenca[1], Luis-Pedro García[2], Domingo Giménez[3], José González[4], and Antonio Vidal[5]

[1] Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia. Apartado 4021, 30001 Murcia, Spain
javiercm@ditec.um.es

[2] Servicio de Apoyo a la Investigación Tecnológica, Universidad Politécnica de Cartagena. Paseo Alfonso XIII, 30203 Cartagena, Murcia, Spain.
luis.garcia@sait.upct.es

[3] Departamento de Informática y Sistemas, Universidad de Murcia. Apartado 4021, 30001 Murcia, Spain
domingo@dif.um.es

[4] Intel Barcelona Research Center Intel Labs, Barcelona
pepe.gonzalez@intel.com

[5] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. Cno. Vera s/n, 46022 Valencia, Spain
avidal@dsic.upv.es

Abstract. This paper shows some ways of combining empirical studies with the theoretical modellization of parallel linear algebra routines. With this combination the accuracy of the model is improved, and the model can be used to take some decisions which facilitate the reduction of the execution time. Experiments with the QR and Cholesky factorizations are shown.

1 Introduction

The modellization of sequential or parallel linear algebra routines can be carried out by simulations [5] or by analytic models [3,6]. The models enable some decisions to be taken:

- In algorithms by blocks it is interesting to obtain a block size which provides execution times close to the optimum. This decision could be included in the installation of the routines or libraries to obtain paramaterized libraries [4].
- It is also of advantage to select the number of processors and the topology to be used. The best number of processors and topology for a particular routine will vary from one system to other or with the system workload [2].

* This work has been funded in part by CICYT, project number TIC2000/1683-C03, and Fundación Séneca, project number PI-34/00788/FS/01, and has been developed using the systems of the CEPBA, the CESCA and the ICL at the University of Tennessee.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 169–174, 2004.
© Springer-Verlag Berlin Heidelberg 2004

- The solution of the problem is carried out normally by calling basic routines, which can be taken from one library or another. The use of the best library in each one of the basic routines can also reduce the execution time [1].

It is difficult to develop useful models for a wide range of systems, routines and problem sizes. In this paper we propose the use of some empirical studies to improve the models' accuracy, and we show how the ideas are applied when using the QR and Cholesky factorizations. These ideas have been successfully applied to other routines [1]. The model can be combined with a library development methodology to design adaptive libraries or a hierarchy of adaptive libraries.

2 Improvements in the Model

The traditional basic model of a parallel linear algebra routine includes a parameter for the computation cost, another for the start-up time and another for the word-sending time. We propose some modifications in the traditional formulae and in the way they are used to predict the execution time:

- The value of the parameters in the formula may vary with the problem size. It could be preferable to separate the formula into different parts, and to obtain different values for the parameters in each part.
- The accuracy of the parameters in the model is traditionally improved in various ways [2,3]: when arithmetic operations of levels two and three appear in the algorithm it is better to use different parameters for the operations of different levels; it is also possible to use different parameters for different routines of the same level; the values can also be different for the same routine but with different data storage schemes; and the way in which the data are accessed in the algorithm also influences the value of the parameters.
- When different types of communications are used in the program, the cost of the communication parameters varies: the start-up and the word-sending times will be different when using point to point communications from when using broadcast.
- The communication cost varies with the system. Traditionally, different models are used for different systems, but a parameter could be included to model the influence of the architecture in the cost.

Since the values of the parameters depend on a set of other parameters, the parameters for the costs in the system can be represented as a function of these algorithmic parameters. Hence, a multidimensional function is obtained for each parameter. The function will be represented by a table with a number of dimensions equal to the number of algorithmic parameters which significatively influence the value of the system parameter. In all the cases the value must be obtained by experiments with the basic operations in the algorithm, and with the data storage and access scheme used in the algorithm. What we propose is to identify the basic routines most widely used, together with the size of the parameters and the data storage and access schemes, and to develop the

basic libraries including an engine to estimate these values as they are installed. Thus, using the model of the execution time of the algorithm and the values of the parameters estimated when the basic libraries were installed, the selection of satisfactory values of the parameters can be made automatically and without important additional cost. The user can also be provided with intelligent routines which will execute efficiently, and independently of the system where they are executed.

3 Experimental Results

3.1 Division of the Model in Different Formulae

Two non block versions of the Cholesky factorization are considered. The block versions of the factorization work better than these non block versions, but it is more difficult to model low level routines, and what we want is to show how more accurate modellizations can be obtained.

In a matrix-vector product version the cost is $\frac{1}{3}k_2n^3$, with k_2 corresponding to the BLAS 2 routine `dgemv`. With only this parameter in the model, the prediction of the execution time was not satisfactory. To improve the model the computation has been divided into four parts to include the cost of the parameters depending on the size and form of the submatrices. The model is:

$$\frac{5}{96}n^3k_{gemv_{\frac{7n}{8}\times\frac{n}{8}}} + \frac{11}{96}n^3k_{gemv_{\frac{5n}{8}\times\frac{3n}{8}}} + \frac{11}{96}n^3k_{gemv_{\frac{3n}{8}\times\frac{5n}{8}}} + \frac{5}{96}n^3k_{gemv_{\frac{n}{8}\times\frac{7n}{8}}} \quad (1)$$

where the operations have been grouped for vector sizes between 1 and $\frac{n}{4}$, $\frac{n}{4}+1$ and $\frac{n}{2}$, $\frac{n}{2}+1$ and $\frac{3n}{4}$, $\frac{3n}{4}+1$ and $n-1$.

In an outer product version, the operations of order n^3 are performed with the `daxpy` operation. The cost is divided into four parts:

$$\frac{5}{96}n^3k_{axpy_{\frac{7n}{8}}} + \frac{11}{96}n^3k_{axpy_{\frac{5n}{8}}} + \frac{11}{96}n^3k_{axpy_{\frac{3n}{8}}} + \frac{5}{96}n^3k_{axpy_{\frac{n}{8}}} \quad (2)$$

The values of the parameters are obtained by performing a set of matrix updating on vectors in consecutive columns in the matrix, like in the algorithm. If the values are obtained with updating in non consecutive vectors, the theoretical prediction is much worse.

Experiments have been performed in a Pentium III and a Pentium 4, and with reference BLAS (BLASref) and a specific BLAS (BLASspe) for Pentium III and Pentium 4. The modelled (mod.) and experimental (exp.) times are compared in Table 1. The way in which the values of the parameters in the model are obtained for each problem size produces a good estimation of the execution time and consequently a good selection of the best library and algorithm to use. The optimal decision is not always the obvious, but it is well taken using the model.

Table 1. Non block Cholesky factorization, in Pentium III and 4, and using different libraries and algorithms. Execution times in seconds

	matrix-vector product				outer product			
	BLASref		BLASspe		BLASref		BLASspe	
n	mod.	exp.	mod.	exp.	mod.	exp.	mod.	exp.
				PentiumIII				
256	0,117	0,058	**0,081**	**0,041**	0,141	0,061	0,129	0,058
512	0,826	0,702	**0,350**	**0,335**	1,099	1,021	1,135	1,041
1024	5,982	5,734	**2,660**	**2,615**	7,889	8,749	7,952	8,872
2048	46,135	45,450	**18,931**	**19,199**	61,817	71,544	64,784	76,184
				Pentium4				
256	0,032	0,020	**0,026**	**0,014**	0,192	0,024	0,188	0,021
512	**0,179**	**0,154**	0,476	0,416	0,446	0,218	0,441	0,209
1024	**1,350**	**1,118**	4,690	3,960	2,244	1,646	2,167	1,596
2048	**9,551**	**8,586**	29,667	28,792	13,316	13,048	12,550	11,934

3.2 Selection of Parameters

In an algorithm by blocks the block size and the library to be used must be decided. A "dot product" version by blocks of the Cholesky factorization has been studied. A and G are divided in blocks A_{11}, A_{21}, A_{21}^T, A_{22}, G_{11}, G_{21} and G_{22}; with block A_{11} of size $b \times b$, A_{21} of $(n-b) \times b$, and A_{22} of size $(n-b) \times (n-b)$, and G_{11} and G_{22} lower triangular. The Cholesky factorization of the diagonal blocks is computed using the level 2 routine of LAPACK `dpotf2`. The factor for blocks in the lower triangular part ($G_{21} = A_{21}(G_{11}^T)^{-1}$) is obtained using the level 3 routine of BLAS `dtrsm`. And the blocks in the lower triangular part and in the diagonal are updated using the level 3 routines of BLAS `dgemm` and `dsyrk` ($\tilde{A}_{22} = A_{22} - G_{21}G_{21}^T = G_{22}G_{22}^T$). The execution time is modelled:

$$2k_{3,gemm}\left(\frac{n^3}{6} - \frac{n^2 b}{2} + \frac{nb^2}{3}\right) + k_{3,trsm}\left(\frac{n^2 b}{2} - \frac{nb^2}{2}\right) + k_{3,syrk}\left(\frac{n^2}{2} - \frac{nb}{2}\right)(b+1) + k_{2,potf2}\frac{nb^2}{3} \quad (3)$$

where the k_3 values depend on that of b, and are practically constant when the matrix size varies.

Table 2 shows the execution time predicted by the model (model), the lowest experimental time obtained by performing experiments with different block sizes (exp. low.) and the time when the value of b provided by the model is used (exp. mod.). The model gives a good prediction for the execution time, leading to a good library and parameter selection.

3.3 Selection of Parameters in Parallel Algorithms

For the parallel block QR factorization the cost is modelled by:

$$\frac{4}{3}k_{3,gemm}\frac{n^3}{p} + \frac{1}{4c}bk_{3,trmm}n^2 + \frac{1}{r}bk_{2,geqr2}n^2 + \frac{1}{2r}nk_{2,larft}n^2 + t_s\frac{n}{b}((2+3b)\log r + 2\log c) + t_w\left(n^2\left(\frac{r-1}{r^2} + \frac{\log r}{2c} + \frac{\log r}{2r}\right) + nb\log p\right) \quad (4)$$

Table 2. Comparison of the execution time (in seconds) and the block size of the block Cholesky factorization using reference and optimized BLAS. In Pentium III and Pentium 4

	BLASref					BLASspe				
	model		exp. (low./mod.)		dev.	model		exp. (low./mod.)		dev.
n	time	blo.	time	blo.	(%)	time	blo.	time	blo.	(%)
PentiumIII										
256	0,036	32	0,025/0,029	64/32	60/19	0,020	64	**0,014**	64	30
512	0,280	32	0,279	32	0,25	0,127	64	**0,114**/0,116	32/64	24/8,5
1024	2,196	32	2,193	32	0,14	0,847	128	**0,817**	128	3,5
2048	17,390	32	17,318	32	0,42	5,996	128	**5,805**	128	2,7
Pentium4										
256	0,016	32	0,010/0,012	64/32	42/25	0,008	64	**0,007**	64	15
512	0,117	32	0,112	32	4	0,051	64	**0,050**	64	1,5
1024	0,908	32	0,922	32	1,5	0,306	128	**0,307**	128	0,4
2048	7,169	32	8,105	32	12	1,985	128	**2,015**	128	1,5

Table 3. Comparison of the execution time (in seconds) of the QR factorization. In a network of 8 Pentium III with Fast-Ethernet

	n	128	256	384	1024	2048	3072
mod.	time	0.025	0.087	0.178	1.92	9.90	28.00
	processors	1×2	1×8	1×8	1×8	1×8	1×8
	block size	8/16	8	8	16	16	32
	library	BLASII	BLASII	BLASII	BLASII	BLASII	BLASII
low.	time	0.025	0.086	0.176	1.92	9.40	25.11
	processors	1×2	1×8	1×8	1×8	2×4	2×4
	block size	16	8	8	16	32	32
	library	BLASII	BLASIII	BLASIII	BLASII	BLASII	BLASII

Table 4. Parameters selection for the QR algorithm, in different systems

	IBM SP2		Origin 2000		nSUN	nPIII	
	p=4	p=8	p=4	p=8	p=4	p=4	p=8
n	b r c	b r c	b r c	b r c	b r c	b r c	b r c
1024	16 1 4	16 1 8	32 4 1	32 4 2	16 1 4	16 1 4	16 1 8
2048	32 1 4	16 1 8	64 4 1	32 4 2	16 1 4	16 1 4	16 1 8
3072	32 1 4	32 2 4		32 4 2		32 1 4	32 1 8
4096	32 1 4	32 2 4		64 4 2		32 1 4	32 1 8

The matrix is distributed in a logical two-dimensional mesh of $p = r \times c$ processors, with block size b.

Experiments have been carried out in a network of Pentium III with Fast-Ethernet. Three libraries have been used: ATLAS, a BLAS for Pentium II Xeon (BLASII), and a BLAS for Pentium III (BLASIII). In Table 3 the execution time with the parameters provided by the model (mod.) and the lowest execution time (low.) are shown together with their parameters. The times with the model are always close to the lowest times.

3.4 Adjustment of the Communication Cost

In the QR factorization Formula 4 is valid when the broadcast is performed using a hypercube topology, but in a network of processors the logarithmic terms

would be linear. The inclusion in the model of a parameter to reflect this situation means the execution time is well predicted. The predicted values of the parameters in an IBM-SP2, an Origin 2000, a network of SUN Ultra 1 with Ethernet (nSUN) and a network of Pentium III with Fast-Ethernet (nPIII) are shown in Table 4. In all the cases the selection is satisfactory. The values of the parameters predicted vary for different systems and problem sizes.

4 Conclusions and Future Works

We have shown some modifications to the traditional modellization of parallel linear algebra routines. The combination of the model with the experimental estimation of some system parameters allows a better selection of algorithmic parameters, and the possibility of designing routines adaptable to systems of different characteristics. At present we are working on the inclusion of an engine for the automatic obtention of the parameters in parallel linear algebra libraries.

References

1. Alberti, P., Alonso, P., Vidal, A., Cuenca, J., García. L. P., Giménez, D.: Designing polylibraries to speed up linear algebra computations, TR LSI 1-2003, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Murcia (2003). dis.um.es/~domingo/investigacion.html.
2. Cuenca, J., Giménez, D., González, J., Dongarra, J., Roche, K.: Automatic Optimisation of Parallel Linear Algebra Routines in Systems with Variable Load. In 11th EUROMICRO Workshop on Parallel, Distributed and Networked Processing PDP 2003, IEEE (2003) 409-416.
3. Dackland, K., Kågström, B.: A Hierarchical Approach for Performance Analysis of ScaLAPACK-based Routines Using the Distributed Linear Algebra Machine, 3th International Workshop, PARA 1996, LNCS Vol 1184 (1996).
4. Dongarra, J.: Self Adapting Numerical Software (SANS) for Grid Computing. Innovative Computing Laboratory, University of Tennessee (2002), http://icl.utk.edu/iclprojects/pages/sans.html.
5. Kvasnicka, D. F., Ueberhuber, C. W.: Developing Architecture Adaptive Algorithms using Simulation with MISS-PVM for Performance Prediction. 11th ACM Conference on Supercomputing (1997).
6. Yunquan Zhang, Ying Chen: Block Size Selection of Parallel LU and QR on PVP-based and RISC-based Supercomputers. R & D Center for Parallel Software, Institute of Software, Beijing, China (2003) 100-117.

Efficiency of Divisible Load Processing

Maciej Drozdowski* and Łukasz Wielebski

Institute of Computing Science, Poznań University of Technology,
ul. Piotrowo 3a, 60-965 Poznań, Poland
{Maciej.Drozdowski,Lukasz.Wielebski}@cs.put.poznan.pl

Abstract. Effective exploitation of a parallel computer system is a result of cooperation between the communication and the computing parts of a parallel computer system, and the application. These elements are described by various parameters. It is not easy to grasp the connection between the values of particular parameters and the efficiency of parallel computations. In this paper we propose a new way of representing the relations between the parameters of a parallel computer system, and its performance. Results of simulations are presented and discussed.

Keywords: Performance of parallel and distributed systems, divisible load theory, communication delays, clusters, grid.

1 Introduction

Performance of a parallel computer system is guided by complex relations between the computer system components and the application. Understanding the reasons for the limitations in the performance is the first step towards using the power of the hardware effectively.

In this paper we make an attempt to gain new insights into the relationships between the performance of parallel systems and their key parameters. The method we propose is built on the basis of *divisible load theory* (DLT) [2, 3,4], and *isoefficiency function* [6]. DLT assumes that computations (the load) can be divided into parts of arbitrary sizes. The parts can be processed independently by remote computers. DLT appeared to be useful in representing distributed computations in various interconnection topologies. It has become not only theoretically useful, but also practically viable which has been demonstrated by a series of parallel applications [5]. Surveys of DLT can be found in [2,3,4]. The isoefficiency function has been defined as problem size required to sustain constant efficiency for changing processor number. We generalize this concept.

The rest of this paper is organized as follows. In Section 2 the notion of isoefficiency is presented, and generalized. In Section 3 a DLT model of a distributed computation is presented. Section 4 gives results of performance modeling, and discusses the consequences.

* This research was partially supported by the grant of Polish State Committee for Scientific Research.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 175–180, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 The Concept of Isoefficiency

In this section we present, and generalize, the concept of isoefficiency function defined in [6].

Let m denote the number of processors used by a parallel application. We will denote by $T(m)$ the execution time of a parallel application when run on m processors. By speedup we mean ration $S = \frac{T(1)}{T(m)}$. Speedup is the most commonly used measure of a parallel application performance. *Efficiency* $E = \frac{S}{m} = \frac{T(1)}{T(m)m}$ is a measure derived from speedup. Note that both speedup, and efficiency are dimensionless measures. In a sense speedup indicates how much an application can be accelerated by using additional processors. The higher speedup is the better. Efficiency is an indicator showing how well a parallel application uses the processors. Efficiency can be also understood, as an average fraction of all m processors that really work in the parallel application. Thus, the closer efficiency to 1 is, the better.

It has been observed that speedup and efficiency depend on the size of the problem that a parallel application solves. It is hard to maintain high efficiency for small problems and big processor number m. Consequently, with growing processor number m also problem size should increase in order to preserve constant efficiency. This observation resulted in a definition of the isoefficiency function $f(k, m)$ in [6], as the problem size for which efficiency of a parallel application is $E = k$ on m processors. For example, consider an algorithm calculating a minimum spanning tree in a graph with n vertices, where n expresses the size of the problem. There exists an algorithm [1] with running time $T(m) = \frac{c_1 n^2}{m} + c_2 n \log m$, and $T(1) = c_1 n^2$, where c_1, c_2 are constants. Thus, efficiency of this algorithm is $E = \frac{c_1 n^2}{c_1 n^2 + c_2 m n \log m} = k$. In order to maintain constant efficiency value $k < 1$, the size of the problem should be $f(k, m) = \frac{c_2 m \log m}{c_1 (\frac{1}{k} - 1)}$. Thus, n should be proportional to $m \log m$.

The above idea of isoefficiency function can be further generalized. Efficiency depends not only on the problem size, and the number of processors m, but also on other parameters of a parallel system, e.g., communication, and computation speeds. Therefore, we can define a set of *isoefficient points* in a multidimensional space of the parallel system parameters as the points for which efficiency is constant. It is convenient to think about isoefficient points, or isoefficiency lines, as of analogues of isolines in other branches of science, such as isobars, izohyets, isotherms, isodoses, etc. In order to draw a diagram of the isoefficiency lines a mathematical model linking system parameters and efficiency is needed. We present such a model in the following section.

3 Divisible Load Model

In this section we present a DLT model for a simple distributed system working in a star interconnection (cf. Fig.1). In the star interconnection the load in amount V initially resides on processor P_0 called originator. The originator sends

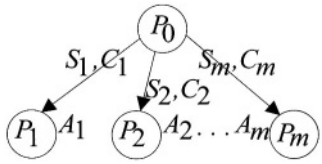


Fig. 1. Star interconnection.

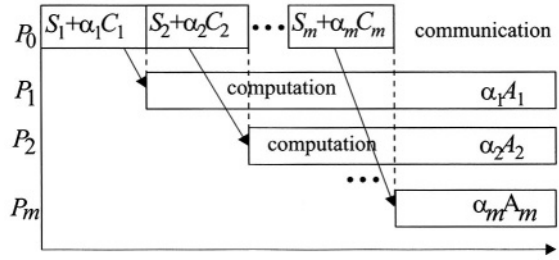


Fig. 2. Gantt chart depicting communications and computation in a star.

α_i units of load to processor P_i, for $i = 1, \ldots, m$. Processors receive their load parts one by one from P_1 to P_m, and immediately start processing the load. Messages are sent only between the originator, and the processors. Thus, the star interconnection is equivalent to a bus interconnection, or a master-slave, and client-server systems. For simplicity of presentation we assume that the originator is not computing, but is communicating only. This assumption does not limit the generality of the considerations because otherwise the computing ability of the originator can be represented as an additional processor. For simplicity of the presentation we assume that the time of returning the results is negligible. Also this assumption does not limit the generality of the study, because the process of returning the results can be easily included in the model [2,3,4,5]. The time of sending α_i units of load to P_i is $S_i + \alpha_i C_i$, where S_i is communication startup time, C_i is reciprocal of bandwidth. The time of computing on P_i is $\alpha_i A_i$, for $i = 1, \ldots, m$, where A_i is a reciprocal of computing speed. A Gantt chart depicting the communications and computations for the above described system is shown in Fig.2. It is possible to control the computation completion times by the selection of part sizes α_i. When the results are not returned the schedule is the shortest when all processors finish their computations simultaneously [2]. Consequently, the time of computation on a processor activated earlier is equal to the time of sending the load to the next processor and computing on it. Hence, it is possible to formulate the problem of finding the optimum schedule length as a set of linear equations:

$$A_i \alpha_i = S_{i+1} + \alpha_{i+1}(C_{i+1} + A_{i+1}) \text{ for } i = 1, \ldots, m-1 \tag{1}$$

$$V = \sum_{i=1}^{m} \alpha_i, \tag{2}$$

where the length of the schedule is $S_1 + (C_1 + A_1)\alpha_1$. Due to the simple structure, the above equation system can be solved for α_i in $O(m)$ time provided that a feasible solution exists. A solution of (1)-(2) does not exists when $\alpha_i < 0$ for some i. This means in practice that load size V is too small to keep all m processors working, and communication delay incurred in activating all m processors is longer than the time needed to process load V on less than m processors. Using

the solution of (1)-(2) efficiency can be calculated as

$$E = \frac{S_1 + (C_1 + A_1)V}{m(S_1 + (C_1 + A_1)\alpha_1)}. \tag{3}$$

Consequently, we obtained a method of modeling the performance and calculating the isoefficiency diagrams.

4 Performance Modeling

In this section we present preliminary results of the simulations performed on the basis of DLT model. The study we present here is limited by admissible size of the paper. For the simplicity of the presentation a homogeneous system was studied. Thus, $\forall_i A_i = A, C_i = C, S_i = S$. The isoefficiency function is depicted in 2-dimensional maps resembling weather maps, e.g., with isobars. The diagrams have two axes depicting two variable parameters, the other parameters are fixed. In the map isoefficiency lines representing points of equal efficiency are shown. The isoefficiency maps separate two areas: the area of highefficiency ($E \approx 1$), and the area of efficiency close to 0 or the equation system (1)-(2) is not solvable. In the latter case the combination of the parameters prevents activating all processors with the given load without idle periods. The area of such points is denoted as $E = 0$. Before discussing the results of the simulations let us comment on the nature of the dependencies presented in the isoefficiency maps. When an isoefficiency line is parallel to one of the axes of the map, then the efficiency changes incurred by this parameter does not influence the efficiency. It also means that one parameter cannot be compensated for by a modification of the value of the other parameter.

In Fig.3 isoefficiency map for variable m, V, and C =1, S = 1000, A = 1 is depicted. As the values of A, C, S in Fig.3 are unitless, these can be, for example, $C = 1\mu s/byte, S = 1ms, A = 1\mu s/byte$. The isoefficiency lines are not smooth which is a result of approximating the hull span on the discrete points for which the efficiency values were calculated. It can be seen in Fig.3 that when m decreases, efficiency grows. Especially for $V <$ 1E2 the increase has a character of a step function which means that only a limited number of processors can be activated for certain problem size V (moving along horizontal lines in the map). The isoefficiency lines have some slope for $V \in$[1E2,1E6]. This means that with growing m, also V should increase to exploit the parallel system efficiently. And vice versa, the number of used processors m should decrease when V is decreasing. Thus, this kind of efficiency behavior confirms observations made in the earlier publications [6]. For big problem sizes V >1E6 efficiency no longer depends on V, as the isoefficiency lines are vertical. This phenomenon can be explained in the following way: When V is very big, the ratio between V and α_1 approaches some constant $l(A, C, m) < 1$ dictated by C, A, and m. The formula (3) can be rewritten as: $E = \frac{S_1+(C_1+A_1)V}{m(S_1+(C_1+A_1)Vl(A,C,m))}$. Hence, $\lim_{V\to\infty} E = \frac{1}{ml(A,C,m)}$, and efficiency depends on m, but not on V. It can be concluded that parameters m, V are mutually related in the process of performance optimization.

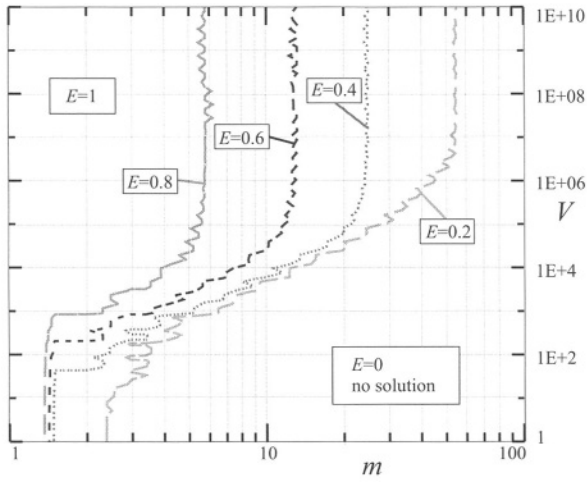


Fig. 3. Isoefficiency map as a function of m, and V.

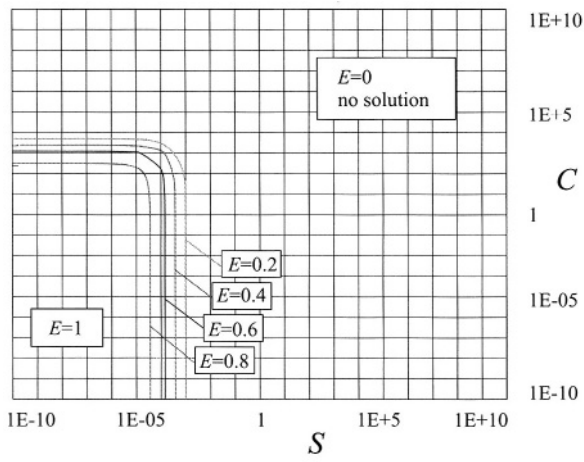


Fig. 4. Isoefficiency map as a function of C, and S.

In Fig.4 the isoefficiency relation for variable C, S, and $m = 10, V =$ 1E8, $A =$ 1E-3 is shown. The dependence of efficiency on C has a character of a step function (moving along vertical lines). For S the situation is similar (moving along horizontal lines). The isoefficiency lines have almost piece-wise linear form separating a rectangular area of high efficiency from the remaining points in the map. This rectangular form of the efficient area means that C, S are unrelated, as far as efficiency is considered. In general, the changes in parameter C (reciprocal of communication speed) cannot be compensated for by startup time S. Only in the narrow north-east corner of the efficient area, where the isoefficiency lines form a kind of a knee, can the changes in S be compensated

for by C. The increasing startup time S can be compensated for by decreasing C (i.e. increasing speed). Yet, the range of such a compensation is very limited. The rectangular efficient area determines also the methods of optimizing the performance of parallel computer systems. For example, efficiency of a parallel system with $S \approx 1\mathrm{E}{-6}$, and $C = 1\mathrm{E}{+5}$ can be improved by reducing C only, which is equivalent to increasing speed. Analogously, a system with $S \approx 1$, and $C = 1\mathrm{E}-5$ can be made efficient only by reducing the startup S. From the above discussion we conclude that parameters S, and C are, to a great extent, independent in the process of a parallel system efficiency optimization.

5 Conclusions

In this paper we proposed a new method of representing the relation between the parameters determining the performance of a parallel computer system. Simulations demonstrated that there exist parameters of parallel computer systems, such as m, V, which are mutually related. The changes of one parameter should be accompanied by the changes of the other parameter, in order to maintain constant efficiency. On the other hand there are also parameters which are independent, e.g. C, S. Future applications of the method may include analysis of the influence of a parallel system heterogeneity, memory limitations, interconnection topology, and load scattering algorithm, on the efficiency of the computations.

References

1. S.G. Akl, *The Design and Analysis of Parallel Algorithms,* Prentice-Hall Int. Inc., Englewood Cliffs, New Jersey, 1989
2. V. Bharadwaj, D. Ghose, V. Mani, T. Robertazzi, *Scheduling divisible loads in parallel and distributed systems,* IEEE Computer Society Press, Los Alamitos CA, 1996.
3. V. Bharadwaj, D. Ghose, T. Robertazzi: Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing* **6,** No.1, 2003, 7-18.
4. M. Drozdowski, *Selected problems of scheduling tasks in multiprocessor computer systems,* Series: Monographs, No.321, Poznań University of Technology Press, Poznań, (1997), (see also `http://www.cs.put.poznan.pl/~maciejd/h.ps`).
5. M. Drozdowski, P. Wolniewicz, Experiments with Scheduling Divisible Tasks in Clusters of Workstations, in: A.Bode, T.Ludwig, W.Karl, R.Wismüller (eds.), *Euro-Par 2000, LNCS 1900,* Springer-Verlag, 311-319, (2000).
6. A. Gupta, V. Kumar, Performance properties of large scale parallel systems, *Journal of Parallel and Distributed Computing* **19**, 234-244, (1993).

Gray Box Based Data Access Time Estimation for Tertiary Storage in Grid Environment

Darin Nikolow[1], Renata Słota[1], and Jacek Kitowski[1,2]

[1] Institute of Computer Science, AGH-UST, al.Mickiewicza 30, Cracow, Poland
[2] Academic Computer Center CYFRONET AGH, ul.Nawojki 11, Cracow, Poland

Abstract. In this paper experiments on estimation of data access time for data located on tertiary storage systems are presented. Estimation algorithm based on the gray-box approach using event driven simulation method is described. The implementation of such a system for the Legato DiskXtender HSM system is tested and results are shown. Application of the system for grid environment is discussed.

1 Introduction

The evolution of the Grid computational technology has made possible execution of very large scale scientific applications. They often deal with large volume of data which cannot be stored and processed locally. Therefore data grids, like European Data Grid (EDG) and CrossGrid (CG) [1,2] have been initiated.

One of the issues in data grid research is the problem of optimization of data access. Data replication technique is used to speedup the access to geographically distributed data. For example studies focusing on constructing of scalable replica location services are in progress [3]. Another challenge is choosing the best replica [4] for data needed by a grid application. The optimization of usage of local resources, which can hold different kinds of data and deal with different types of storage systems installed locally, is the main topic in [5].

Replication algorithms base on access characteristics of storage and network resources. Obtaining and using the dynamic access characteristic is challenging, due to wide variety of storage equipment. Selecting the replica location is based on cost estimation for delivering the data to the grid user. The cost consists of the network cost and the storage element cost. While the former problem is currently being solved [1], the latter one still lacks sufficient solution.

One of the most sophisticated storage system are the Hierarchical Storage Management (HSM) systems, which are often used as parts of the Grid, to economically store massive data. These systems are mainly based on some kind of tertiary storage, like tape libraries and optical jukeboxes. The main idea of HSM systems is to use low-cost, slow media (tapes) to store less popular data and more expensive, fast access media (hard disk) for the most popular data. Data can be located on any level of the storage hierarchy. It causes data access time variation for HSM systems: from a few seconds to tens of minutes. This mainly depends on the system load at the time of making a request, but other

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 181–188, 2004.
© Springer-Verlag Berlin Heidelberg 2004

parameters like location of the needed data on the storage medium, transfer rates, and seek times of the drives are also important.

In [6] a real storage system implementation with access time estimation functionality is shown for the transfer time rather than the latency startup time estimation. In [7] a performance prediction method for Storage Resource Broker (SRB) system is described based on time measurements of basic SRB operations assuming that the file is already in the disk cache. No estimating the staging time is discussed.

The purpose of this paper is to discuss experiments with estimation of data access time for grid data located on HSM systems. Because the focus of our study are HSMs the network overhead time is not taken into account and the data access is characterized by two parameters only – the startup latency time and the transfer rate of the HSM. The reason for the study is to develop a data access estimation system for tertiary storage to be included into the cost estimation modules in CG and EDG.

The paper is organized as follows. Next section addresses the problem of data access time estimation for HSM and describes the estimation algorithm. Experimental results for the Legato DiskXtender HSM system are presented in the third section. The last section concludes the paper.

2 Data Access Time Estimation for HSM Systems

The method presented in this paper for estimating of access time to data residing on HSM systems is based on the gray-box approach. We consider the HSM system as a gray-box if we have some knowledge available about its internal algorithms or some information about its current state can be provided by the system's diagnostic tools or interfaces. Therefore, the estimation algorithm depends to a certain extend, on the given HSM system. The algorithm presented below is based on partial knowledge (gathered mainly by observing the system behavior) of the internal algorithm of the Legato DiskXtender HSM system. The Legato DiskXtender HSM system offers commercial quality and therefore is popular in computer centers. Its drawback is high latency time (including mounting, seek and transfer times), because before delivering data to the user the whole data is to be introduced into the HSM disk cache [8].

The general architecture of such system has been proposed in our previous work [9] and consists of three main modules: Monitor, Simulator and Request Monitor & Proxy. The implementation details of the previous version of the system for the Legato DiskXtender HSM have been presented in [10].

2.1 Estimation Algorithm

The Simulator is based on event driven simulation. Initially, the simulator adds appropriate events to its event queue, depending on the current state of the model. For example, if there is a request in the queue and the needed tape has not been yet mounted, then an event informing that the tape is mounted, is

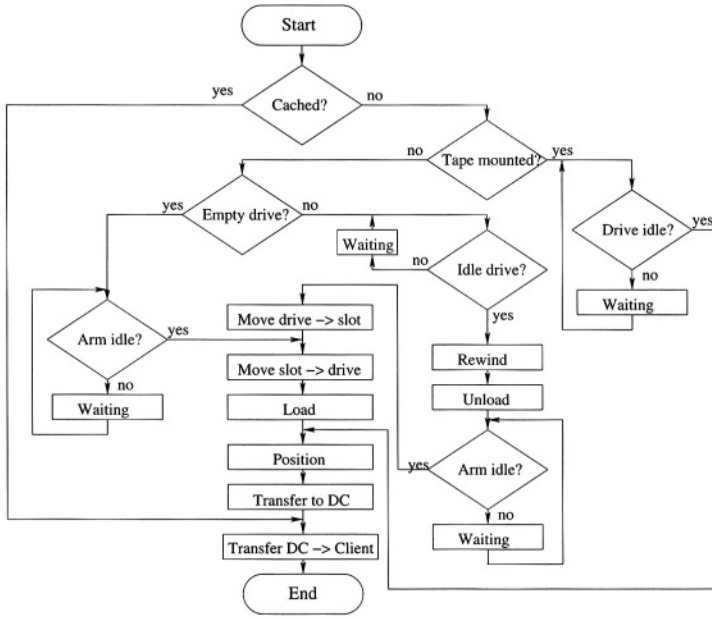


Fig. 1. HSM request serving algorithm.

scheduled. Every time when the simulator is changing the state of the model none or one further event is scheduled and added to the event queue. The state is changed when an event occurs. When the event occurs, the model clock is increased with the event estimation time. If we are interested in ETA for a particular file and the event "the file is transferred to the disk cache" has just occurred, the simulation is stopped.

A typical algorithm used by the HSM to serve data access requests is presented in Fig. 1. To complete a request the algorithm goes through different stages, defined on the basis of the atomic tertiary storage operations. The stages, which can be identified during retrieving a file residing on tape(s) from the HSM system, are described in Table 1. The duration of some stages can be substituted by a constant value, which is obtained from the device technical specification or derived experimentally to achieve more adequate values. For some other stages the duration cannot be kept constant and is influenced by some additional data dependent parameters, like block numbers of current and target position, file size or data compression ratio. In these cases appropriate models should be used.

One of the most complex problems is modeling of the positioning time for serpentine tape drives. There are several studies on access modeling to tape drives. The low cost model [11] seems to be the most interesting one, with eight different seek scenarios (classes) depending on the work to be done by the drive proposed. For each class an analytical cost function is provided. To use the model, each type of drive used in the HSM system has to be parameterized and

the tapes have to be characterized with the block address of the beginning of each track obtained experimentally. This is necessary since these addresses can differ for each tape, because of bad blocks and different compression ratio. This seek time model has been adopted into our estimation system.

Table 1. Stages in serving a request by HSM System.

Stage	Description	Duration Estimation
Wait	Waiting for resources (robot arm, drive or tape)	Simulated
Rewind	Rewinding of an idle tape	Modeled based on current tape block position
Unload	Preparing the tape for ejecting	Constant
Move drive→slot	Moving idle tape from drive to slot	Constant
Move slot→drive	Moving required tape from slot to drive	Constant
Load	Bringing the tape on-line	Constant
Position	Positioning the tape to the first block of file being accessed	Modeled based on the current and target position
Transfer to DC	Transferring data from tape to Disk Cache	Modeled based on transfer rate and file size
Transfer DC→Client	Transferring data from Disk Cache to Client	Modeled based on disk rate, number of simultaneous requests and file size

3 Experimental Results

3.1 Testbed Configuration

The experiments were performed at the ACC CYFRONET-AGH in Cracow, where ATL2640 and ATL7100 automated tape libraries with 3 and 4 DLT7000 drives respectively and HP660ex magneto-optical jukebox with 4 MOD (Magneto-Optical Disk) are installed. For these tape libraries constant durations of the stages mentioned in Table 1 are the following: *Load* - 37*s*, *Unload* - 17*s*, *Move drive→slot* - 15*s*, *Move slot→drive* - 14*s*. For the duration estimation of stages *Transfer to DC* and *Transfer DC→Client* the transfer rates of 5*MB/s* and 2*MB/s* have been used respectively.

The tertiary storage hardware is managed by the DiskXtender HSM software licensed for STB and running distributed on two HP9000 servers. The main server is the HP K-class machine with four processors managing DiskXtender file databases, disk cache, MOD jukebox and ATL 2640 library. The second one is the HP D-class server with ATL7100 tape library attached.

During the tests only requests for files residing on the library connected to the main server have been issued.

3.2 Results

In general, three types of tests have been done: single request tests and multiple requests tests on the idle system, and single request test on the generally busy system. The latter case is more realistic for standard HSM operations and is similar to the multiple requests tests. The difference is that requests in the queue are made not only by the testing program but by other users as well. The third type of tests has been chosen further discussion.

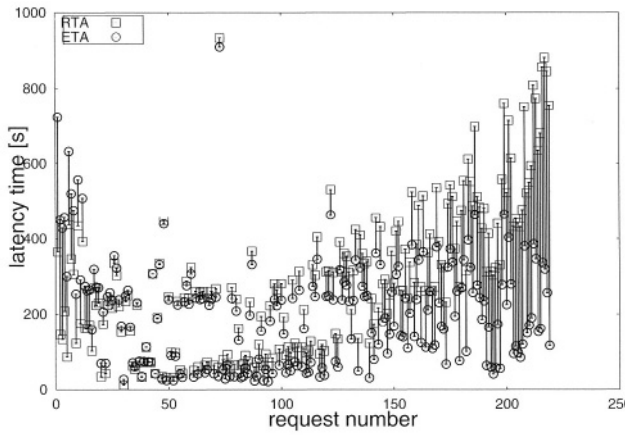


Fig. 2. Single request tests on a generally busy system.

In Fig. 2 the latency time for different requests is presented. The real and estimated times of arrival (RTA and ETA respectively) for each request are shown and the points are connected in order to have a visual representation of the absolute "error", defined as $\Delta = RTA - ETA$. The requests have been pre-sorted according to increase of Δ for better visualization of the small error points.

Next, in Figs. 3 and 4 histograms for the number of requests falling into a given error interval are presented. The Δ interval has been set to $50s$ while the interval of the relative "error", $\delta = \Delta/RTA * 100\%$, at 5% has been fixed. 45% of the requests have $\Delta \in [-50s, 50s]$.

In order to identify the source of discrepancy a deeper look at the estimation times has been taken and additional single request tests on idle system have been performed. Both the real and estimated latency times are broken into mounting time, seek time and transfer time elements. The mounting time represents the interval between the moment of issuing a request and the moment of having the required tape on line (this includes all stages listed before the *Position* stage in Table 1). The seek and transfer times represent the duration of the *Position* and *Transfer* stages respectively. In Table 2 we see that the transfer time is

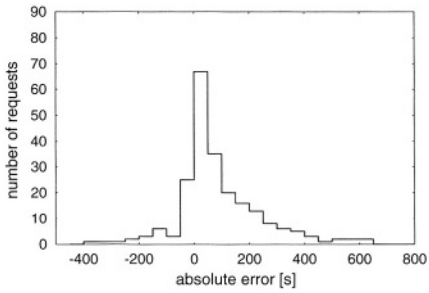


Fig. 3. Absolute "error" histogram.

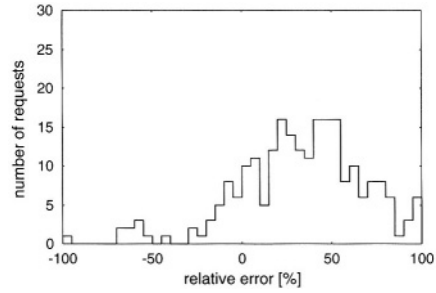


Fig. 4. Relative "error" histogram.

estimated with accepted accuracy. The real mounting time, which is supposed to be constant, deviates from the constant value used in the simulation.

Table 2. Detailed single request tests results for tapes.

No	Real mounting time [s]	Real seek time [s]	Real transfer time [s]	RTA [s]	Estimated mounting time [s]	Estimated seek time [s]	Estimated transfer time [s]	ETA [s]
0	55	52	71	180	170	31	62	264
1	187	174	4	365	170	32	1	203
2	180	62	11	253	170	47	17	234
3	159	21	1	181	170	32	1	203
4	173	31	79	283	170	27	82	279
5	163	92	1	256	170	32	2	204
6	188	96	2	286	170	32	1	203
7	159	73	5	237	170	63	5	238
8	196	75	7	278	170	79	5	254

The mounting time errors are probably due to the distributed configuration of the DiskXtender installation in ACC CYFRONET AGH. This could be also caused by the contamination of tapes and drives. Another reason for the high errors could be the fact that the drives and robots parameters have been taken from the vendor specification without taking into account the software overhead. Additional measurements should be done "on-site" to increase the accuracy.

From Table 2 it follows, that the main source of inaccuracy is the seek time estimation. For gathering more statistical information about the seek time estimation errors a comparison between measured and estimated latencies for files residing on the same tape is made (see Fig. 5). In this test seek and transfer operations are performed only since the tape is forced to stay mounted continuously. The requested files are short (<32MB) and contain compressed data which results in small contribution of transfer time (<10%) to the measured

latency, mainly influenced by the seek operation in this case. In Fig. 5 two series of measurements for the same request sequence are compared with the estimated latency. No essential differences between series have been observed. The measured and estimated characteristics are similar, although the values are underestimated for many requests.

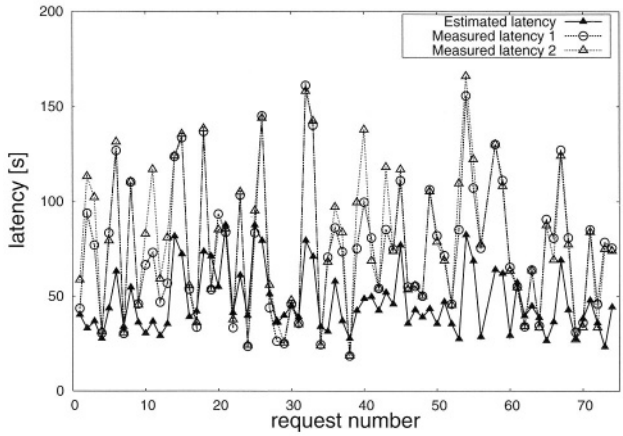


Fig. 5. Comparison between estimated and measured latencies.

4 Conclusions

We have focused on estimation of the access time for the HSM system, which will allow for more detailed operations of replica Grid components and better utilization of the Grid storage resources. Based on the previously proposed gray-box approach we have implemented an estimation system for the Legato DiskXtender HSM system. The tests have shown that the exact estimation of the HSM system access time is difficult (with the seek time estimation resulting as a main source of errors), but still, estimation is feasible for a certain tolerance level.

The estimation system is currently being tested as part of the replica manager module for the CG and EDG projects [12]. It will extend EDG getSECosts( ) functionality for other tape systems, including EDG Castor in the close future.

Our further efforts will concentrate on tuning of the system, improving the positioning time model for DLT tapes. Some additional functionalities like self tuning are also considered for implementation in the next versions.

Acknowledgments. The work described in this paper was supported in part by the European Union through the IST-2001-32243 project "CrossGrid" and by the Polish Committee for Scientific Research (KBN) projects: 112/E-356/SPB/5.PR UE/DZ 224/2002-2004 and 4 T11C 028 24. ACC CYFRONET-AGH is acknowledged.

References

1. "DataGrid – Research and Technological Development for an International Data Grid", EU Project IST-2000-25182.
2. "CROSSGRID – Development of Grid Environment for Interactive Applications", EU Project IST-2001-32243.
3. Chervenak, A.L., et al. "Giggle: A Framework for Constructing Scalable Replica Location Services", in: Proc. of the 3rd Int'l. IEEE Supercomputing Conf., Baltimore, USA, Nov. 2002.
4. Bell, W., Cameron, D., Capozza, L., Millar, P., Stockinger, K., and Zini, F., "Simulation of Dynamic Grid Replication Strategies in OptorSim", in: Parashar, M., (Ed.), Proc. of Third Int. Workshop, Grid Computing – GRID 2002, Nov.18, 2002, Baltimore, USA, Lect. Notes in Comput. Sci., vol.2536, pp. 46-57, Springer, 2002.
5. Dutka, Ł., Słota, R., Nikolow, D., Kitowski, J., "Optimization of Data Access for Grid Environment", presented at lst European Across Grids Conference, Universidad de Santiago de Compostela, Spain, February 13-14, 2003.
6. Bernardo, L., M., Shoshani, A., Sim, A., "Access Coordination of Tertiary Storage for High Energy Physics Application", in: Proc. of the 8th IEEE Symposium on Mass Storage Systems, University of Maryland, College Park, Maryland, USA, March 27-30, 2000.
7. Shen, X., Liao, W., Choudhary, A., "Remote I/O Optimization and Evaluation for Tertiary Storage Systems through Storage Resource Broker", in: Proc. of IASTED Applied Informatics Conference,Innsbruck, Austria, February, 2001.
8. Nikolow, D., Slota, R., Kitowski, J., Nyczyk, P., Otfinowski, J., "Tertiary Storage System for Index-Based Retrieving of Video Seqences", in: Hertberger, B., Hoekstra, B., Williams, R. (Eds.), Proc. Int. Conf. High Perform. Comput. and Networking, Amsterdam, June 25-27, 2001, Lect. Notes in Comput. Sci. vol.2110, pp. 62-71, Springer, 2001.
9. Nikolow, D., Słota, R., Dziewierz, M., Kitowski, J.,"Access Time Estimation for Tertiary Storage Systems", in: Monien, B., Feldman, R. (Eds.), Proc. 8th Int. Euro-Par 2002 Parallel Processing Conf., August 27-30, Paderborn, Germany, Lect. Notes in Comput. Sci., vol.2400, pp. 873-880, Springer, 2002.
10. Nikolow,D., Słota, R., Kitowski, J. "Data Access Time Estimation for HSM Systems in Grid Environment", in: Proc. of the Cracow Grid Workshop, Cracow, Poland, December 11-14, pp.209-216, ACK Cyfronet-AGH, Cracow, 2002.
11. Sandstå, O., Midstraum, R., "Low-Cost Access Time Model for Serpentine Tape Drives", in Proc. of 16th IEEE Symp. on Mass Storage Systems, the 7th NASA Goddard Conf. on Mass Storage Systems and Technologies, San Diego, California, USA, March 1999, pp. 116-127.
12. Stockinger, K., Stokinger, H., Dutka, L., Slota, R., Nikolow, D., Kitowski, J., "Access Cost Estimation for Unified Grid Storage Systems", Supercomputing 2003 Conf., Phoenix, Arizona, accepted.

Performance Modeling of Parallel FEM Computations on Clusters

Tomasz Olas, Roman Wyrzykowski, Adam Tomas, and Konrad Karczewski

Czestochowa University of Technology
Institute of Computer & Information Sciences
Dabrowskiego 73, 42-200 Czestochowa, Poland
{olas, roman, adam.tomas, xeno}@icis.pcz.pl
http://icis.pcz.pl

Abstract. *ParallelNuscaS* is an object-oriented package for the parallel finite element modeling, developed at Czestochowa University of Technology. This paper is devoted to modeling the performance of the package on PC-clusters. Such modeling allows for analyzing and predicting the performance of this complex scientific application on different computing platforms. Because both uniprocessor and SMP nodes are considered, we investigate not only the message-passing model of parallel programming, but also the hybrid model, which is a mixture of multithreading inside SMP nodes and message passing between them.

1 Introduction

Finite element method (FEM) is a powerful tool for study different phenomena in various areas. However, many applications of this method have too large computational or memory costs for a sequential implementation. Parallel computing allows this bottleneck to be overpassed [16,18].

The fast development of the processing power of high-end PCs together with the availability of open source software such as Linux have made it possible to build very cost efficient parallel computers. With the addition of high bandwidth and low latency local networks, PC-clusters [4] are the most common and available parallel systems now.

Based on the object-oriented technique, an environment for the parallel FEM modeling on clusters was developed [8] at Czestochowa University of Technology. This environment is an extension of the sequential *NuscaS* software [12].

When incorporating cluster computing into *NuscaS,* the basic assumption was to provide transparency of mechanisms supporting parallelism for developers of new application modules. For this aim, these mechanisms (communication between processors and allocation of mesh nodes to processors) are embedded into a parallel kernel of the environment. Their use is accomplished through such object-oriented techniques as inheritance and class templates. As a result, a suitable parallel code will be generated, depending on a model of memory access in the target parallel architecture Another important design solution is to use iterative methods [11] for solving large linear systems with sparse matrices,

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 189–200, 2004.
© Springer-Verlag Berlin Heidelberg 2004

which arise as a result of FEM discretization. These methods allow the sparsity of matrices to be exploited more efficiently than direct methods.

Predicting and analyzing the performance of FEM computations on clusters is not trivial. Performance modeling using either analytic or simulation models provide a solution to this problem.

Analytic models [17] are potentially most accurate, but they easily become unsolvable due to the complexity of the simulated system, and usually show a limited flexibility [13]. Simulation models allow for the study of very complex systems. Their versatility makes it possible to estimate the performance of hardware and software architecture during the various phase of their development. Hybrid models combining analytic and simulation techniques can provide the most powerful solution.

This paper is devoted to modeling the performance of *ParallelNuscaS* on PC-clusters. Such modeling allows for analyzing and predicting the performance of this complex scientific application on different computing platforms. Because both uniprocessor and SMP nodes are considered, we investigate not only the message-passing model of parallel programming, but also the hybrid model, which is a mixture of multithreading inside SMP nodes and message passing between them.

2 Performance Model of Parallel FEM Computation for Clusters with Uniprocessor Nodes

For uniprocessor nodes the message-passing model is used in *ParallelNuscaS*. In this case, a FEM mesh is decomposed [5] into a number of submeshes (or subdomains) such that the computational load per submesh is approximately the same; these subdomains are then processed concurrently over different processors, whose number p is equal to the number of submeshes.

The core of FEM computations is solving large linear systems with sparse matrices [18]. In *ParallelNuscaS,* the Conjugate Gradient (CG) iterative algorithm with Jacobi preconditioner is used at present [3]. A single iteration of this algorithm is illustrated in Fig.1. The kernel of this algorithm is the sparse matrix-vector multiplication. When implementing this operation, the overlapping of computation and communication is exploited to reduce the algorithm execution time.

According to the scheme shown in Fig.1, the total execution time t for one iteration of the CG algorithm can be expressed as follows:

$$t = t_{comp_initial} + t_{comm} + t_{mul} + t_{comp_final} = t_{comp}^{initial+final} + t_{comm} + t_{mul}, \quad (1)$$

where

- t_{mul} - time of matrix-vector multiplication,
- t_{comm} - time of communication during one iteration of the CG algorithm.

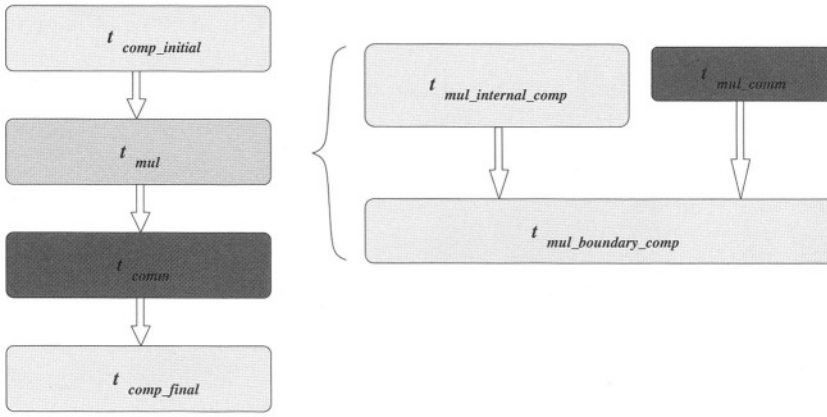


Fig. 1. Computations in one iteration of the CG algorithm

2.1 Time Model for $t_{comp}^{initial+final}$

This term in formula (1) is estimated [14] using the following expression:

$$t_{comp}^{initial+final} = n_{flops} t_f, \tag{2}$$

where:

- n_{flops} is the number of floating point operations,
- t_f is the average time of one floating point operation.

Table 1. Values of t_f for different PC architectures

Architecture	$t_f[ns]$
Intel Pentium III 700MHz	31.6
AMD Athlon 1200MHz	18.7

The parameter t_f was measured for different PC architectures, using the sequential CG algorithm with a known number of floating point operations. The results of measurement are shown in Table 1.

When estimating the number of operations for various stages of the CG algorithm, we assume an uniform assignment of mesh nodes to processors. So in average for each subdomain, we have:

- n^i of *internal nodes* - they are coupled only with nodes belonging to this subdomain,
- n^b of *boundary nodes* - they correspond to those nodes in the given subdomain that are coupled with nodes of other subdomains,
- n^e of *external nodes* - they correspond to those nodes in other subdomains that are coupled with the boundary nodes of the given subdomain.

Internal and boundary nodes are called local nodes, their number n^l is given by $n^l = n^i + n^b$.

Based on these assumptions, we obtain finally:

$$t_{comp}^{initial+final} = (11n^l + 4n + 10)t_f, \tag{3}$$

where $n = n^i + n^b + n^e$.

2.2 Communication Time

The standard model is used to estimate the time T_{comm} necessary to transfer a message containing w elements [1]:

$$T_{comm} = \alpha + \beta w, \tag{4}$$

where:

- α - start-up time,
- β - per-word transfer time,
- w - number of transferred words.

Based on this expression, we obtain the following formula for the term t_{comm} in equation (1):

$$t_{comm} = 2p(\alpha + 3\beta) \tag{5}$$

The parameters α and β were measured for different interconnects used in our ACCORD cluster [9,10]. For this aim, we measured the time $T_{comm}^{\star}$ necessary to transfer a message between two processors (there and back), so $T_{comm} = \frac{1}{2}T_{comm}^{\star}$. The results of measurement are shown in Table 2. Note that values of the parameter β depend on w. In accordance with formula (5), the value of β placed in Table 2 corresponds to transfer of three double precision numbers (8 bytes each).

Table 2. Values of α, β for different cluster architectures

Cluster architecture	$\alpha[\mu s]$	$\beta[\frac{\mu s}{w}]$
ACCORD Intel FastEthernet	69.73	9.79
ACCORD AMD FastEthernet	69.2	7.22
ACCORD Intel Myrinet	14	3.35

2.3 Time of Matrix-Vector Multiplication

According to Fig.1, we have:

$$t_{mul} = max(t_{mul_internal_comp}, t_{mul_comm}) + t_{mul_boundary_comp}, \tag{6}$$

where:

- $t_{mul_internal_comp}$ - time spent for computation involving only internal nodes assigned to a given processor (see Fig. 2),
- $t_{mul_boundary_comp}$ - time spent for computation involving boundary nodes assigned to this processor (Fig.2),
- t_{mul_comm} - time necessary for communication with neighbour processors.

Let us nz denotes the number of nonzero elements in the matrix **A** (see Fig.2). Then the number of nonzero elements in the internal and boundary parts of **A**-matrix (Fig.2) can be derived from the following expressions:

$$nz^i = \frac{nz}{n^i}, \quad nz^b = \frac{nz}{n^b}. \tag{7}$$

Assuming an uniform assignment of mesh nodes to processors, we obtain:

$$t_{mul_internal_comp} = 2nz^i t_f^\star, \tag{8}$$

$$t_{mul_boundary_comp} = 2nz^b t_f^\star. \tag{9}$$

The communication time t_{mul_comm} is the sum of the time $t_{mul_send_comm}$ necessary to send boundary components of a vector being updated in a given processor to its neighbours, and the time $t_{mul_receive_comm}$ necessary to receive vector components corresponding to external nodes

$$t_{mul_comm} = t_{mul_send_comm} + t_{mul_receive_comm}, \tag{10}$$

where

$$t_{mul_send_comm} = s(\alpha + \frac{n^b}{s}\beta) = a\alpha + n^b\beta, \tag{11}$$

$$t_{mul_receive_comm} = s(\alpha + \frac{n^e}{s}\beta) = a\alpha + n^e\beta. \tag{12}$$

Here s denotes the number of neighbour subdomains for a given subdomain.

After substitution of expressions (11) and (12) to equation (10), we obtain:

$$t_{mul_comm} = 2s\alpha + (n^e + n^b)\beta, \tag{13}$$

and finally from formula (6) we have:

$$t_{mul} = max(2nz^i t_f^\star, 2s\alpha + (n^e + n^b)\beta) + 2nz^b t_f^\star, \tag{14}$$

2.4 Final Formula

After substituting expressions (3), (5) and (14) to equation (1), we have:

$$t = \Big((11n^l + 4n + 10)t_f + 2p(\alpha + 3\beta) + max(2s\alpha + (n^e + n^b)\beta, 2nz^i t_f^\star) + 2nz^b t_f^\star.\Big) \tag{15}$$

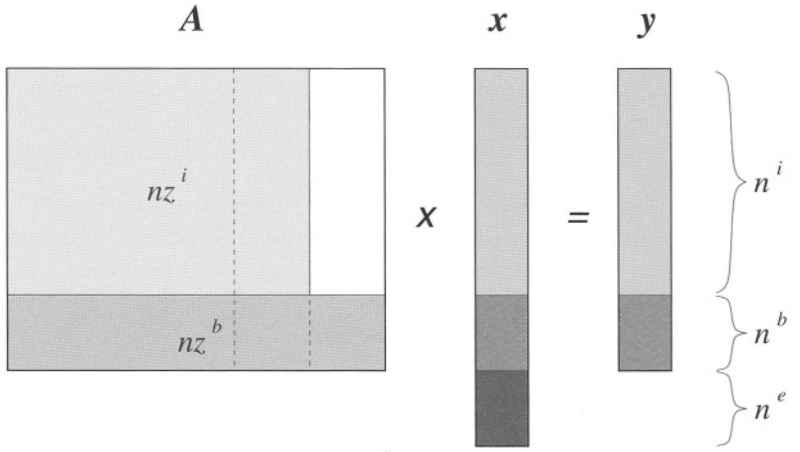


Fig. 2. Matrix-vector multiplication

When the communication network in a cluster possesses a sufficiently high bandwidth, for large systems of equations the communication time t_{mul_comm} is smaller than the computation time $t_{mul_internal_comp}$. It has been verified experimentally on our ACCORD cluster that even for Fast Ethernet this assumption is satisfied for processing more than several thousand mesh nodes on a processor. Hence we obtain finally:

$$t = (11n^l + 4n + 10)t_f + 2nzt_f^\star + 2p(\alpha + 3\beta). \tag{16}$$

3 Validation of Performance Model and Performance Prediction

To validate the performance model given by formula (16), we use the ACCORD cluster – Academic Cluster of Czestochowa fOr Research and eDucation, built in the Institute of Computer and Information Sciences, Czestochowa University of Technology [9,10].

The first version of the ACCORD cluster was built on January 2001, as the second high-performance cluster in Poland. It contains 18 Pentium III 750 MHz processors, or 9 Intel server platforms ISP 2150G as SMP nodes. Each node has 512 MB RAM, except for the front-end node with 1 GB RAM. For tightly coupled distributed computations, the interconnect must provide high-data-rate and low-latency communication between nodes. That is why, the Myrinet network [7] is used in the cluster, for the first time in Poland. In ACCORD, the measured value of latency (start-up time in formula (4)), obtained for MPICH over GM implementation of MPI, is 14 microseconds. The Myrinet network in ACCORD is built around M2M-SW16 switch of the second generation. It is a cut-through, crossbar switch with 16 SAN ports. All the nodes in the cluster have Myrinet SAN interfaces of the third generation.

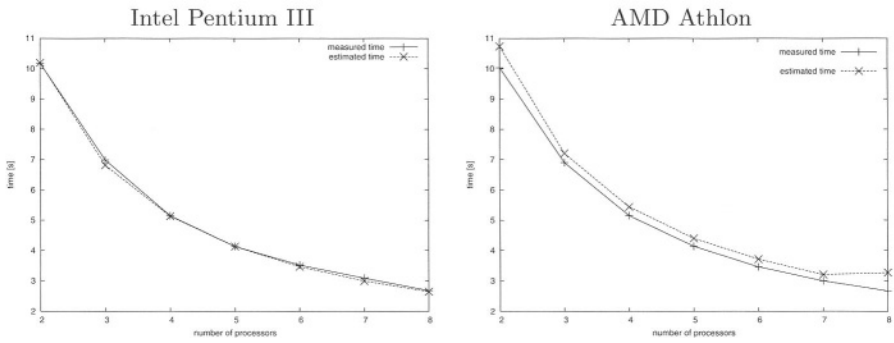


Fig. 3. Validation of performance model: Pentium III vs Athlon MP processors, Fast Ethernet interconnect

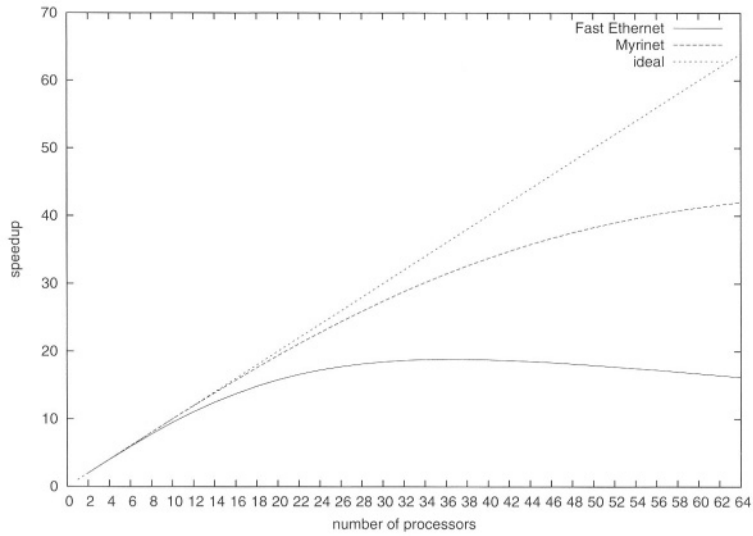


Fig. 4. Performance prediction for Fast Ethernet and Myrinet interconnects, Pentium III processors

For 32-bit architectures, a cost-effective alternative to Intel processors is the AMD Athlon architecture with its multiprocessor support for data-movement-intensive applications. That is why, the ACCORD cluster was firstly upgraded with 16 AMD Athlon MP 1.2 GHz processors, and then with 16 AMD Athlon MP 2100+ processors. At the moment, these 16 SMP nodes based on Athlon MP processors have 512 MB DDRAM memory each. They are connected by Fast Ethernet, which provides the peak performance of only 100 Mb/s. ACCORD is operating under the control of the LINUX system (Debian distribution) and Condor workload management system. The MPICH over GM and LAM implementations of MPI [2] are used as basic programming tools. It is supported by the Vampir/Vampirtrace packages for the performance monitoring and analysis of parallel programs.

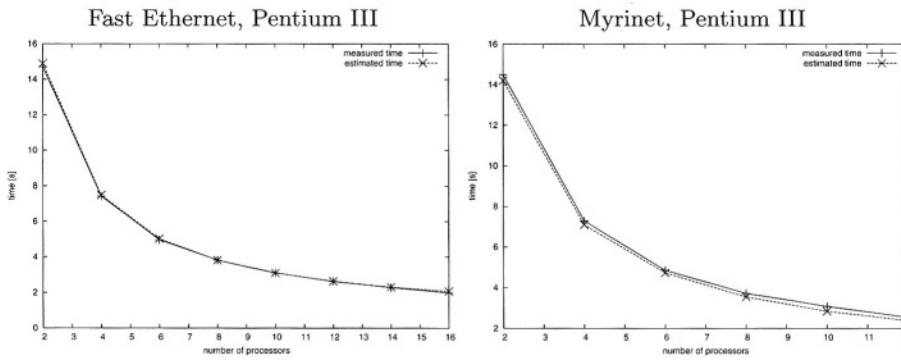


Fig. 5. Validation of performance model for pure message passing on SMP nodes: Fast Ethernet vs Myrinet

Some results of tests performed on the ACCORD cluster for a mesh containing approximately 250 000 nodes are presented in Fig.3, which corresponds to Fast Ethernet. The results of tests confirm a good accuracy of our model, especially in case of Pentium III processors. The similar conclusion is true for Myrinet.

Assuming the number of nodes greater than available in the ACCORD cluster, this performance model can be used to predict the scalability of our package, for different communication networks. Fig.4 presents results of such a performance prediction. These results confirm a much better scalability of Myrinet against Fast Ethernet. Considering the number of processors in the range from $p = 1$ to $p = 64$, the maximum speedup for Myrinet is $S_p = S_{64} \approx 41.5$, while for Fast Ethernet the maximum speedup $S_p \approx 18.5$ is practically obtained for $p = 32$ processors, and beginning from $p = 36$ the speedup is even decreasing.

4 Modification of Performance Model for Clusters with SMP Nodes

4.1 Pure Message-Passing Model

For SMP nodes, the implementation of the pure message-passing model is similar to that for uniprocessor nodes. It is only necessary to configure an MPI library in a way providing as many MPI processes on a SMP node as the number of processors in the node.

The performance model for this case can be easily obtained by reinterpretation of formula (16). It is sufficient to replace the number p of processors in formula (16) by the number of nodes, and use new values of times t_f and $t_f^\star$, measured while running the code on one SMP node. The results of tests (see Fig.5) confirm a good accuracy of such a model.

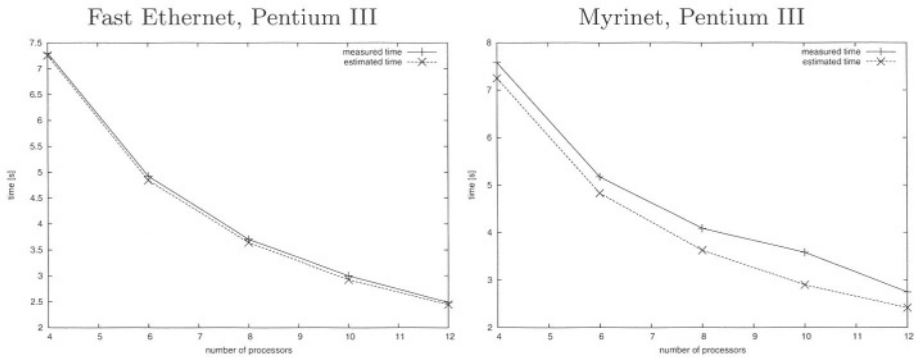


Fig. 6. Validation of performance model for block decomposition method: Fast Ethernet vs Myrinet

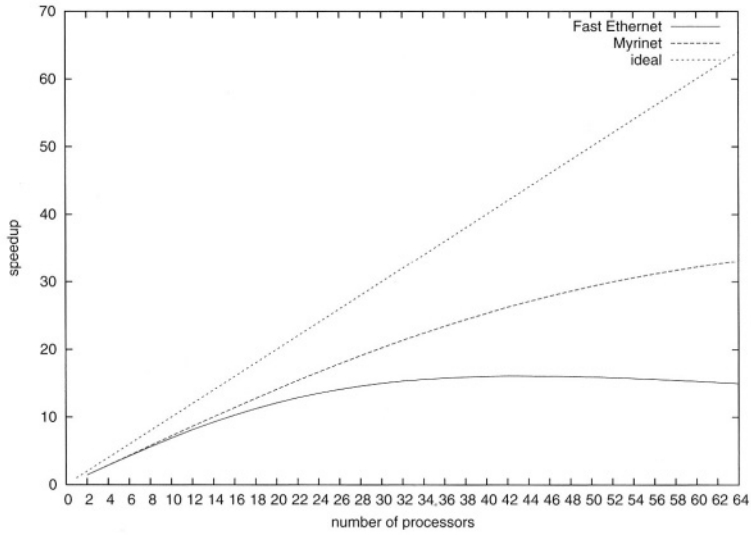


Fig. 7. Performance prediction for pure message passing on SMP nodes: Fast Ethernet vs Myrinet

4.2 Hybrid Model

The hybrid model of parallel programming seems to be the best choice for achieving high performance on clusters of SMPs [15]. It assumes a mixture of shared-memory (or multithreaded) programming inside SMP nodes, and message passing between them. The POSIX threads (shortly Pthreads) library [6] has been chosen for implementing the SMP extension in *ParallelNuscaS*

Among several methods of mapping FEM computations into a target SMP node running Pthreads, considered [9,10] for implementation in *ParallelNuscaS,* only the block-decomposition is fully implemented and verified at this moment.

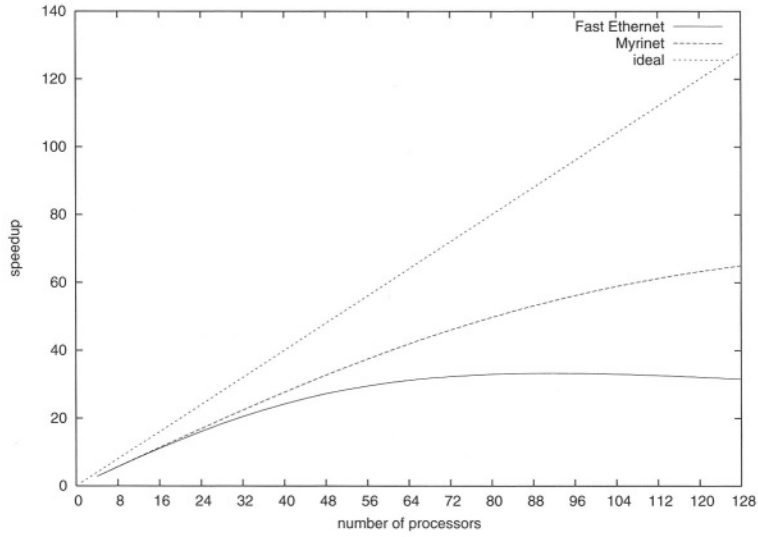


Fig. 8. Performance prediction for block-decomposition method: Fast Ethernet vs Myrinet

In this method, a certain thread operates on a block of $]n_j^i/p_n[$ contiguous rows of the local matrix $\underline{\mathrm{A}}$ corresponding to the internal nodes of a subdomain (see Fig.2), as well as a block of $]n^b/p_n[$ contiguous rows of $\boldsymbol{A}$ corresponding to the boundary nodes. Here p_n is the number of processors in a SMP node, and $]x[$ denotes the integer greater or equal to x. The same rule is observed for elements of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ (see Fig.2). These data are then used by the thread to build and solve a local system of equations. The speedup obtained when using this method on a two-way SMP node is shown in Table 3.

Table 3. Speedup of block decomposition on a two-way SMP node

Architecture	N=40613	N=80401	N=249925	N=501001
Intel Pentium III	1.38	1.36	1.36	1.36
AMD Athlon MP	1.63	1.51	1.57	1.57
AMD Opteron	1.32	1.41	1.28	1.28

As in case of the pure message-passing model, we modify expression (16) to obtain the performance model for the block-decomposition method. For this aim, besides modifications described in Section 4.1, it is necessary to change the last term of formula (16), which corresponds to communication overheads during one iteration of the CG algorithm, in the following way:

$$t_{comm} = 2\frac{p}{r}(\alpha + 3\beta). \tag{17}$$

This modification is a result of reducing these overheads proportionally to the number r of processors in a node.

Fig. 6 shows some results of tests performed on the ACCORD cluster. They indicate a certain, fortunately relatively small discrepancy between the model and reality. This discrepancy is a result of neglecting overheads arising from the synchronization of threads inside a node.

4.3 Performance Prediction

As in case of uniprocessor nodes, the obtained performance models allow for prediction of the scalability of the code for the number of nodes greater than available in the ACCORD cluster. Fig.7 and Fig.8 present results of such a prediction. These results indicate a better scalability of the hybrid model using the block decomposition than the pure message-passing model.

References

1. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing (second edition). Pearson Education Ltd., 2003.
2. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface. MIT Press, Cambridge MA, 1995.
3. Meisel, M., Meyer, A.: Hierarchically Preconditioned Parallel CG-Solvers with and without Coarse-Mesh Solvers Inside. SFB399-Preprint 97-20, Technische Universitat Chemnitz, 1997.
4. High Performance Cluster Computing, vol.1,2. R. Buya, ed., Prentice-Hall, 1999.
5. Karypis, G., Kumar, V.: METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Tech. Report, Dept. of Computer Science and Engineering, Univ. of Minnesota, 1998
6. Lewis, P., Berg, D.J.: Multithreaded Programming with Pthreads. Sun Microsystems Press, 1998.
7. Myrinet Home Page: http://www.myri.com
8. Olas, T.: Solving Application Finite Element Modeling Problems in Parallel using *NuscaS* Software. Proc. Int. Workshop on Parallel Numerics - ParNum 2000, Bratislava, Slovakia, pp. 125-143.
9. Olas, T., Karczewski, K., Tomas, A., Wyrzykowski, R: FEM Computations on Clusters Using Different Models of Parallel Programming. Lect. Notes in Comp. Sci. **2328** (2002) 170-182.
10. Olas, T., Lacinski, L., Karczewski, K., Tomas, A., Wyrzykowski, R.: Performance of Different Communication Mechanisms for FEM Computations on PC-Based Clusters with SMP Nodes. Proc. Int. Conf. on Parallel Computing in Electrical Engineering - PARALEC 2002, Warsaw, IEEE Computer Society, 2002, pp.305-311.
11. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing, New York, 1995.
12. Sczygiol, N.: Object-Oriented Analysis of the Numerical Modeling of Castings Solidification. Computer Assisted Mechanics and Engineering Science **8** (2001) 79-98.
13. Spinnato, P.F., van Albada, G.D., Sloot, P.: Performance Modeling of Distributed Hybrid Architectures. IEEE Trans. Parallel and Distributed Systems **15** (1) (2004) 81-92.

14. Taufer, M., Stricker, T., Roos, G., Gunter, P.: On the Migration of the Scientific Code Dyana from SMPs to Clusters of PCs and on the Grid. Proc. 2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, 21-24-May 2002, Berlin, IEEE Computer Society, 2002, pp.93-101.
15. Tanaka, Y., Matsuda, M., Kubota, K., Sato, M.: COMPaS: A Pentium Pro PC-based SMP cluster. In: High Performance Cluster Computing, Buyya, R., ed., Prentice Hall, 1999, pp.661-681.
16. Topping, B.H., Khan, A.I.: Parallel Finite Element Computations. Saxe-Coburg Publications, 1996.
17. Tvrdik, P., Simecek, I.: Analytical Modeling of Optimized Sparse Linear Code (these Proceedings).
18. Wyrzykowski, R., Sczygiol, N., Olas, T., Kaniewski, J.: Parallel Finite Element Modeling of Solidification Processes. Lect. Notes in Comp. Sci. **1557** (1999) 183-195.
19. Wyrzykowski, R., Olas, T., Sczygiol, N.: Object-Oriented Approach to Finite Element Modeling on Clusters. Lect. Notes in Comp. Sci. **1947** (2001) 250-257.

Asymptotical Behaviour of the Communication Complexity of One Parallel Algorithm

Pavol Purcz

Technical University of Košice, Civil Engineering Faculty,
Vysokoškolská 4, 042 00 Košice, Slovakia,
Pavol.Purcz@tuke.sk,
http://svfweb.tuke.sk/pracoviska/km/purcz/index.htm

Abstract. Early suggested parallel "ring" algorithm for solving of the spatially one-dimensional initial-boundary-value problem (IBVP) for a parabolic equation using an explicit difference method is shortly described. Asymptotical behaviour of the communication complexity of this parallel algorithm is studied. Communication complexity is determined as a ratio between the number of interchanges and the number of arithmetical operations. It is proved that the coefficient of the communication complexity for spatially m-dimensional IBVP tends in general to $\frac{3}{4}$.

1 Introduction

Numerical applications become dominant in parallel scientific computing. The importance of the boundary-value problems (BVP) for engineering applications attracts and motivates continuous development of fast numerical algorithms for its solution as well as effective usage of parallel computational systems. A number of various approaches have been suggested, and some of them can be found in [1], [2], [3], [7], [9] and in other papers.

Some theoretical aspects of the use of an explicit difference method in case of spatially one-dimensional initial-boundary-value problem (IBVP) were suggested by Tyrtyshnikov [11] as a possible approach to development of a parallel algorithm. Some properties of recurrence relations like those of the explicit difference method were studied also by Kogge [6]. In this paper, such algorithm is shortly presented in the case of spatially one-dimensional IBVP. Some theoretical estimates of the communication complexity of the algorithm in case of spatially one- and m-dimensional IBVP are given. This parallel algorithm in one-, two- and three-dimensional case is described in detail in [8,10].

2 One-Dimensional IBVP Problem

Let us consider a spatially one-dimensional IBVP for a parabolic equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad t > 0, \quad 0 < x < l_x \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 201–206, 2004.
© Springer-Verlag Berlin Heidelberg 2004

$$u(0,x) = \varphi(x), \quad 0 < x < l_x \tag{2}$$
$$u(t,0) = \alpha(t), \quad t > 0 \tag{3}$$
$$u(t,l_x) = \beta(t), \quad t > 0 \tag{4}$$

where φ, α, β are sufficiently smooth given functions such that

$$\varphi(0) = \alpha(0),$$
$$\varphi(l_x) = \beta(0).$$

We look for a sufficiently smooth function $u(t,x)$, which satisfies the equation (1) with conditions (2) - (4). For the numerical solution of the problem (1) - (4) let us consider the following explicit difference method of the form

$$\frac{u_i^k - u_i^{k-1}}{\tau} = \frac{u_{i-1}^{k-1} - 2u_i^{k-1} + u_{i+1}^{k-1}}{h_x^2}, \tag{5}$$

where $\tau > 0$ and $h_x > 0$ are time step and spatial step of discretization, u_i^k are approximate values of the solution at the points (k,i), $(i = 1,2,\ldots,n_x - 1$, $k = 1,2,\ldots, \quad n_x \cdot h_x = l_x)$. Moreover, $u_0^k = \alpha(t_k)$, $u_{n_x}^k = \beta(t_k)$, and $u_i^0 = \varphi(x_i)$, where $x_i = ih_x$, $t_k = k\tau$.

Let us consider a natural number $n_x = q \cdot n$, where $q \geq 1$, $n \geq 2$. Then explicit difference scheme (5) can be realized on a parallel computer with n processors $P_1, P_2, \ldots, P_n$.

At the beginning each processor computes values u_i^0 at $q+1$ points lied on the axis O_x using the initial condition (2). Then each processor starts computation of values in the first time level according to the difference scheme (5). In order to compute each value u_i^1, $i = 1, ..., n_x - 1$, we need three values u_{i-1}^0, u_i^0, u_{i+1}^0 from previous time level. By the same procedure it is possible to determine next values u_i^k in the next time level. If some processor P_i has not enough data for computing another values u_i^k, a next phase of the computational process - data transfer between processors must be set. After finishing the data transfer it follows a new stage of computing with usage also transferred values. In this phase each processor P_i may compute next values u_i^k using the difference scheme (5) and the set of the data in its memory. So whole process of computations and data transfers regularly repeats. Generally for n processors the whole data exchange process can be written in the form:

$$P_{(i \bmod n)+1} \rightarrow P_i \rightarrow P_{(n+i-2 \bmod n)+1}, \; i = 1,2,...,n \tag{6}$$

3 Communication Complexity

3.1 Studied Problem

Let us consider a multi-processor computational system that performs two kinds of action:

a) numerical operations (addition, multiplication, ...),
b) information interchanges.

For many computational systems the mean time t_o of an action of type (a) is often much less than the time t_c of an action of type (b), i.e. $t_o << t_c$. The term information interchanges, or shortly interchanges represents both transfer of data from memory to processor and communication between processors. When the performance of an algorithm α is necessary NO_α numerical operations and NC_α interchanges, then the time t_α of its realization under the above conditions will have the form:

$$t_\alpha = NO_\alpha.t_o + NC_\alpha.t_c. \tag{7}$$

In the case $NO_\alpha \sim NC_\alpha$ we obtain

$$t_\alpha \sim NO_\alpha.t_c,$$

i.e. the system efficiency during the algorithm realization is given by the speed of the slowest component of the entire system. Naturally, the algorithms of this kind are solved unsatisfactorily on the computational system.
Let us rewrite (7) in the form

$$t_\alpha = NO_\alpha.t_o.[1 + \frac{NC_\alpha}{NO_\alpha}.\frac{t_c}{t_o}]. \tag{8}$$

We can see from the equation (8) that the "ideal" time of the algorithm realization would be $NO_\alpha.t_o$, because in this case the system works with the speed of the fastest component of the system. The magnitude

$$P_\alpha = \frac{NC_\alpha}{NO_\alpha}, \tag{9}$$

is the breaking coefficient that rises due to the data interchanges. This coefficient is called the *transmission ratio* or *information density* or in the last time *communication complexity* of the algorithm α. Due to (8) a satisfactory algorithm on the parallel computational system is such an algorithm that has the small communication complexity P_α.
The concept of the communication complexity is discussed in detail in works [4], [5] and [12].
Other splitting strategies of the computational effort between processors can be found in [13], [14],[15], [16] and in other papers.

3.2 Number of Arithmetical Operations

We suppose that points u_i^0, $i = 0, 1, ..., n_x$, given by initial condition $\varphi(x)$, are computed in advance and stored in memory. We consider a layer, which is $nq+1$ points long and q points wide. This layer regularly repeats in all considered parallel process. In general, the number of arithmetical operations for computing

one point u_i^k inside the considered layer and on its border is not the same. It depends on the computational complexity of the points specified by boundary conditions. We denote the number of arithmetic operations needed for computation of the values u_0^k and $u_{n_x}^k$ by $r(\alpha)$ and $r(\beta)$, respectively. The number of arithmetic operations needed in the parallel case is

$$NO_{par} = max\{5q^2; 5\frac{q^2}{2} + 5(\frac{q^2}{2} - q) + [r(\alpha) + r(\beta)]q\}.$$

3.3 Number of Interchanges

Let us again consider the same layer of the points, as at the previous subsection. According to the difference scheme (5) we need 5 interchanges for one step of the explicit difference method. Let's denote the number of interchanges needed for computing the values u_0^k and $u_{n_x}^k$ as $rc(\alpha)$ and $rc(\beta)$, respectively.

In the parallel case each processor working inside the considered area needs $5q^2$ interchanges. Each processor, which uses for computing also the boundary conditions, needs

$$5\frac{q^2}{2} + 5(\frac{q^2}{2} - q) + [rc(\alpha) + rc(\beta)]q$$

interchanges. According to the full description of the algorithm in the previous section we need next $\frac{q}{2} + \frac{q}{2} = q$ interchanges in the data exchange process. Then the whole number of interchanges needed in the parallel case is

$$NC_{par} = max\{5q^2; 5(q-1)q + [rc(\alpha) + rc(\beta)]q\} + q.$$

3.4 Asymptotical Behaviour of the Communication Complexity

According to the definition (9) and the previous results we can define the communication complexity for one-dimensional IBVP in the form

$$P_{par} = \frac{NC_{par}}{NO_{par}} = \frac{max\{5q; 5(q-1) + [rc(\alpha) + rc(\beta)]\} + 1}{max\{5q; 5(q-1) + [r(\alpha) + r(\beta)]\}}.$$

If $q >> 1$ then we can see, that

$$\lim_{q\to\infty} P_{par} = 1 \tag{10}$$

Hence, in general for spatially m-dimensional IBVP, we can deduce fhe following theorem.

Theorem 1.

$$\lim_{q\to\infty} P_{par} = \frac{3m+2}{4m+1}$$

Proof. The proof is possible to execute using the mathematical induction. The first step was performed by limit (10). That means the statement of the theorem holds for $m = 1$. The second step will be described now. Let us consider spatially l-dimensional IBVP ($m = l$) along with corresponding explicit difference scheme in the form:

$$\frac{u^k_{i_1 i_2 \ldots i_l} - u^{k-1}_{i_1 i_2 \ldots i_l}}{\tau} = \sum_{j=1}^{l} \frac{u^{k-1}_{i_1 i_2 \ldots i_{j-1} \ldots i_l} - 2u^{k-1}_{i_1 i_2 \ldots i_l} + u^{k-1}_{i_1 i_2 \ldots i_{j+1} \ldots i_l}}{h^2_{x_j}} \tag{11}$$

Let us consider spatially $l+1$-dimensional IBVP ($m = l+1$). Then to right side of the explicit difference scheme (11) we must add a term

$$\frac{u^{k-1}_{i_1 i_2 \ldots i_l} - 2u^{k-1}_{i_1 i_2 \ldots i_{l+1}} + u^{k-1}_{i_1 i_2 \ldots i_{l+2}}}{h^2_{x_{l+1}}}$$

what means that we must perform extra 4 numerical operations and 3 interchanges. Then

$$\frac{3l + 2 + 3}{4l + 1 + 4} = \frac{3(l+1) + 2}{4(l+1) + 1}$$

and the proof is finished.

Corollary 1.

$$\lim_{m \to \infty} \lim_{q \to \infty} P_{par} = \frac{3}{4}$$

4 Conclusion

Asymptotical behaviour of the communication complexity shows that considered parallel algorithm is more effective when dimension m of the IBVP is larger.

This result was obtained under assumption that the process of the interchange both transfer of data from memory to processor and communication between processors is separate, because in the theoretical description of this parallel algorithm we can see a regularly repetition of the both execution phases computation and data transfer together in all processors. On the other side, in a virtual computer environment we cannot eliminate some processes and effects, which can bring a particular covering of the both processes data transfer within processor and between processors. Therefore this theoretical result we can consider as a upper estimation of the communication complexity of this parallel algorithm and in virtual computer environment can be lower, which we can interpret as better parallel performance in the case of growing size of problem.

References

1. Burrage, K.: Parallel Methods for Initial Value Problems. J. Appl. Num. Math. **11**, (1993) 5–25

2. Crank, J., Nicolson, P.: A Practical Method for Numerical Evaluation of Solutions of PDEs of the Heat-Conduction Type. Proc. Camb. Phil. Soc. **43,** (1947) 60–67
3. Freeman, T.L., Phillips, C.: Parallel Numerical Algorithms. Prentice Hall (1992)
4. Gallivan, K., Plemmons, R.J., Sameh, A.H.: Algorithms for Dense Linear Algebra Computations. SIAM Review **32,1,** (1990) 54–135
5. Golub, G.H., Van Loan, C.F.: Matrix Computations. The Johns Hopkins University Press, Baltimore and London (1989)
6. Kogge, P.M.: Parallel Solution of Recurrence Problems. IBM Journal of Research and Development, **18,2,** (1974) 138–148
7. Ortega, J.M., Voigt, R.G.: Solution of PDE on Vector and Parallel Computers. SIAM, Philadelphia (1985)
8. Purcz, P.: Generalization of The Parallel Algorithm for Spatially Two - and Three - Dimensional Dirichlet Problem for a Parabolic Equation. Journal of Electrical Engineering, **50,10/s,** (1999) 36–39.
9. Peaceman, D.W, Rachford, H.H.: The Numerical Solution of Parabolic and Elliptic Differential Equations. J. Soc. Indust. Appl. Math. **3**, (1955) 28–41
10. Purcz, P.: Parallel Algorithm for Spatially One- and Two-Dimensional Initial-Boundary-Value Problem for a Parabolic Equation. Kybernetika **37,2,** (2001) 171–181
11. Tyrtyshnikov, E.E.: Parallelization of Some Numerical Methods. In: Numerical Solution of Partial Differential Equation, Lecture at the Slovak Mathematical Society Seminar Kosice, Slovakia (1992)
12. Tyrtyshnikov, E.E.: Blocks Algorithms of Linear Algebra. Computational Processes and Systems, **N9**, Nauka, Moscow (1993) 3–34
13. Zoeteweij, P.: A Coordination-Based Framework for Distributed Constraint Solving. Lecture Notes in Computer Science, **2627,** (2003) 171–184
14. Adve, V.S., Mellor-Crummey, J., Anderson, M., Wang, J.CH., Reed, D.A., Kennedy, K.: An Integrated Compilation and Performance Analysis Environment for Data Parallel Programs, Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM), San Diego, California, United States (1995) 50–es
15. Habbas, Z., Herrmann, F., Merel, P.-P., Singer, D.: Load Balancing Strategies for Parallel Forward Search Algorithm with Conflict Based Backjumping. Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS, (1997) 376–381
16. Reinefeld, A., Schnecke, V.: Work-Load Balancing in Highly Parallel Depth-First Search. Proceedings of the Scalable High-Performance Computing Conference, (1994) 773–780

Analytical Modeling of Optimized Sparse Linear Code

Pavel Tvrdík and Ivan Šimeček

Department of Computer Science and Engineering,
Czech Technical University, Prague
{tvrdik,xsimecek}@fel.cvut.cz

Abstract. In this paper, we describe source code transformations based on sw-pipelining, loop unrolling, and loop fusion for the sparse matrix-vector multiplication and for the Conjugate Gradient algorithm that enable data prefetching and overlapping of load and FPU arithmetic instructions and improve the temporal cache locality. We develop a probabilistic model for estimation of the numbers of cache misses for 3 types of data caches: direct mapped and s-way set associative with random and with LRU replacement strategies. Using HW cache monitoring tools, we compare the predicted number of cache misses with real numbers on Intel x86 architecture with L1 and L2 caches. The accuracy of our analytical model is around 97%. The errors in estimations are due to minor simplifying assumptions in our model.

1 Introduction

Sparse matrix-vector multiplication (shortly $SpM \times V$) is an important building block in algorithms solving sparse systems of linear equations, e.g., FEM. Due to matrix sparsity, the memory access patterns are irregular and the utilization of cache suffers from low spatial and temporal locality. Approaches to improve the performance of $SpM \times V$ are based on matrix reordering and register blocking [1,4], sometimes combined with sw-pipelining [2]. Due to its overhead, the register blocking achieves good speedups only for large number of executions of $SpM \times V$ with the same matrix A. All results in [1,2,4] are experimental and no analytical models are given. An analytical model for $SpM \times V$ is developed in [3], where the dependence of the number of cache misses on data and cache parameters is studied.

The contribution of this paper is twofold. (1) We have designed source code transformations based on sw-pipelining, loop unrolling, and loop fusion for $SpM \times V$ and for the Conjugate Gradient algorithm (CGA) that enable data prefetching and subsequently overlapping of load and FPU arithmetic instructions and improve the temporal cache locality. (2) We have developed a probabilistic model for estimations of the numbers of cache misses for data caches of 3 types: direct mapped and s-way set associative with random and with LRU replacement strategies. We have concentrated on Intel architecture with L1 and L2 caches. Using HW cache monitoring tools, we have verified that the accuracy of our analytical model is around 97%. The errors in estimations are due to minor simplifying assumptions in our model.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 207–216, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Terminology and Notation

A common method for solving sparse systems of linear equations appearing in FEM is the *Conjugate Gradients algorithm* (CGA). Such a sparse system of linear equations with n variables is usually represented by a *sparse* $(n \times n)$-matrix $\mathcal{A}$ stored in the format of *compressed rows* (see Figure 1). Let n_Z be the total number of nonzero elements in $\mathcal{A}$. Then $\mathcal{A}$ is represented by 3 linear arrays *A, adr,* and x. Array $A[1, \ldots, n_Z]$ stores the nonzero elements of $\mathcal{A}$, array $adr[1, \ldots, n]$ contains indexes of initial nonzero elements of rows of $\mathcal{A}$, and array $c[1, \ldots, n_Z]$ contains column indexes of nonzero elements of $\mathcal{A}$. Hence, the first nonzero element of row j is stored at index $adr[j]$ in array *A*. Let n_{zpr} denote the *average* number of nonzero elements in matrix $\mathcal{A}$ per row: $n_{zpr} = n_Z/n$. Let w_B denote the *bandwidth* of matrix *A,* i.e., the maximal difference between column indexes of 2 nonzero elements in rows of $\mathcal{A}$.

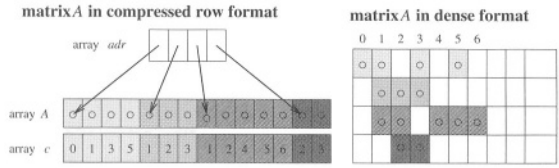


Fig. 1. The compressed row representation of a sparse matrix $\mathcal{A}$.

The most time-consuming part of the CGA is a multiplication $\boldsymbol{y} = \mathcal{A}\boldsymbol{x}$ of sparse matrix $\mathcal{A}$ by dense vector $\boldsymbol{x}$ (shortly $SpM \times V$). Because of indirect addressing, memory accesses to the elements of vector $\boldsymbol{x}$ are irregular and the cache utilization suffers from low spatial and temporal locality.

The cache model we consider corresponds to the structure of L1 and L2 caches in the Intel x86 architecture. An *s-way set-associative* cache consists of h *sets* and one set consists of s independent *blocks* (called *lines* in the Intel terminology). Let C_S denote the size of the data part of a cache in bytes and B_S denote the cache block size in bytes. Then $C_S = s \cdot B_S \cdot h$. Let S_D denote the size of type `double` and S_I the size of type `integer`. Let n_{dpb} denote the number of items of type `double` per cache block and n_{ipb} the number of items of type `integer` per cache block. Obviously, $B_S = n_{dpb} S_D = n_{ipb} S_I$.

We distinguish 2 types of cache misses: *Compulsory* (sometimes called *intrinsic*) misses that occur when empty cache blocks are loaded with new data and *thrashing* misses (also called *cross-interference* or *conflict* misses) that occur when useful data are loaded into a cache block, but the cache associativity s is too small, these data are replaced prematurely, and subsequently, they must be reloaded again.

3 Linear Code Optimizations

3.1 Sparse Matrix-Vector Multiplication

Consider a sparse matrix $\mathcal{A}$ represented by linear arrays *A, adr,* and c as defined in Section 2 and a vector $\boldsymbol{x}$ represented by dense array $x[1, \ldots, n]$. The goal is to compute the vector $\boldsymbol{y} = \mathcal{A}\boldsymbol{x}$ represented by dense array $y[1, \ldots, n]$.

Standard sparse matrix-vector multiplication. The following algorithm SMVMS is a straightforward implementation of $SpM \times V$.

```
Algorithm SMVMS(in A, adr, c, x; out y) (* Standard implementation of SpM×V *)
double A[1,...,nz], c[1,...,nz], x[1,...,n], y[1,...,n]; integer adr[1,...,n+1];
(1) low = adr[1];
(2) for i = 1 to n do {
(3)       s = 0; up = adr[i+1];
(4)       for j = low to up − 1 do {k = c[j]; s+ = A[j] * x[k];}
(5)       y[i] = s; low = up; }
```

The multiplication requires indirect addressing (codeline (4)), which causes performance degradation.

Sparse matrix-vector multiplication with prefetching. The number of load instructions in a $SpM \times V$ is about three times the number of FP arithmetic operations. Therefore, the CPU performance depends strongly on the CPU-memory bandwidth.

```
Algorithm SMVMM(in A, adr, c, x;out y) (* SpM×V with prefetching *)
double A[1,...,nz], c[1,...,nz], x[1,...,n], y[1,...,n]; integer adr[1,...,n+1];
(1)  low = adr[1];
(2)  for i = 1 to n do {
(3)       s = s1 = s2 = 0; up = adr[i+1];
(4)       if ((up − low) > 2) then (* for "long" rows *)
(5)       {  k1 = c[low]; a1 = A[low]; x1 = x[k1];
(6)          k2 = c[low+1]; a2 = A[low+1]; x2 = x[k2]; j = low+2;
(7)          do {
(8)             s1+ = a1 * x1; k1 = c[j]; a1 = A[j]; x1 = x[k1];
(9)             s2+ = a2 * x2; k2 = c[j+1]; a2 = A[j+1]; x2 = x[k2]; j+ = 2;
(10)         } while (j < (up − 1));
(11)         if (j < up) then {s = a1 * x1; k1 = c[j]; a1 = A[j]; x1 = x[k1];}
(12)         s1+ = a1 * x1; s2+ = a2 * x2; s+ = s1 + s2; }
(13)      else (* for "short" rows *)
(14)         for j = low to up − 1 do {k = c[j]; s+ = A[j] * x[k];}
(15)      y[i] = s; low = up;
(16) }
```

To improve the performance, the *sw-pipelining* [5] is applied. Data are prefetched before they are used, which leads to overlapping of load and numerical instructions. The multiplication on codeline (8) is performed in parallel with loading data for multiplication on the next codeline (9). Hence, the sw-pipelining requires unrolling to a factor of 2 to achieve a concurrent execution of load and arithmetic operations. Variables a_1, a_2, x_1, x_2, k_1, and k_2 are used for prefetching.

3.2 CG Algorithm

Consider a sparse *symmetric positive definite* matrix $\mathcal{A}$ in the compressed row format as defined in Section 2 and an input dense array $y[1, \ldots, n]$, representing vector $\boldsymbol{y}$. The goal is to compute the output dense array $x[1, \ldots, n]$, representing solution vector $\boldsymbol{x}$ of linear system $A\boldsymbol{x} = \boldsymbol{y}$.

A standard CGA implementation. A straightforward implementation of the CGA is the Axelsson's variant without preconditioning.

Algorithm CGS(**in** A, adr, c, y;**out** x) (* CGA *standard implementation* *)
Auxiliary: double $d[1,\ldots,n], p[1,\ldots,n], r[1,\ldots,n], nom, nory, denom, \alpha$;
(1) $nory = 0$;
(2) **for** $i = 1$ **to** n **do**
(3) { $nory+ = y[i] * y[i]$; $r[i] = -y[i]$; $x[i] = 0.0$; $d[i] = y[i]$; }
(4) $nom = nory$;
(5) **while** (residual vector is "large") **do** {
(6) **call** SMVMS$(A, adr, c, d; p)$;
(7) **call** Dot_product$(d, p; denom)$; $\alpha = nom/denom$;
(8) **for** $i = 1$ **to** n **do** { $x[i]+ = \alpha * d[i]$; $r[i]+ = \alpha * p[i]$; }
(9) $denom = nom$; **call** Dot_product$(r, r; nom)$; $\alpha = nom/denom$;
(10) **for** $i = 1$ **to** n **do** { $d[i] = \alpha * d[i] - r[i]$; }
(11) }

This code has a serious drawback. If the data cache size is less than the total memory requirements for storing all input, output, and auxiliary arrays (A, adr, c, d, p, r, x, y), then due to thrashing misses, part or all of these arrays are flushed out of the cache and must be reloaded during the next iteration of the **while** loop at codelines (5)-(1 1). This inefficiency can be reduced by application of the *loop reversal* [5] and *loop fusion* [5].

An Improved CGA Implementation Based on Loop Restructuring

Algorithm CGM(**in** A, adr, c, y;**out** x) (* CGA *modified implementation* *)
Auxiliary: double $d[1,\ldots,n], p[1,\ldots,n], r[1,\ldots,n], nom, nory, denom, \alpha$;
(1) $nory = 0$;
(2) **for** $i = 1$ **to** n **do**
(3) {$nory+ = y[i] * y[i]$; $r[i] = -y[i]$; $x[i] = 0.0$; $d[i] = y[i]$;}
(4) $nom = nory$;
(5) **while** (residual vector is "large") **do** {
(6′) **Loop fusion** (SMVMM$(A, adr, c, d; p)$, Dot_product$(d, p; denom)$);
(8′) $\alpha = nom/denom$; $nory = 0.0$;
(8″) **for** $i = 1$ **to** n **do** {$x[i]+ = \alpha * d[i]$; $r[i]+ = \alpha * p[i]$; $nory+ = r[i] * r[i]$;}
(9′) $denom = nom$; $nom = nory$; $\alpha = nom/denom$;
(10′) **for** $i = n$ **downto** 1 **do** { $d[i] = \alpha * d[i] - r[i]$; }
(11) }

CGM code has been obtained from CGS code by applying 4 transformations:

1. The SMVMS algorithm has been replaced by the SMVMM algorithm.
2. Codelines (6, 7) in CGS are grouped together by *loop fusion* and this allows to reuse immediately the new computed values of array p.
3. Similarly, codelines (8,9) in CGS are grouped together by loop fusion and this allows to reuse immediately the new computed values of array r (see codelines (8"), (9') in CGM).

4. The loop on codeline (10) in CGS is reversed by *loop reversal* so that the last elements of arrays d and r that remain in the cache from the loop on codelines (8, 9) can be reused.

The CGM code has better *temporal locality* of data and in Section 4.2 we perform the quantitative analysis.

4 Probabilistic Analysis of the Cache Behavior

In this section, we propose a probabilistic model of the cache behavior for $SpM\times V$ and CGAs and compare the values predicted by the model with real measured data. The proposed analytical model is valid for both L1 and L2 caches, since in the Intel architecture, they have the same structure and differ only in quantitative parameters. Therefore, we do not distinguish between them explicitly and talk only about a cache in general.

4.1 Sparse Matrix-Vector Multiplication

Algorithms SMVMS and SMVMM (see Section 3.1) produce the same sequence of memory access instructions and therefore, the analytical model is the same for both. It is based on the following simplified assumptions:

1. There is no correlation among mappings of arrays A, c, and x into cache blocks. Hence, we can view load operations as mutually independent events.
2. We consider *thrashing misses* only for load operations from arrays A, c, and x.
3. We assume that the whole cache size is used for data of $SpM\times V$.
4. We assume that each execution of $SpM\times V$ starts with empty cache.

We use the following notation.

- $P(Z[i])$ denotes the probability of a thrashing miss of the cache block containing element i of array Z.
- N_{CM} denotes the number of cache misses during one execution of $SpM\times V$.
- d denotes the average number of iterations of the innermost loop of SMVMS at codeline (4) between 2 reuses of the cache block containing some element of array $x[1,\ldots,n]$.

We distinguish 3 relevant types of sparse matrices to estimate the value of d. (1) A symmetric sparse matrix $\mathcal{A}$ with bandwidth w_{B} and with *uniform* distribution of nonzero elements on rows. Then d can be approximated by $d = w_{\mathrm{B}}$ [3]. (2) A symmetric sparse banded matrix $\mathcal{A}$ with *similar row structure*. Two rows i and $i+1$ are said to be *similar* if row i contains nonzero elements

$$\cdots, A[i][i-\Delta], A[i][i], A[i][i+\Delta], \cdots, \tag{1}$$

whereas row $i+1$ contains nonzero elements

$$\cdots, A[i+1][i+1-\Delta], A[i+1][i+1], A[i+1][i+1+\Delta], \cdots, \tag{2}$$

where Δ is a constant.

In other words, we assume that the indexes of nonzero elements of row $i+1$ are only "shifted" by one position with respect to row i. For A with structurally similar rows, the cache block containing an element $x[i]$ is reused with high probability during loading $x[i+1]$ after n_{zpr} iterations[1]. Hence, $d = \min(w_{\mathrm{B}}, n_{\mathrm{zpr}}) = n_{\mathrm{zpr}}$. (3) A sparse banded matrix A where $\frac{w_{\mathrm{B}}}{n_{\mathrm{zpr}}} \approx 1$. Then $x[i]$ is reused with high probability during loading $x[i+1]$ in the next iteration, and therefore, $d = 1$.

Due to the assumption that each new execution of $SpM{\times}V$ starts with empty cache, all n_{Z} elements of arrays A and c and all n elements of arrays x, *adr*, y must be loaded into the cache once and the number of *compulsory misses* is

$$N_{\mathrm{CM}}^{\mathrm{C}} = \frac{n_{\mathrm{Z}}\,(S_{\mathrm{D}} + S_{\mathrm{I}}) + n\,(2 \cdot S_{\mathrm{D}} + S_{\mathrm{I}})}{B_{\mathrm{S}}}. \tag{3}$$

Since caches have always limited size, thrashing misses occur: Data loaded into a cache set may cause replacement of other data that will be needed later. Hence, the total number of cache misses is

$$N_{\mathrm{CM}} = N_{\mathrm{CM}}^{\mathrm{C}} + N_{\mathrm{CM}}^{\mathrm{T}}. \tag{4}$$

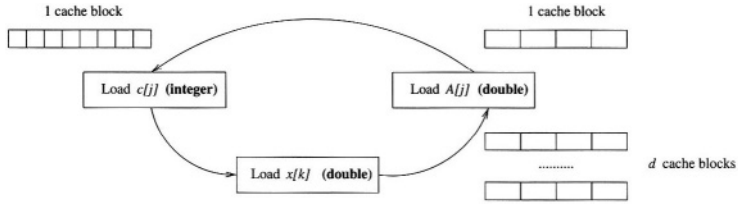


Fig. 2. The iterated sequence of load operations in the innermost loop of Algorithm SMVMS. The diagram assumes that $n_{\mathrm{dpb}} = 4$ and $n_{\mathrm{ipb}} = 8$.

Direct mapped cache ($s = 1$). The diagram on Figure 2 shows load operations in the innermost loop of Algorithm SMVMS or SMVMM. The loop has n_{Z} iterations. Each element of arrays A, c, and x in each iteration is either reused or replaced due to thrashing. Under our assumption of independence of these replacements for all 3 arrays, the total number of thrashing misses can be approximated by formula:

$$N_{\mathrm{CM}}^{\mathrm{T}} = n_{\mathrm{Z}}\,(P(A[j]) + P(c[j]) + P(x[k])) \quad ; \forall j, k \tag{5}$$

The probability that 2 randomly chosen cache blocks from the cache are distinct is $1 - \frac{B_{\mathrm{S}}}{C_{\mathrm{S}}} = 1 - h^{-1}$. Hence,

$$P(c[j]) = P(A[j]) = 1 - (1 - h^{-1})^2. \tag{6}$$

[1] For simplicity, we assume that all rows corresponding to discretization of internal mesh nodes are similar. But in real applications, boundary mesh nodes produce equations with a slightly different structure. This simplification should not have a significant impact, since the number of boundary nodes is of order of square root of all mesh nodes.

Arrays A and c are accessed linearly, their indexes are incremented in each iteration of the innermost loop, and in a given moment, only one cache block is actively used unless thrashing occurs. The access pattern for array x is more complicated due to indirect addressing. In the worst case, an element of x, after loading into the cache, is reused only after d iterations of the innermost loop, since it is used at each row of matrix A only once (said otherwise, array x actively uses d cache blocks). Every load during this time can cause cache thrashing. Hence,

$$P(x[k]) = 1 - (1 - h^{-1})^{3d}. \tag{7}$$

s-way set-associative cache, random replacement strategy. The probability can be derived as in the previous section, only the thrashing occurs with probability $\frac{1}{s}$. So, one of s cache blocks containing elements of A or c can be replaced by both loads, which are assumed independent, with probability

$$P(A[j]) = P(c[j]) = \frac{1}{s}(1-(1-h^{-1})^2) \quad \text{and} \quad P(x[k]) = \frac{1}{s}(1-(1-h^{-1})^{3d}). \tag{8}$$

s-way set-associative cache, LRU replacement strategy. In the LRU case, cache block can be replaced only if at least s immediately preceding loads accessed this block. Hence,

$$P(A[j]) = P(c[j]) = \begin{cases} 0 & \text{if } s > 2, \\ h^{-2} & \text{if } s = 2, \\ 1 - (1 - h^{-1})^2 & \text{if } s = 1. \end{cases} \tag{9}$$

Arrays A and c are accessed in the linear order of indexes as before and during d iterations completely fill $\varrho = \lfloor \frac{d(S_D+S_I)}{sB_S} \rfloor$ bcache sets. We distinguish 2 cases of sparsity of matrix A:

- $\frac{w_B}{n_{zpr}} \approx 1$. This corresponds to a sparse banded matrix $\mathcal{A}$ of type (3) in Section 4.1 where cache sets are almost completely filled with array x. Then memory access pattern is the same as for arrays A and c.

$$P(x[k]) = P(A[j]) = P(c[j]). \tag{10}$$

- $\frac{w_B}{n_{zpr}} \geq n_{dpb}$. This corresponds to a sparse matrix $\mathcal{A}$ of type (1) or (2) in Section 4.1 such that every cache block contains during execution of $SpM\times V$ at most one element of array x. If the load operations for arrays A or c replace a cache block containing array x, one load miss occurs.

$$P(x[k]) = 1 - (1 - h^{-1})^{\varrho} \tag{11}$$

4.2 CGA

We consider the same data structures A, adr, c, x, y as in Algorithms `CGS` and `CGM`. Let us further assume the following simplified conditions.

1. There is no correlation among mappings of arrays A, d, p, r, x, and y into cache blocks.

2. Thrashing misses occur only within the subroutines for $SpM \times V$. Hence, we consider only *compulsory* cache misses.
3. We again assume that the whole cache size is used for input, output, and auxiliary arrays.
4. We assume that each iteration of the **while** loop of the CGA starts with an empty cache.

Define

NS_l = the predicted number of cache misses that occur on codeline (l) of the algorithm CGS.

NM_l = the predicted number of cache misses that occur on codeline (l) of the algorithm CGM.

The total number of cache misses for $SpM \times V$ on codeline (6) in CGS and on codeline (6') in CGM was evaluated in Section 4.1, Equation (4).

$$NS_{(6)} = NM_{(6')} = N_{\mathrm{CM}}.$$

The number of compulsory misses for the dot product of 2 vectors on codeline (7) in CGS is the number of cache blocks needed for $2n$ doubles. Whereas in CGM, the dot product is computed directly during the multiplication due to the loop fusion and all elements of arrays d and p are reused. Hence,

$$NS_{(7)} = \frac{2n}{n_{\mathrm{dpb}}} \quad \text{and} \quad NM_{(7')} = 0.$$

The codeline (8) contains 2 linear operations on 4 vectors and the same holds for CGM. Therefore,

$$NS_{(8)} = NM_{(8'+8'')} = \frac{4n}{n_{\mathrm{dpb}}}.$$

The codeline (9) contains a dot product of vector r, whereas in CGM on the codeline (8") the same dot product is computed directly during those linear vector operations and all elements of array r are reused. So,

$$NS_9 = \frac{n}{n_{\mathrm{dpb}}} \quad \text{and} \quad NS_{9'} = 0.$$

The codeline (10) contains linear operations on 2 vectors. For large n, we can assume that after finishing the loop on the codeline (9), the whole cache is filled only with elements of arrays d, p, r, and x. In CGM, the loop on codeline (10') is reversed and so, in the best case, the last $\frac{C_{\mathrm{S}}}{4S_{\mathrm{D}}}$ elements of array r (similarly for array d) can be reused. Therefore,

$$NS_{10} = \frac{2n}{n_{\mathrm{dpb}}} \quad \text{and} \quad NM_{10'} = \frac{2n}{n_{\mathrm{dpb}}} - \frac{C_{\mathrm{S}}}{2B_{\mathrm{S}}}.$$

Table 1. Summary of the predicted (N_{CM}) and measured (RN_{CM}) numbers of cache misses in L1 and L2 caches for algorithms `SMVMS` and `SMVMM` and corresponding speedups in cycles per program.

n	n_Z	w_B	$N_{\mathrm{CM}}(L1)$	$RN_{\mathrm{CM}}(L1)$	$N_{\mathrm{CM}}(L2)$	$RN_{\mathrm{CM}}(L2)$	R_1	R_2	CPP SMVMS	CPP SMVMM	*speedup*
[K]	[K]		[K]	[K]	[K]	[K]	[%]	[%]	[M]	[M]	
5.1	64.9	209	27.9	28.6	27	27.6	97.5	97.8	3.05	2.82	1.08
9.9	127	289	54.7	57.1	53	54.2	95.7	97.7	5.72	5.36	1.07
20.2	260	409	112	116	108	111	96	97.8	11.51	10.77	1.07
39.5	509	569	219	227	212	217	96.3	97.6	22.44	20.89	1.07
80.4	1040	809	446	464	432	445	96.3	97.2	45.82	40.16	1.14

Table 2. Summary of the predicted (N_{CM}) and measured (RN_{CM}) numbers of cache misses in L1 and L2 caches for algorithms `CGS` and `CGM`.

n	$N_{\mathrm{CM}}(L1)$	$RN_{\mathrm{CM}}(L1)$	$N_{\mathrm{CM}}(L2)$	$RN_{\mathrm{CM}}(L2)$	R_1	R_2	CPP CGS	CPP CGM	*speedup*
[K]	[K]	[K]	[K]	[K]	[%]	[%]	[G]	[G]	
5.1	35.3	35.8	32.7	33.7	98.6	96.9	0.609	0.566	1.08
9.9	69.4	71.4	65.9	66.9	97.1	98.5	1.69	1.54	1.1
20.2	142	146	136	140	96.7	97.3	5.06	4.52	1.12
39.5	278	287	269	275	96.9	97.9	13.9	12.4	1.12
80.4	567	584	551	562	97.1	98.1	42.9	36.0	1.19

4.3 Evaluation of the Probabilistic Model

Tables 1-2 give performance numbers for Pentium Celeron 1GHz, 256 MB running W2000 Professional with the following cache parameters: L1 cache is data cache with $B_S = 32$, $C_S = 16K$, $s = 4$, $h = 128$, and LRU replacement strategy. L2 cache is unified with $B_S = 32, C_S = 128K, s = 4, h = 1024$, and LRU strategy. The parameters

$$R_1 = \frac{N_{\mathrm{CM}}(L1)}{RN_{\mathrm{CM}}(L1)} \quad \text{and} \quad R_2 = \frac{N_{\mathrm{CM}}(L2)}{RN_{\mathrm{CM}}(L2)}$$

denote the ratios of the estimated and real numbers of misses for L1 and L2 caches, respectively. They represent the accuracy of the probabilistic model.

Table 1 illustrates that the real number of cache misses was predicted with average accuracy 97% in case of $SpM{\times}V$ and that the optimized $SpM{\times}V$ algorithm achieves speedup around 10% due to the overlapping of load and FPU operations. The speedup increases with the matrix order n. Table 2 shows that the accuracy of the probabilistic cache model is also around 97% and the speedup of optimized CGA is due to fusions more than 10% and it again increases with n. The accuracy of an analytical model is influenced by the following assumptions.

1. Successive loads of items of array x are assumed mutually independent from the view of mapping into cache sets and that happens if the structure of matrix A is random, but this is not true in real cases.
2. In the $SpM{\times}V$, we consider only arrays A, c, and x, but the algorithms `SMVMS` and `SMVMM` (see Section 3.1) also load arrays *adr* and y to caches.

3. We assume that both L1 and L2 caches are data caches. In the Intel architecture, this assumption holds only for the L1 cache, whereas the L2 cache is unified and it is used also for storing instructions. This fact is not taken into account in our formulas, but the error is small due to small code sizes. Similarly, a small part of the error is due to system task codes in L2.
4. In the CGAs, we assume that every iteration is independent. This assumption is valid for $C_S \leq n(5 \cdot S_D + S_I) + n_{zpr}(S_D + S_I)$ (all memory requirements for storing the arrays A, *adr, c, d, p, r, x, y* in the CGAs).

5 Conclusions

The code transformations proposed in this paper for $SpM\times V$ and CGA cause that data for FPU operations are prefetched and the overlapping of load and FPU instructions increases. Since the number of load instructions is about three times the number of FPU operations and everything is pipelined, the speedup achieved by this technique is limited. Our speedups are about 10% in cycles per program for $SpM\times V$ and 15% for the CGA. These numbers grow with n. Another main limitation follows from the small number of GPRs and FPRs in the Intel CPU, since unrolling to a factor of $k \geq 2$ requires $O(k)$ registers.

Our analytical probabilistic model for predicting cache behavior is similar to the model in [3], but we differ in 2 aspects. We have explicitly developed and verified a model for s-way set associative caches with LRU block replacement strategy and obtained average accuracy of predicted numbers of cache misses equal to about 97% for both $SpM\times V$ and CGA. In contrast to [3], our results indicate that cache miss ratios for these 2 applications are sensitive to the replacement strategy used. In contrast to [3], we consider both compulsory and thrashing misses for arrays A and c.

References

1. D. B. Heras, J. C. Cabaleiro, and F. F. Rivera. Modeling data locality for the sparse matrix-vector product using distance measures. *Parallel Computing,* 27(7):897–912, June 2001.
2. S. Rollin and R. Geus. Towards a fast parallel sparse matrix-vector multiplication. In E. H. D'Hollander, J. R. Joubert, F. J. Peters, and H. Sips, editors, *Parallel Computing: Fundamentals and Applications, Proc. of PARCO'99,* pages 308–315. Imperial College Press, 2000.
3. O. Temam and W. Jalby. Characterizing the behavior of sparse algorithms on caches. In *Supercomputing,* pages 578–587, 1992.
4. R. Vuduc, J. W. Demmel, K. A. Yelick, S. Kamil, R. Nishtala, and B. Lee. Performance optimizations and bounds for sparse matrix-vector multiply. In *Proceedings of Supercomputing 2002,* Baltimore, MD, USA, November 2002.
5. M. Wolfe. *High-Performance Compilers for Parallel Computing.* Addison-Wesley, Reading, Massachusetts, USA, 1995.

A BSP Parallel Model for the Göttfert Algorithm over F_2

Fatima Abu Salem

Oxford University Computing Laboratory,
Wolfson Building,
Parks Road,
OX1 3QD, UK
fkas@comlab.ox.ac.uk

Abstract. In Niederreiter's factorization algorithm for univariate polynomials over finite fields, the factorization problem is reduced to solving a linear system over the finite field in question, and the solutions are used to produce the complete factorization of the polynomial into irreducibles. For fields of characteristic 2, a polynomial time algorithm for extracting the factors using the solutions of the linear system was developed by Göttfert, who showed that it is sufficient to use only a basis for the solution set. In this paper, we develop a new BSP parallel algorithm based on the approach of Göttfert over the binary field, one that improves upon the complexity and performance of the original algorithm for polynomials over F_2. We report on our implementation of the parallel algorithm and establish how it achieves very good efficiencies for many of the case studies.

1 Introduction

One important problem in symbolic computation is the factorization of univariate polynomials over finite fields, where the Niederreiter algorithm has been introduced [1]. A major feature of the algorithm consists in its linearization of the factorization problem, or reducing it to solving a linear system over the finite field in question, where the solutions lead to a complete factorization in a variety of ways. One such method was presented by Göttfert [2] for fields of characteristic 2, and is best featured as a simple and polynomial time algorithm for extracting the factors using only the basis elements of the solution set of the Niederreiter system. In this paper, we develop a new BSP parallel approach to the Göttfert algorithm over F_2. Our algorithm achieves high efficiency in many of our test cases and can thus be used efficiently to factorize very large polynomials over F_2, provided a basis of the solution set is given. In Section 2, we give a brief survey of the algorithms underlying our work and some background information describing the BSP parallel model. In Section 3 we present our parallel algorithm and discuss its BSP cost analysis. In Section 4 we report on our experimental results and discuss the scalability of the algorithm.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 217–224, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Göttfert's Refinement of the Niederreiter Algorithm over F_2 and the BSP Model

For complete proofs and results related to the Niederreiter and Göttfert algorithms we refer the reader to [1], [2], [3], [4]. Let F_2 be the binary field of order 2 consisting only of the elements 0,1; it is thus understood that all polynomials described in this paper are monic. Let f be a polynomial of degree d over F_2, and $f = g_1^{e_1} ... g_m^{e_m}$ be its canonical factorization of over the field. Let N_f be the Niederreiter matrix of coefficients of f ([1], [2]). The Niederreiter linear system can be expressed as $(N_f - I_d)h^T = 0$, where I_d is the $d \times d$ identity matrix over F_2 and $h = (h_0, ..., h_{d-1})$ is the coefficient row vector of an unknown polynomial h over F_2 of degree less than d. A crucial result establishes that there are 2^m solutions h of the system, and that any of them can eventually produce a complete factorization into irreducibles. In particular, the solutions h of the linear system form a linear subspace of $F_2[x]$ of dimension m over F_2. For full details relating to the Niederreiter algorithm, we refer the reader to [1], [3], [4]. Suppose that $m > 1$ so that the factorization is non-trivial. Several methods have been suggested to achieve a factorization using solutions of the sytem, but have the disadvantage of either requiring all 2^m solutions to be found or several linear systems to be solved before a complete factorization is established. In a new method, Göttfert showed that it is enough to use the m polynomials corresponding to any basis of the linear system, without having to solve other than the original linear system associated with f. To illustrate, let $\{h_1, ..., h_m\}$ be a basis spanning the solution set of the linear system. For $i = 1, ..., m$, the corresponding polynomials $b_i = f/\gcd(f, h_i)$ are square-free factors of f [1]. The factors are listed in a collection of rows as follows. The first row contains only b_1. The second row consists of at most three polynomials, specifically, the non-constant polynomials among $\gcd(b_2, b_1)$, $b_1/\gcd(b_2, b_1)$, and $b_2/\gcd(b_2, b_1)$. In general, the polynomials of row n, for $n = 3, ..., m$, consist of the non-constant polynomials among d_1, r_1/d_1, ..., d_s, r_s/d_s, $b_n/d_1 ... d_s$, where $r_1, ..., r_s$ are the polynomials in row $n-1$ and $d_j = \gcd(b_n, r_j)$ for $j = 1, ..., s$. In [2], it is shown that the polynomials in any row are pairwise relatively prime square-free factors of f, that the polynomial b_n appears in row n, either in its original form or split up into some non-constant factors, and that every polynomial in row $n-1$ also appears in row n, either in its original form or split up into two non-constant factors. More importantly, it is shown that this process ends successfully by setting up at most m rows, as a consequence of the following theorems [2]:

Theorem 1. *The irreducible square-free factors of f are determined once a row containing m non-constant polynomials has been reached.*

Theorem 2. *A row of index at most m contains the polynomials $g_1, ..., g_m$, the distinct irreducible factors of f.*

It can also be shown that the total cost of this algorithm is at most $O(m^2 M(d) \log d)$ field operations, where $\log d$ is the binary logarithm of d, and

$M(d)$ is the arithmetic complexity to multiply (or divide) two polynomials of degree at most d over F_2.

The *bulk synchronous parallel model* (BSP) [5], [6], [7], is a model for programming which provides a simple framework to achieve portable parallel algorithms independent of the architecture of the computer on which the parallel work is carried out. A BSP computer consists of a set of p processors each with its own private memory, and having remote access to other processors' private memories through a communication network. A BSP algorithm consists of a sequence of parallel steps, denoted by *supersteps*. Communication supersteps are followed by *synchronization barriers*, whereby all transferred data is updated. Processors are distinguished by their own identification number, $id = 0, ..., p-1$. A BSP computer can be described by machine dependent parameters: s, the processor speed (in flop/sec), g, the time (in flop time units) it takes to communicate (send or receive) a data element, and ℓ, the time (in flop time units) it takes all processors to synchronize. The BSP cost is established using the parameters g and ℓ and the cost of an algorithm is simply the sum of the BSP costs of its supersteps. The estimate of the execution time is obtained by dividing the BSP cost in flop time units by s. The complexity of a superstep is defined as $w_{max} + g.h_{max} + \ell$,where w_{max} is the maximum number of flops performed, and h_{max} is the maximum number of messages sent or received, by any one processor during that superstep. In the present paper, all field operations are considered as flops, since we are working over F_2.

3 A Parallel Approach to Göttfert's Algorithm

Let $\#r_n$ denote the maximum number of non-constant polynomials P_i, for $i = 1, ..., \#r_n$, that can appear in any row n described in the construction above. Each P_i can be the result of a gcd or a division operation, in which case we denote it by a D-polynomial or an R-polynomial respectively. It is easy to see that that there are at most $(\#r_n - 1)/2$ non-constant D-polynomials and at most $(\#r_n + 1)/2$ non-constant R-polynomials in each row n. We denote D and R-polynomials in row n by $n; D_j$ and $n; R_{j'}$ respectively, where j and j' are the polynomials' indices along row n. For consistency throughout the text, we can arrange the computations along rows so that all the D polynomials are computed first, their corresponding R polynomials next, and the polynomial $b_n / \prod_j n; D_j$ (where the product is over non-constant polynomials $n; D_j$) last. With this notation, it is also easy to see that, if the polynomials in row $n-1$ are written as $(n-1); D_i$, for some $i = 1, ..., (\#r_{n-1} - 1)/2$, and $(n-1); R_i$, for some $i = (\#r_{n-1} + 1)/2, ..., \#r_{n-1}$, then row n consists of

$$n; D_i = \begin{cases} \gcd(b_n, (n-1); D_i), & \text{if } 1 \le i \le \frac{\#r_{n-1}-1}{2} \quad (1) \\ \gcd(b_n, (n-1); R_{i-(\#r_{n-1}-1)/2}), & \text{if } \frac{\#r_{n-1}+1}{2} \le i \le \#r_{n-1} \quad (2) \end{cases}$$

$$n; R_i = \begin{cases} (n-1); D_i / n; D_i, & \text{if } 1 \le i \le \frac{\#r_{n-1}-1}{2} \quad (3) \\ (n-1); R_{i-(\#r_{n-1}-1)/2} / n; D_i, & \text{if } \frac{\#r_{n-1}+1}{2} \le i \le \#r_{n-1} \quad (4) \end{cases}$$

and $n; R_{\#r_{n-1}+1} = b_n / \prod_{i=1}^{\#r_{n-1}} n; D_i$ for non-constant $n; D_i$. Furthermore, we assert the following (see [8] for proof):

Claim. $\#r_n = 2^n - 1$ for $n = 1, ..., m$.

The first step in our parallel approach consists of studying the dependencies between the gcd and division computations and structuring them in a parallel hierarchy. Without loss of generality we may assume that the number of threads coincides with the number of processors available. We define a parallel queue as one which consists of a list of polynomials that can be computed independently by a number of p processors using a number of parallel supersteps, such that the supersteps can be executed without requiring a synchronization point throughout the queue. The first parallel queue consists of the polynomials b_i, for $i = 1, ..., m$. The second parallel queue consists of the polynomial $2; D_1$ only, since all other polynomials (in its row or in following rows) depend on it. This constitutes the only queue where not enough distinct tasks are available to engage all processors. In fact, the ensuing queues start filling up immediately according to an iterative formula derived from the dependencies that we describe in the following algorithm:

Algorithm 1 *Set_Queues*$(queue_k, queue_{k'})$
Input: $queue_k = \{P_1, ..., P_s\}$, *a list of non-constant polynomials from the Göttfert setting computed in a parallel queue* $k \geq 2$.
Output: *a list* $queue_{k'}$ *of polynomials to be computed in the parallel queue* $k' > k$.
1. $queue_{k'} \leftarrow ()$; *for* $j \in \{1, ..., s\}$ *do*
  if $P_j = n; D_i$ *for some* $n = 2, ..., m$ *and some* $i = 1, ..., \#r_{n-1}$ *do*
2. $queue_{k'} \leftarrow queue_{k'} \cup n; R_i \cup (n+1); D_i$.
  end;
  if $P_j = n; D_i$ *for some* $n = 2, ..., m$ *and* $i = \#r_{n-1}$ *do*
3. $queue_{k'} \leftarrow queue_{k'} \cup n; R_{\#r_{n-1}+1}$.
  end;
  if $P_j = n; R_i$ *for some* $n = 2, ..., m$ *and some* $i = 1, ..., \#r_{n-1} + 1$ *do*
4. $queue_{k'} \leftarrow queue_{k'} \cup (n+1); D_{i+((\#r_n-1)/2)}$.
  end;
end.

Theorem 3. *The algorithm works correctly as specified, producing all the rows in the Göttfert algorithm required to achieve a complete factorization.* As a result, *the algorithm requires at most* $3s$ *steps for a list of size* s.

Proof: see [8].

3.1 The Parallel Göttfert Algorithm

One major characteristic of the algorithm is that it consists mainly of task parallelism, since distributing the data would require much more synchronization between processors in the inner loops than would be the case in our present

algorithm. The polynomials are represented by integer arrays whose entries are either zero or one and where the coefficients are packed into bit-words (where w_l is the bit-size of the computer word being used). We refer the reader to our report in [8] for a detailed description of the data structures used. Unless otherwise stated, all arrays are global. The arrays *Type, Row*, and *Index* are embedded within two queues $queue_j$ and $queue'_j$; $queue_j$ is a sequence of triples $(Type_j[i], Row_j[i], Index_j[i])$, for $i = 0, ..., \#(queue_j) - 1$, and each such triple describes a polynomial already computed in some parallel queue. On the other hand, $queue_{j'}$ consists of similar triples describing polynomials to be computed in a forthcoming parallel queue.

Algorithm 2 *Parallel_Göttfert*
Input: f *a polynomial of degree* d *over* F_2, $m > 1$ *the number of irreducible factors of* f, $\{h_0, ..., h_{m-1}\}$ *a basis for the solution set of the linear system,* $\{b_0, ..., b_{m-1}\}$ *the corresponding set of squarefree factors of* f *defined by* $b_i = f/\gcd(f, h_i)$ *for* $i = 0, ..., m-1$, p *the total number of processors operating in parallel, and id the processor identification number ranging from* $0, ..., p-1$.
Output: *the* m *irreducible factors of* f.

```
1. P_0 ← gcd(b_0, b_1);
if (P_0 ≠ 1) do
2.   Store_value(P_0), Update(length(2));
end;
3. queue_j ← {P_0}, Set_Queues(queue_j, queue_j');
while not all m irreducible factors have been found do
4.   k ← id;
     while (k < queue_length) do
5.        P_k ← Compute_Polynomial(Type, row, index, k);
          if (P_k ≠ 1) do
6.             Store_value(P_k), n ← row(P_k),
               Update(partial_length(n)), Broadcast_value(P_k),
               Broadcast(partial_length(n));
7.        k ← k + p;
          end;
8.   BSP_synchronize();
9.   for i ∈ {2, ..., m} do
          Assemble(partial_lengths(i));
     end;
     if (not all irreducible factors have been found) do
10.       queue_j ← queue_j', Sort(queue_j),
          Set_Queues(queue_j, queue_j');
     end;
end.
```

For elaborate details describing the algorithm we refer the reader to [8]. In this paper, we give a brief description of the main steps comprising the process above. The algorithm is called by all processors which implement the same copy

of it for various data . The second parallel queue consists of the polynomial $2; D_1$ (see Algorithm 1) which is computed by all processors. If $P_0 = 2; D_1$ is not trivial, it is stored permanently, and the length of row 2 is updated. We call Algorithm 1 to set up the ensuing $queue_{j'}$ of polynomials to be computed in parallel. Thereafter, the main loop of the algorithm is iterated so long as none of the rows has attained m non-constant polynomials. The variable k, which loops over indices in $queue_{j'}$, is a global variable which, when first set to *id* and then incremented by p, guarantees that all processors compute almost an equal number of polynomials. The processors receive information about the polynomials they should compute through the global data found in $Type = Type_{j'}[k]$, $n = Row_{j'}[k]$, and $i = Index_{j'}[k]$, and embedded within $queue_{j'}$. They then call the sub-routine *Compute_Polynomial* which determines the polynomial P_k as defined in the Göttfert setting. If P_k is non-constant, processor *id* stores it permanently and updates its own local value of $length(n)$. When all computations in $queue_{j'}$ have been performed, a synchronization barrier is met, which updates the values of the non-constant polynomials and the partial lengths of rows as computed by every individual processor. We note the absence of a synchronization point immediately after the broadcasting of the non-constant polynomials due to the fact that they were not needed in any computation within the innermost loop of the algorithm. Also, although updating the total row lengths inside the innermost loop definitely discards any unnecessary gcd or division computations remaining in the queue, our choice not to perform accordingly can be justified by the fact that this will require a synchronization point within the innermost loop, one whose repeated application could prove to be expensive. All processors then assemble the partial lengths of all rows as computed by the relevant processors which have contributed in non-constant polynomials. If any row length becomes equal to m, all processors are signalled to stop. Else, $queue_{j'}$ is transferred onto $queue_j$ (so that the most recent polynomials can help determine what the new parallel queue will be), and $queue_j$ is sorted through a call to *Sort*. Since some processors compute constant polynomials whose index k leaves the corresponding location in memory empty, the *Sort* sub-routine re-arranges them (and their corresponding pointers in the arrays *D* or *R*) so that the non-constant factors are stored consecutively after each other. *Sort* also returns the length of the sorted list. Finally, a new $queue_{j'}$ is set according to Algorithm 1. The outermost loop can be shown to end, since we are bound to reach a row containing all m non-constant irreducible factors.

3.2 The BSP Cost of the Algorithm

Before discussing the parallel complexity of our algorithm, we derive several preliminary results, whose complete proofs can be found in [8].

Theorem 4. *In the parallel setting described in algorithm 1, every row* n *has its first element* $n; D_1$ *computed in the parallel queue* n *and its last element* $n; R_{\#r_{n-1}+1}$ *computed in the parallel queue* $2n - 1$.

Corollary 1. *It takes at most* $2m-1$ *parallel queues for a complete factorization into irreducibles to be established.*

Theorem 5. *If* n *is odd, then queue* n *contains polynomials belonging only to rows* $(n+1)/2+j$, for $j=0,...,(n-1)/2$, if $2 \leq n \leq m$, *and for* $j=0,...,m-(n+1)/2$, if $m < n \leq 2m-1$. *Else*, if n is even, *then queue* n *contains polynomials belonging only to rows* $n/2+1+j$, for $j=0,...,n/2-1$, *if* $2 \leq n \leq m$, *and for* $j=0,...,m-(n/2+1)$, *if* $m < n \leq 2m-1$.

Theorem 6. *Each parallel queue consists of at most* $2m$ *gcd and division operations and contributes to at most* m *non-constant polynomials.*

Theorem 7. *The BSP cost of algorithm 2 is of the order* $O\left(\frac{m^2}{p}M(d)\log d + gm^2\left\lceil\frac{d}{w_l}\right\rceil + m\ell\right)$ *flops.*

Corollary 2. *Algorithm 2 has low synchronization and communication requirements.*

4 Implementation and Run Times

For a full report on our parallel performance the reader can refer to [8]. Our run times in table 1 of [8] suggest a speed gain in almost all cases, an outcome that is to be expected as a result of the negligible communication and synchronization requirements of our algorithm. The efficiencies demonstrate that almost all our experiments scale very well up to 8 processors. Thereafter, the efficiency remains very good either as d increases or as m increases. Efficiency also remains almost constant around 1 for $256000 \leq d \leq 400000$. We remark the absence of a sharp fluctuation in the efficiency levels, mainly because our algorithm does not involve data partitioning (but only task parallelism), which results in the computation being either entirely *in cache* or *out of cache* across all processors for the same d. This has the advantage of revealing the real scalability of the algorithm and avoiding cache effects. We expect our algorithm to continue scaling well as d increases more considerably than what is actually reported in this paper, and experiments related to the output of the algorithm in [9] for solving large Niederreiter linear systems for trinomials over F_2 are currently under-way.

5 Conclusion

In this paper we presented and analyzed a complete BSP algorithm for extracting the factors of a polynomial over F_2 using the Göttfert refinement of the Niederreiter algorithm, which, given a basis for the solution set of the Niederreiter linear system, performs the last phase of the factorization algorithm in polynomial time. Our BSP theoretical model resulted in an efficient BSP cost requiring relatively small communication and synchronization costs. The parallel

algorithm not only achieves considerable speed gains as the number of processors increases up to 16, but maintains a moderate to high efficiency that is better maintained as the degree of the polynomial or the number of its irreducible factors increases. The algorithm can be applied over fields of characteristic 2 in general, provided an input basis is available. When combined with our work in [9] which exploits sparsity in the Niederreiter linear system, the hybrid algorithm provides a cheaper and more memory efficient alternative to the factorization of trinomials over F_2 than the implementation in [10], which uses dense explicit linear algebra and a maximum of 256 nodes to achieve a polynomial record of degree 300000. When compared with the Black Box Niederreiter algorithm of [11], the hybrid algorithm is a simpler approach for moderately high record factorizations of sparse polynomials over F_2, requiring reasonable running times [9]. Apart from the significance of its experimental results, our algorithm provides a good model of how parallelism in general, and the BSP model in particular, can be incorporated elegantly and successfully into problems in symbolic computation.

Acknowledgements. The author is grateful to the Oxford Supercomputing Centre for allowing the use of its facilities to generate the reported experiments.

References

1. Niederreiter, H.: "A New Efficient Factorization Algorithm for Polynomials over Small Finite Fields", *AAECC* , Vol. 4, 1993, pp. 81-87.
2. Göttfert, G.: "An Acceleration of the Niederreiter Factorization Algorithm in Characteristic 2", *Math. Comp*., Vol. 62, 1994, pp. 831-839.
3. Niederreiter, H.: "Factorization of Polynomials and some Linear Algebra Problems over Finite Fields", *Lin. Alg. and its App*., Vol. 192, 1993, pp. 301-328.
4. Niedrreiter, H.: "Factoring Polynomials over Finite Fields Using Differential Equations and Normal Bases", *Mathematics of Computation*, vol. 62, 1994, pp. 819-830.
5. Valiant, L. G.: "A Bridging Model for Parallel Computation", *Comm. of the ACM*, Vol. 33, 1990, pp. 103-111.
6. Hill, J. M. D., McColl, W. F., and Skillicorn, D. B.: "Questions and Answers about BSP", *Report PRG-TR-15-96*, Oxford University Computing Laboratory, 1996.
7. Hill, J. M. D., McColl, W. F., Stefanescu, D. C., Goudrea, M. W., Lang, K., Rao, S. B., Suel, T., Tsantilas, T., Bisseling, R. H.: "BSPlib: The BSP Programming Library", *Parallel Computing*, Vol. 24, 1998, pp. 1947-1980.
8. Abu Salem, F: "A BSP Parallel Model of the Göttfert Algorithm for Polynomial Factorization over F_2", Report PRG-RR-03-14, Oxford University Computing Laboratory, July 2003.
9. Abu Salem, F: "A New Sparse Gaussian Elimination Algorithm and the Niederreiter Linear System for Trinomials over F_2", Report PRG-RR-03-18, Oxford University Computing Laboratory, August 2003.
10. Roelse, P.: "Factoring High-Degree Polynomials over F_2 with Niederreiter's Algorithm on the IBM SP2", *Math. Comp*., Vol. 68, 1999, pp. 869-880.
11. Fleischmann, P., Holder, M., and Roelse, P.: "The Black-Box Niederreiter Algorithm and its Implementation over the Binary Field", *Math. Comp*., Vol. 72, 2003, pp. 1887-1899.

Parallelizing the Unsupervised k-Windows Clustering Algorithm

Panagiotis D. Alevizos[1,2], Dimitris K. Tasoulis[1,2], and Michael N. Vrahatis[1,2]

[1] Department of Mathematics, University of Patras, GR-26500 Patras, Greece
{alevizos, dtas, vrahatis}@math.upatras.gr
[2] University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR-26500 Patras, Greece

Abstract. Clustering can be defined as the process of partitioning a set of patterns into disjoint and homogeneous meaningful groups, called clusters. The growing need for parallel clustering algorithms is attributed to the huge size of databases that is common nowadays. This paper presents a parallel version of a recently proposed algorithm that has the ability to scale very well in parallel environments mainly regarding space requirements but also gaining a time speedup.

1 Introduction

Clustering, that is the partitioning a set of patterns into disjoint and homogeneous meaningful groups (clusters), is a fundamental process in the practice of science. In particular, clustering is fundamental in knowledge acquisition. It is applied in various fields including data mining [6], statistical data analysis [1], compression and vector quantization [15]. Clustering is, also, extensively applied in social sciences.

The task of extracting knowledge from large databases, in the form of clustering rules, has attracted considerable attention. Due to the ever increasing size of databases there is also an increasing interest in the development of parallel implementations of data clustering algorithms. Parallel approaches to clustering can be found in [9,10,12,14,16].

Exploiting recent software advances [7,11], collections of heterogeneous computers can be used as a coherent and flexible concurrent computational resource. These technologies have allowed the vast number of individual Personal Computers available in most scientific laboratories to be used as parallel machines at no, or at a very low cost. Network interfaces, linking individual computers, are necessary to produce such pools of computational power. In many such cases the network infrastructure comprises a bottleneck to the entire system. Thus applications that exploit specific strengths of individual machines on a network, while minimizing the required data transfer rate are best suited for network–based environments.

The results reported in the present paper indicate that the recently proposed k-windows algorithm [17] has the ability to scale very well in such environments.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 225–232, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The k-windows algorithm endogenously determines the number of clusters. This is a fundamental issue in cluster analysis, independent of the particular technique applied.

The paper is organized as follows; Section 2 describes briefly the workings of the k-windows algorithm; Section 3 discusses the parallel implementation of the algorithm; while Section 4, reports the results of the experiments conducted. The paper closes with concluding remarks and a short discussion about further research directions.

2 The Unsupervised k-Windows Algorithm

The unsupervised k-windows algorithm is a straightforward generalization of the original algorithm [17], by considering a large number of initial windows. The main idea behind k-windows is to use windows to determine clusters. A window is defined as an orthogonal range in d-dimensional Euclidean space, where d is the number of numerical attributes. Therefore each window is a d-range of initial fixed area a.

Intuitively, the algorithm tries to place a window containing all patterns that belong to a single cluster; for all clusters present in the dataset. At a first stage, the windows are moved in the Euclidean space without altering their area. Each window is moved by setting its center to the mean of the patterns it currently includes (sea solid line squares in Fig. 1. This process continues iteratively until further movement does not increase the number of patterns included. At the second stage, the area of each window is enlarged in order to capture as many patterns of the corresponding cluster as possible. The process of enlargement of a window terminates when the number of patterns included no longer increases.

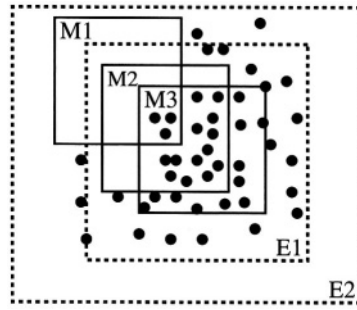


Fig. 1. Sequential Movements (M1 M2 M3 solid lines) and sequential enlargements (E1 E2 dashed lines) of a window.

In more detail; at first, k means are selected (possibly in a random manner). Initial d-ranges (windows), of area a, have as centers those initial means.

Then, the patterns that lie within each d-range are found, using the Orthogonal Range Search technique of Computational Geometry [2,4,5,8,13]. The latter technique has been shown to be effective in numerous applications and a considerable amount of work has been devoted to this problem [13]. An orthogonal range search is based on a pre–process phase where a *range tree* is constructed. Patterns that lie within a d-range can be found traversing the range tree, in polylogarithmic time. The *orthogonal range search* problem can be stated as follows:

- **Input:**
  a) $V = \{p_1, \ldots, p_n\}$ is a set of n points in $\mathbb{R}^d$ the d-dimensional Euclidean space with coordinate axes $(Ox_1, \ldots, Ox_d)$,
  b) a query d-range $\mathcal{Q} = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ is specified by two points $(a_1, a_2, \ldots, a_d)$ and $(b_1, b_2, \ldots, b_d)$, with $a_j \leqslant b_j$.

- **Output:**
  report all points of V that lie within the d-range $\mathcal{Q}$.

Having identified the patterns that lie within each d-range, their mean is calculated. The mean defines the new center for the d-range, which implies the movement of the d-range. The last two steps are executed repeatedly, as long as the number of patterns included in the d-range increases as a result of the movement.

Subsequently, the d-ranges are enlarged in order to include as many patterns as possible from the cluster. The enlargement process terminates if further enlargement does not increase the number of patterns included in the window. Enlargement and movement are repeatedly executed until both processes do not yield an increase in the number of patterns in the window.

Then, the relative frequency of patterns assigned to a d-range in the whole set of patterns, is calculated. If the relative frequency is small, then it is possible that a missing cluster (or clusters) exists. Thus, the whole process is repeated.

The key idea to determine the number of clusters automatically is to apply the k-windows algorithm using a sufficiently large number of initial windows. The windowing technique of the k-windows algorithm allows for a large number of initial windows to be examined, without any significant overhead in time complexity. Then, any two overlapping windows are merged, before the step of enlarging the windows is performed. The remaining windows, after the quality of the partition criterion is met, define the final set of clusters.

3 Parallel Implementation

At present the majority of databases are spread over numerous servers each one holding its own data. The proposed parallel implementation of k-windows is taking into consideration this situation. So the parallelism is mostly a storage space parallelism. For this task we propose a parallel algorithmic scheme that uses a **multidimensional binary tree** [3] for range search.

Let us consider a set $V = \{p_1, p_2, \ldots, p_n\}$ of n points in d-dimensional space $\mathcal{R}^d$ with coordinate axes $(Ox_1, Ox_2, \cdots, Ox_d)$. Let $p_i = (x_1^i, x_2^i, \cdots, x_d^i)$ be the representation of any point p_i of V.

Definition: Let V_s be a subset of the set V. The *middle point* p_h of V_s with respect to the coordinate x_i $(1 \leqslant i \leqslant d)$ is defined as the point which divides the set V_s-$\{p_h\}$ into two subsets V_{s_1} and V_{s_2}, such that:

i) $\forall p_g \in V_{s_1}$ and $\forall p_r \in V_{s_2}$, $x_i^g \leqslant x_i^h \leqslant x_i^r$.
ii) V_{s_1} and V_{s_2} have approximately equal numbers of elements: If $|V_s| = t$ then $|V_{s_1}| = \lceil \frac{t-1}{2} \rceil$ and $|V_{s_2}| = \lfloor \frac{t-1}{2} \rfloor$.

The **multidimensional binary tree** T which stores the points of the set V is constructed as follows:

1. Let p_r be the *middle point* of the given set V, with respect to the first coordinate x_1. Let V_1 and V_2 be the corresponding partition of the set V-$\{p_r\}$. The point p_r is stored in the root of T.
2. Each node p_i of T, obtains a left child $left[p_i]$ and a right child $right[p_i]$ as follows: MBT(p_r,V_1,V_2,1)

procedure MBT(p,L,M,k)
begin
$k \longleftarrow k + 1$
if $k = d + 1$ **then** $k \longleftarrow 1$
if $L \neq \emptyset$ **then**
begin
let u be the middle point of the set L with respect to the coordinate x_k.
The point u divides the set L-$\{u\}$ in two subsets L_1 and L_2.
$left[p] \longleftarrow u$
MBT(u,L_1,L_2,k)
end
if $M \neq \emptyset$ **then**
begin
let w be the middle point of the set M with respect to the coordinate x_k and let M_1 and M_2 be the corresponding partition of the set M-$\{w\}$.
$right[p] \longleftarrow w$
MBT(w,M_1,M_2,k)
end
end

Let us consider a query d-range $\mathcal{Q} = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$ specified by two points $(a_1, a_2, \ldots, a_d)$ and $(b_1, b_2, \ldots, b_d)$, with $a_j \leqslant b_j$. The search of the tree T is performed through the following algorithm, which accumulates the retrieved points in a list $\mathcal{A}$, initialized as empty:

The orthogonal range search algorithm

1) $\mathcal{A} \longleftarrow \emptyset$
2) Let p_r be the root of T: SEARCH(p_r,$\mathcal{Q}$,$\mathcal{A}$,1)
3) return $\mathcal{A}$

procedure SEARCH(p_t,$\mathcal{Q}$,$\mathcal{A}$,i)
begin
  if $i = d + 1$ **then** $i \longleftarrow 1$
    let $p_t = (x_1^t, x_2^t, \ldots, x_d^t)$
  if $a_i \leqslant x_i^t \leqslant b_i$ **then if** $p_t \in \mathcal{Q}$ **then** $\mathcal{A} \longleftarrow \mathcal{A} \cup \{p_t\}$
  if $p_t \neq leaf$ **then**
  begin
    if $a_i < x_i^t$ **then** SEARCH($left[p_t]$,$\mathcal{Q}$,$\mathcal{A}$,$i + 1$)
    if $x_i^t < b_i$ **then** SEARCH($right[p_t]$,$\mathcal{Q}$,$\mathcal{A}$,$i + 1$)
  end
end

The proposed parallel implementation uses the aforementioned range search algorithm and is a Server–Slave model. Assume m computer nodes are available, each one having a portion of the dataset V_i where $i = 1, \ldots, m$. Firstly at each node i a **multidimensional binary tree** T_i is constructed using the MBT algorithm, which stores the points of the set V_i. Then parallel search is performed as follows:

The parallel orthogonal range search algorithm

1) $\mathcal{A} \longleftarrow \emptyset$
2) For each node i do
3) $A_i \longleftarrow \emptyset$
4) Let $p_{r,i}$ be the root of T_i: SEARCH(p_{r_i},$\mathcal{Q}$,A_i)
5) $\mathcal{A} \longleftarrow \mathcal{A} \cup A_i$
6) end do
7) return $\mathcal{A}$

More specifically, the algorithm at a preprocessing step constructs a **multidimensional binary tree** for each node holding data known only to that node.

Then a server node is used to execute the k-windows algorithm. From that point onward the algorithm continues to work normally. When a range search is to be executed, the server spawns the range query over all the nodes and computes the union of the results.

The algorithmic complexity for the preprocessing step for n points in d-dimensions is reduced to $\theta(\frac{dn \log n}{m})$ from $\theta(dn \log n)$ of the single node version [13]. Furthermore the storage requirements at each node come up to $\theta(\frac{dn}{m})$ while for the single node remain $\theta(dn)$ Since the **orthogonal range search algorithm** has a complexity of $O(dn^{1-\frac{1}{d}} + k)$ [13], the **parallel orthogonal range search algorithm** has a complexity of $O(d\,(\frac{n}{m})^{1-\frac{1}{d}} + k + \epsilon(d,m))$, where k is the total number of points included in the range search and $\epsilon(d,m)$ is a func-

tion that represents the time that is required for the communication between the master and the nodes. It should be noted that the only information that needs to be transmitted from each slave is the number of points found and their mean value as a d-dimensional vector. So the total communication comes to a broadcast message from the server about the range, and m messages of an integer and a d-dimensional vector from each slave. Taking these under consideration, the $\epsilon(d, m)$ can be computed for a specific network interface and a specified number of nodes. For the parallel algorithm to achieve an execution time speedup the following relation must hold:

$$O\left(\frac{d\,(\frac{n}{m})^{1-\frac{1}{d}} + k + \epsilon(d,m)}{dn^{1-\frac{1}{d}} + k}\right) \leqslant 1,$$

which comes to:

$$O(\epsilon(d,m)) \leqslant O\left(d\,\left(n^{1-\frac{1}{d}} - \left(\frac{n}{m}\right)^{1-\frac{1}{d}}\right)\right). \tag{1}$$

As long as Inequality (1) holds, the parallel version of the algorithm is faster than the single node version. In any other case the network infrastructure presents a bottleneck to the whole system that can not be overcome. In that case the parallel version advantage is limited to storage space requirements.

4 Results

The k-windows clustering algorithm was developed under the Linux operating system using the C++ programming language. Its parallel implementation was based on the PVM parallel programming interface. PVM was selected, among its competitors because any algorithmic implementation is quite simple, since it does not require any special knowledge apart from the usage of functions and setting up a PVM daemon to all personal computers, which is trivial.

The hardware used for our purposes consisted of 16 Pentium III personal computers with 32MB of RAM and 4GB of hard disk availability each. A Pentium 4 personal computer with 256MB of RAM and 20GB of hard disk availability was used as a server for the algorithm, while the network infrastructure was a Fast Ethernet 100MBit/s network.

To measure the efficiency of the algorithm, two datasets were used, namely Dset1 and Dset2, that represent a single image stored and displayed in the RGB space, with 2 different scalings. The datasets contained approximately 3^{10}, and 4^{10} number of points, respectively, that correspond to the number of pixels in each image scaling. Since the color of each pixel follows red/green/blue (RGB) color specification (three numbers between 0 and 255 indicating red, green, and blue), each datapoint is represented by a three-dimensional vector, corresponding to its RGB values. In Table 1 the speedup achieved for different number of slave nodes is exhibited. It is evident from this table, that the speedup achieved for Dset2 is greater than the speedup for Dset1. This is also suggested

Table 1. Speedup achieved for Dset1 and Dset2

Number of Nodes	Speedup for Dset1	Dset2
2	1.0000	1.0000
4	1.4643	1.7801
8	2.5949	2.6421
16	4.2708	4.7358

by Equation (1). Furthermore we constructed a random dataset using a mixture of Gaussian random distributions. The dataset contained 21000 points with 50 numerical attributes. The points were organized in 4 clusters (small values at the covariance matrix) with 2000 points as noise (large values at the covariance matrix). To test this dataset, we stored the binary tree to the hard disk of each node. Thus, each search required much more time compared to the previous cases. As it is exhibited in Fig. 2, for this dataset the algorithm achieves almost 9 times smaller running time when using 16 CPUs. On the other hand at every node only the 1/16 of the total storage space is required. From Fig. 2, we also observe an abrupt slow–down in speedup when moving from 8 to 16 nodes. This behavior is due to the larger number of messages that must be exchanged during the operation of the algorithm which results to increased network overhead.

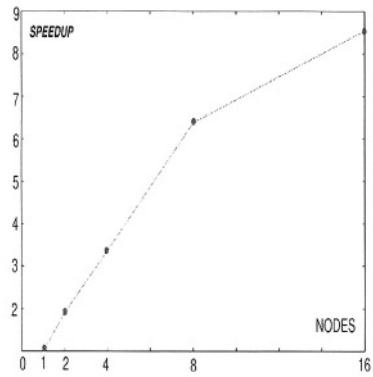


Fig. 2. Speedup for the different number of CPUs

5 Conclusions

Clustering is a fundamental process in the practice of science. Due to the increasing size of current databases, constructing efficient parallel clustering algorithms has attracted considerable attention. The present study presented the parallel version of a recently proposed algorithm, namely the k-windows. The specific algorithm is characterized by the highly desirable property that the number

of clusters is not user defined, but rather endogenously determined during the clustering process. The parallel version proposed is able to achieve considerable speedup in running time, and at the same time it attains a linear decrease on the storage space requirements with respect to the number of computer nodes comprising the PVM.

References

1. M. S. Aldenderfer and R. K. Blashfield, *Cluster Analysis,* in Series: Quantitative Applications in the Social Sciences, SAGE Publications, London, 1984.
2. P. Alevizos, *An Algorithm for Orthogonal Range Search in* $d \geqslant 3$ *Dimensions,* Proceedings of the 14th European Workshop on Computational Geometry, Barcelona, 1998.
3. P. Alevizos, B. Boutsinas, D. Tasoulis, M.N. Vrahatis, *Improving the Orthogonal Range Search k-windows Clustering Algorithm,* Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence, Washington D.C. 2002 pp.239-245.
4. J.L. Bentley and H.A. Maurer, *Efficient Worst-Case Data Structures for Range Searching,* Acta Informatica, 13, 1980, pp. 1551-68.
5. B. Chazelle, *Filtering Search: A New Approach to Query-Answering,* SIAM J. Comput., 15, 3, 1986, pp.703-724.
6. U. M. Fayyad, G. Piatetsky-Shapiro and P. Smyth, *Advances in Knowledge Discovery and Data Mining,* MIT Press, 1996.
7. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing,* MIT Press, Cambridge, 1994.
8. B. Chazelle and L. J. Guibas, *Fractional Cascading: II. Applications,* Algorithmica, 1, 1986, pp.163-191.
9. D. Judd, P. McKinley, and A. Jain, *Large-Scale Parallel Data Clustering,* Proceedings of the Int. Conference on Pattern Recognition, 1996.
10. D. Judd, P. McKinley, A. Jain, *Performance Evaluation on Large-Scale Parallel Clustering in NOW Environments,* Proceedings of the Eight SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, March 1997.
11. *MPI The Message Passing Interface standard,* http://www-unix.mcs.anl.gov/mpi/.
12. C.F. Olson, *Parallel Algorithms for Hierarchical Clustering,* Parallel Computing, 21:1313- 1325, 1995.
13. F. Preparata and M. Shamos, *Computational Geometry,* Springer Verlag, 1985.
14. J.T. Potts, *Seeking Parallelism in Discovery Programs,* Master Thesis, University of Texas at Arlington, 1996.
15. V. Ramasubramanian and K. Paliwal, *Fast k-dimensional Tree Algorithms for Nearest Neighbor Search with Application to Vector Quantization Encoding,* IEEE Transactions on Signal Processing, 40(3), pp.518-531, 1992.
16. K. Stoffel and A. Belkoniene, *Parallel k-means Clustering for Large Data Sets,* Proceedings Euro-Par '99, LNCS 1685, pp. 1451-1454, 1999.
17. M. N. Vrahatis, B. Boutsinas, P. Alevizos and G. Pavlides, *The New k-windows Algorithm for Improving the k-means Clustering Algorithm,* Journal of Complexity, 18, 2002 pp. 375-391.

Parallel Simulated Annealing for Bicriterion Optimization Problems

Piotr Czarnas[1], Zbigniew J. Czech[2]*, and Przemyslaw Gocyła[3]

[1] University of Wrocław, Wrocław, Poland
piotr.czarnas@finexaq.com
[2] Silesia University of Technology, Gliwice, and University of Silesia, Sosnowiec, Poland
zjc@polsl.gliwice.pl
[3] University of Silesia, Sosnowiec, Poland
bld2@poczta.fm

Abstract. A parallel simulated annealing algorithm to solve the vehicle routing problem with time windows is proposed. It is a complex bicriterion optimization problem in which both the number of vehicles and the total distance traveled by vehicles should be minimized. The aim is to establish the best possible solutions to the well-known instances of the problem by using parallelism. The empirical tests show that parallel simulated annealing can solve effectively bicriterion optimization problems.

Keywords. Parallel simulated annealing, bicriterion optimization, vehicle routing problem with time windows.

1 Introduction

The vehicle routing problem with time windows (VRPTW) analyzed in this work consists in establishing a set of routes beginning and ending at a depot which serves a set of customers. For the purpose of delivery (or pick up) there is a limited set of vehicles, each vehicle of some capacity. The customers have the given delivery demands and a vehicle on its route cannot serve more customers than its capacity allows. For each customer a time interval, called the time window, and a time of service are defined. The aim is to find a set of routes which visits each customer exactly once, ensures that the service at any customer begins within the time window and preserves the vehicle capacity constraints. In addition, the set of routes should minimize, firstly, the number of vehicles used, and secondly, the total distance traveled by the vehicles. Thus the VRPTW is a bicriterion optimization problem. The practical applications of the VRPTW include deliveries of goods to department stores, school bus routing, newspaper, laundry and mail distribution, security patrol or maintenance services, etc.

* This research was supported in part by the State Committee for Scientific Research grant BK-279-RAu2-2002.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 233–240, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The previous works on the VRPTW can be divided into two groups, exact optimization and heuristic (or approximate) algorithms. The methods in the first group are able to solve to optimality some of the Solomon (1987) benchmark problems of size up to 100 customers. In the second group a variety of metaheuristics to solve the VRPTW can be found. Among them are local search, genetic algorithms, tabu search, evolution strategies, ant colony systems, constraint programming and simulated annealing.

In this work a parallel simulated annealing algorithm to solve the VRPTW is presented. The aim is to establish the best possible solutions to Solomon (1987) instances of this problem by using parallelism. The empirical tests show that parallel simulated annealing can solve effectively bicriterion optimization problems. To our knowledge parallel simulated annealing to solve the VRPTW was not reported in the literature.

In section 2 the problem under analysis is formulated. Section 3 describes a sequential annealing algorithm. In section 4 the parallel simulated annealing algorithm is proposed. Section 5 describes the empirical results. Section 6 concludes the work.

2 Problem Formulation

The vehicle routing problem with time windows which is a modification of the well-known vehicle routing problem (VRP) can be formulated as follows. There is a central depot of cargo and n customers (nodes) located at the specified distances from the depot. The locations of the depot ($i = 0$) and the customers ($i = 1, 2, \ldots, n$), and the shortest distances d_{ij} and the corresponding travel times t_{ij} between any two locations i and j are given. The cargo have to be delivered to (or picked up from) each customer i according to the delivery demand q_i by a fleet of vehicles. Each vehicle serves a subset of customers on the route which begins and ends at the depot. The vehicles have the same capacity Q. The total sum of demands of customers served by a vehicle on a route cannot exceed Q. For each customer a service time window $[e_i, f_i]$ and a service time s_i are defined. e_i and f_i determine, respectively, the earliest and the latest time for start servicing. The customer i is served by a single vehicle exactly once, within the time window $[e_i, f_i]$. The vehicle can arrive at the customer before the time window but in such a case it has to wait until time e_i when the service can begin. The latest time for arrival of the vehicle at customer i is f_i. Let b_i be the current time when service can begin at customer i, and let b_{ij} be the current time when service can begin at customer j, given the customer j is visited immediately after customer i in the route. Then we have $b_{ij} = \max\{e_j, b_i + s_i + t_{ij}\}$. A wait time $w_j = e_j - (b_i + s_i + t_{ij})$ is required if a vehicle arrives at customer j before e_j. The objective is to find the set of routes which guarantees the delivery of cargo to all customers and satisfies the time window and vehicle capacity constraints. Furthermore, the size of the set equal to the number of vehicles needed (primary goal) and the total travel distance (secondary goal) should be minimized.

3 Sequential Simulated Annealing

The algorithm of simulated annealing can be regarded as a variant of local search. The application of simulated annealing to solve the VRPTW is as follows. Initially a solution to the problem is taken as the best solution to the problem known so far, or the solution found using some heuristics (Gocyła 2002). On every step a neighbor solution is determined by moving one or more customers among the routes. Both the customers and the routes are chosen randomly. The neighbor solutions of lower costs obtained in this way are always accepted. The solutions of higher costs are accepted with the probability

$$e^{-\frac{\delta}{T_i}} \tag{1}$$

where T_i, $i = 0, 1, \ldots, i_{\max}$, is a parameter called a temperature of annealing, which falls from the initial value $T_0 = \gamma \cdot cost(s_0)$ according to the formula $T_{i+1} = \beta T_i$, where γ and $\beta < 1$ are constants and s_0 is the initial solution to the problem, and δ denotes the increase in the solution cost. Equation (1) implies that large increases in solution cost, so called uphill moves, are more likely to be accepted when T_i is high. As T_i approaches zero most uphill moves are rejected. The cost of solution s is computed as follows:

$$cost(s) = c_1 N + c_2 D \tag{2}$$

where N is the number of routes in solution s (equals to the number of vehicles needed), D is the total travel distance of the routes, and c_1, c_2 are some constants. Since the basic criterion of optimization is the number of routes, usually it is assumed that $c_1 \gg c_2$.

The sequential algorithm of annealing halts if a stopping condition is satisfied. In this work we assume that the condition is satisfied if a falls of the temperature of annealing, so called cooling stages, are executed. Contrary to the classical approach in which a solution to the problem is taken as the last solution obtained in the annealing process, we memorize the best solution found during the whole annealing process.

Summing up, the simulated annealing algorithm performs the local search by sampling the neighborhood randomly. It attempts to avoid becoming prematurely trapped in a local optimum by sometimes accepting an inferior solution. The level of this acceptance depends on the magnitude of the increase in solution cost and on the search time to date.

4 Parallel Simulated Annealing

Let us assume that p processes can be executed and each of them is capable of generating its own annealing chain. The processes can be used either to speed up the sequential annealing algorithm or to achieve a higher accuracy of solutions to a problem. In this work we consider the latter goal. The accuracy of a solution is meant as its proximity to the global optimum solution.

Table 1. The test results for the C10x and C20x data sets (N — number of vehicles needed; D — total distance traveled). The best known solutions are marked in bold.

Probl. inst.	Best published solution			Best comp. solution		Probl. inst.	Best published solution			Best comp. solution	
	N	D	Ref.	N	D		N	D	Ref.	N	D
C101	**10**	**828.94**	RT-95	**10**	**828.94**	C201	**3**	**591.56**	RT-95	**3**	**591.56**
C102	**10**	**828.94**	RT-95	**10**	**828.94**	C202	**3**	**591.56**	RT-95	**3**	**591.56**
C103	**10**	**828.06**	RT-95	**10**	**828.06**	C203	**3**	**591.17**	RT-95	**3**	**591.17**
C104	**10**	**824.78**	RT-95	**10**	**824.78**	C204	**3**	**590.60**	RT-95	**3**	**590.60**
C105	**10**	**828.94**	RT-95	**10**	**828.94**	C205	**3**	**588.88**	RT-95	**3**	**588.88**
C106	**10**	**828.94**	RT-95	**10**	**828.94**	C206	**3**	**588.49**	RT-95	**3**	**588.49**
C107	**10**	**828.94**	RT-95	**10**	**828.94**	C207	**3**	**588.29**	RT-95	**3**	**588.29**
C108	**10**	**828.94**	RT-95	**10**	**828.94**	C208	**3**	**588.32**	RT-95	**3**	**588.32**
C109	**10**	**828.94**	RT-95	**10**	**828.94**						

In the parallel simulated annealing algorithm the processes $P_1, P_2, \ldots, P_p$ co-operate with each other every ω step passing their best solutions found so far. Suppose for a moment that the temperature of annealing, T, is fixed. Let $V_r^{(j)}(T)$, $j = 1, 2, \ldots, p$, $r = 1, 2, \ldots, r_{\max}$ be the Markov chain for each of the processes, let $\mathcal{P}_T(V)$ be a realization of one step of the chain at temperature T and with starting point V, and let $\bar{V}_r^{(j)}$ be the best solutions found by processes $j = 1, 2, \ldots, p$, so far, i.e. between step 1 and r. We assume the following scheme of co-operation:

$$V_{r+1}^{(1)} = \mathcal{P}_T(V_r^{(1)}), \tag{3}$$

$$V_{r+1}^{(j)} = \mathcal{P}_T(V_r^{(j)}) \quad \text{for } j \neq 1, \text{ and if } r+1 \neq u\omega, \tag{4}$$

$$V_{u\omega}^{(j)} = \mathcal{P}_T(V_{u\omega-1}^{(j)}) \quad \text{if } cost(\mathcal{P}_T(V_{u\omega-1}^{(j)})) \leq cost(\bar{V}_{u\omega}^{(j-1)}), \tag{5}$$

$$V_{u\omega}^{(j)} = \bar{V}_{u\omega}^{(j-1)} \quad \text{otherwise.} \tag{6}$$

In this scheme the processes co-operate at steps $u\omega$, $u = 1, 2, \ldots, u_{\max}$, where each step consists of a single realization in the Markov chain, i.e. of an annealing step. The chain for the first process ($j = 1$) is completely independent. The chain for the second process is updated at steps $u\omega$ to the better solution between the best solution found by the first process so far, $\bar{V}_{u\omega}^{(1)}$, and the realization of the last step of the second process, $\mathcal{P}_T(V_{u\omega-1}^{(2)})$. Similarly, the third process chooses as the next point in its chain the better solution between $\bar{V}_{u\omega}^{(2)}$ and $\mathcal{P}_T(V_{u\omega-1}^{(3)})$. Clearly, the best solution found by the l-th process is propagated for further exploration to processes m, $m > l$. The Markov chains generated by the processes are non-homogeneous since the probability of moving from one solution to another depends not only on the costs of these solutions and the

Table 2. The test results for the R10x and R20x data sets.

Probl. inst.	Best published solution			Best comp. solution		Probl. inst.	Best published solution			Best comp. solution	
	N	D	Ref.	N	D		N	D	Ref.	N	D
R101	**19**	**1645.79**	H-00	19	1650.80	R201	**4**	**1252.37**	HG-99	**4**	**1252.37**
R102	**17**	**1486.12**	RT-95	**17**	**1486.12**	R202	**3**	**1191.70**	RGP-f	**3**	**1191.70**
R103	**13**	**1292.68**	LLH-01	**13**	**1292.68**	R203	3	939.54	M-02	**3**	**939.50**
R104	**9**	**1007.24**	M-02	9	1007.31	R204	**2**	**825.52**	BVH-01	**2**	**825.52**
R105	**14**	**1377.11**	RT-95	**14**	**1377.11**	R205	**3**	**994.42**	RGP-f	**3**	**994.42**
R106	**12**	**1251.98**	M-02	12	1252.03	R206	**3**	**906.14**	SSS-00	**3**	**906.14**
R107	**10**	**1104.66**	S-97	**10**	**1104.66**	R207	2	893.33	BVH-01	**2**	**890.61**
R108	**9**	**960.88**	BBB-01	9	963.62	R208	**2**	**726.75**	M-02	2	727.69
R109	**11**	**1194.73**	HG-99	**11**	**1194.73**	R209	**3**	**909.16**	H-00	**3**	**909.16**
R110	**10**	**1118.59**	M-02	10	1118.84	R210	**3**	**939.34**	M-02	3	939.37
R111	**10**	**1096.72**	RGP-f	**10**	**1096.72**	R211	2	892.71	BVH-01	**2**	**890.93**
R112	**9**	**982.14**	GTA-99	9	987.24						

Table 3. The test results for the RC10x and RC20x data sets.

Probl. inst.	Best published solution			Best comp. solution		Probl. inst.	Best published solution			Best comp. solution	
	N	D	Ref.	N	D		N	D	Ref.	N	D
RC101	**14**	**1696.94**	TBG-97	14	1696.95	RC201	**4**	**1406.91**	M-02	4	1406.94
RC102	**12**	**1554.75**	TBG-97	**12**	**1554.75**	RC202	**3**	**1367.09**	CC-02	**3**	**1367.09**
RC103	**11**	**1261.67**	S-98	**11**	**1261.67**	RC203	**3**	**1049.62**	CC-02	**3**	**1049.62**
RC104	**10**	**1135.48**	CLM-00	**10**	**1135.48**	RC204	**3**	**798.41**	M-02	3	798.46
RC105	**13**	**1629.44**	BBB-01	**13**	**1629.44**	RC205	**4**	**1297.19**	M-02	4	1297.65
RC106	**11**	**1424.73**	BBB-01	**11**	**1424.73**	RC206	**3**	**1146.32**	H-00	**3**	**1146.32**
RC107	**11**	**1230.48**	S-97	**11**	**1230.48**	RC207	**3**	**1061.14**	BVH-01	**3**	**1061.14**
RC108	**10**	**1139.82**	TBG-97	**10**	**1139.82**	RC208	**3**	**828.14**	IKM-01	3	828.71

current temperature but also on the cost of a solution computed by the left neighbor in a line of co-operating processes.

The above scheme of co-operation is a modification of the scheme given by Aarts and Laarhoven (1987), Graffigne (1992), and Azencott and Graffigne (1992). Their scheme uses in Equations (5) and (6) the value of $V_{u\omega}^{(j-1)}$ instead of $\bar{V}_{u\omega}^{(j-1)}$. That is, process j updates its chain to the better solution found by its left neighbor in step $u\omega - 1$, $\mathcal{P}_T(V_{u\omega-1}^{(j-1)})$, and its own realization of this step, $\mathcal{P}_T(V_{u\omega-1}^{(j)})$.

Now note that the temperature of annealing decreases according to the formula $T_{i+1} = \beta T_i$ for $i = 0, 1, 2, \ldots, i_{\max}$, where $i_{\max}$ is the number of the cooling stages. There are three possibilities in establishing the points in which the temperature drops and the processes interact. Namely, we may assume that the processes interact frequently during each of temperature plateaus ($\omega < L$),

that the processes interact after executing L annealing steps ($\omega = L$), or that the temperature drops several times before an interaction takes place ($\omega > L$). In this work the second approach is taken, i.e. the processes interact after executing L annealing steps (this number of steps is executed in each temperature plateau).

5 Experimental Results

The parallel simulated annealing algorithm described in section 4 was slightly modified and implemented using C language. The goal of the modification was to allow the execution of the parallel algorithm on a single-processor computer. The tests of the algorithm were carried out on the problem instances published by Solomon (1987). The Solomon test set consists of 56 problem instances. Each of these instances comprises 100 customers. The location of the depot and the customers are given as integer values from the range 0 .. 100 in a Cartesian coordinate system. It is assumed that the travel times t_{ij} are equal to the corresponding Euclidean distances d_{ij} between the customer locations. The test problems are grouped into six problem types. In problem sets R1 and R2 the customer locations are generated randomly in a given area according to a uniform distribution. The geographical distribution of customers in sets C1 and C2 is clustered, whereas in sets RC1 and RC2 it is semi-clustered with a mix of randomly distributed and clustered customers. Sets R1, C1 and RC1 have narrow time windows and allow fewer customers per route. Problem sets R2, C2 and RC2 have wider windows and allow a larger number of customers per route. As already mentioned the objective of our work is to find as good solutions as possible to Solomon (1987) benchmark problems by using parallelism. For each problem instance 10 executions of the serial version of the parallel algorithm were performed. A single execution of the algorithm takes about 2–3 hours on a 850 MHz Duron processor, therefore all 560 tests would have taken about 1.5 month. Due to the help of our colleagues (see Acknowledgements) who shared with us their computational resources all the tests were completed within a week (Gocyła 2002).

Tables 1, 2 and 3 contain the test results. One can see that using parallel simulated annealing we were able to find the new best solutions to 3 problem instances (R203, R207 and R211). For 38 instances we found the best solutions known so far, and for 15 instances we foud the inferior solutions as compared to the best known. Although the results are quite good we believe that the parallel simulated annealing algorithm has further potentials which can be exploited (see remarks in section 6). During the tests the following values of parameters were used: $a = 200$ (number of cooling stages), $L = 100000$ (number of annealing steps performed in each temperature plateau), $p = 5$ (number of processes), $c_1 = 10000$, $c_2 = 1$ (constants of the cost function), $\beta = 0.98$ (constant of the geometric reduction of the temperature of annealing), $\gamma = 4 \cdot 10^{-4}$ (constant determining the initial temperature of annealing).

6 Conclusions

The parallel simulated annealing algorithm, based on the scheme of co-operation of processes as described in section 4, proved effective to solve the delivery problem (Czech 2001) and the vehicle routing problem with time windows (Czech and Czarnas 2002). In this work the advanced procedure to determine a neighbor solution was used. While building such a solution the procedure introduces substantial changes into a current solution, what facilitates a deeper exploration of the solution space. The experimental results described in section 5 are quite good and indicate that parallel simulated annealing can solve effectively bicriterion optimization problems. We believe that there are still some potentials in the presented algorithm. Among the issues which deserve investigations are the frequency of cooperation of processes (cases $\omega < L$ and $\omega > L$, cf. section 4) and further improvements of the procedure of the neighbor solution generation.

Acknowledgements. We thank the following colleagues for sharing with us their computational resources and devoting their time to execute our algorithm: Artur Całus, Paweł Chałczyński, Adam Cierpka, Wojciech Jastrzębski, Jacek Klukowski, Krzysztof Księżyk, Artur Kucharski, Mariusz Kwiatkowski, Paweł Łabuś, Piotr Nowak, Ryszard 'rychoo', Wojciech Sprycha.

References

1. Aarts, E.H.L., and van Laarhoven, P.J.M., Simulated annealing: Theory and applications, Wiley, New York, 1987.
2. Azencott, R., and Graffigne, C., Parallel annealing by periodically interacting multiple searches: Acceleration rates, In: Azencott, R. (ed.), Simulated annealing. Parallelization techniques, J. Wiley, NY, (1992), 81–90.
3. Bent, R., and Van Hentenryck, P., A two-stage hybrid local search for the vehicle routing problem with time windows, Technical Report CS-01-06, Depart. of Computer Science, Brown University, (2001), (BVH-01).
4. Berger, J., Barkaoui, M., and Bräysy, O., A parallel hybrid genetic algorithm for the vehicle routing problem with time windows, Working paper, Defense Research Establishment Valcartier, Canada, (2001), (BBB-01).
5. Cordeau, J.-F., Laporte, G., and Mericier, A., A unified tabu search heuristic for vehicle routing problems with time windows, Technical Report CRT-00-03, Centre for Research on Transportation, Montreal, Canada, (2000), (CLM-00).
6. Czarnas, P., A simulated annealing algorithm (in Polish), Master thesis, Wrocław, (2001).
7. Czech, Z.J., Parallel simulated annealing for the delivery problem, Proc. of the 9th Euromicro Workshop on Parallel and Distributed Processing, Mantova, Italy, (February 7–9, 2001), 219–226.
8. Czech, Z.J., Czarnas, P., A parallel simulated annealing for the vehicle routing problem with time windows, Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands, Spain, (January, 2002), 376–383, (CC-02).

9. Gambardella, L.M., Taillard, E., and Agazzi, G., MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, In: New Ideas in Optimization, D. Corne, M. Dorigo and F. Glover (eds.), McGraw-Hill, London, (1999), 63–76, (GTA-99).
10. Gocyła, P., A parallel simulated annealing algorithm for solving the vehicle routing problem with time windows (in Polish), Master thesis, Sosnowiec, (2002).
11. Graffigne, C., Parallel annealing by periodically interacting multiple searches: An experimental study, in Azencott, R. (ed.), Simulated annealing. Parallelization techniques, J. Wiley, NY, (1992), 47–79.
12. Homberger, J., Verteilt-parallele Metaheuristiken zur Tourenplanung, Gaber, Wiesbaden, (2000), (H-00).
13. Homberger, J., and Gehring, H., Two evolutionary metaheuristics for the vehicle routing problem with time windows, INFOR 37, 3 (Aug. 1999), 297–318 (HG-99).
14. Ibaraki, T., Kubo, M., Masuda, T., Uno, T., and Yagiura, M., Effective local search algorithms for the vehicle routing problem with general time windows, Working paper, Department of Applied Mathematics and Physics, Kyoto University, Japan, (2001), (IKM-01).
15. Li, H., Lim, A., Huang, J., Local search with annealing-like restarts to solve the VRPTW, Working paper, Department of Computer Science, National University of Singapore, (2001), (LLH-01).
16. Mester, D., An evolutionary strategies algorithm for large scale vehicle routing problem with capacitate and time windows restrictions, Working Paper, Institute of Evolution, University of Haifa, Israel, (2002), (M-02).
17. Rochat, Y., and Taillard, E.D., Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1, (1995), 147–167, (RT-95).
18. Rousseau, L.M., Gendreau, M., and Pesant, G., Using constraint-based operators to solve the vehicle routing problem with time windows, Journal of Heuristics, forthcoming (RGP-f).
19. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G., Record breaking optimization results using the ruin and recreate principle, Journal of Computational Physics 159, (2000), 139–171, (SSS-00).
20. Shaw, P., A new local search algorithm providing high quality solutions to vehicle routing problems, Working paper, University of Strathclyde, Glasgow, Scotland, (1997), (S-97).
21. Shaw, P., Using constraint programming and local search methods to solve vehicle routing problems, In: Principles and Practice of Constraint Programming – CP98, Lecture Notes in Computer Science, M. Maher and J.-F. Puget (eds.), Springer-Verlag, New York, (1998), 417–431, (S-98).
22. Solomon, M.M., Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research 35, (1987), 254–265, see also `http://w.cba.neu.edu/~msolomon/problems.htm`.
23. Taillard, E., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y., A tabu search heuristic for the vehicle routing problem with soft time windows, Transportation Science 31, 2, (May 1997), 170–186, (TBG-97).

Data Decomposition for Parallel K-means Clustering

Attila Gursoy

Computer Engineering Department,
Koç University, Sarıyer, İstanbul, 34450 Turkey
agursoy@ku.edu.tr

Abstract. Developing fast algorithms for clustering has been an important area of research in data mining and other fields. K-means is one of the widely used clustering algorithms. In this work, we have developed and evaluated parallelization of k-means method for low-dimensional data on message passing computers. Three different data decomposition schemes and their impact on the pruning of distance calculations in tree-based k-means algorithm have been studied. Random pattern decomposition has good load balancing but fails to prune distance calculations effectively. Compact spatial decomposition of patterns based on space filling curves outperforms random pattern decomposition even though it has load imbalance problem. In both cases, parallel tree-based k-means clustering runs significantly faster than the traditional parallel k-means.

1 Introduction

Clustering is an important area which finds application in a variety of fields including data mining, pattern recognition, explorative data analysis, image processing, chemistry, and more [1,2,3]. Given a set of patterns, the goal of clustering is to partition the input patterns into groups, called clusters, such that similarity between patterns of a particular cluster is maximized, whereas similarity between patterns of different clusters is minimized. Developing fast algorithms for clustering has been an important area of research and various algorithms have been proposed [1]. K-means [4] is a partitional clustering method and it is one of the most commonly used clustering algorithms. In this paper, we focus on parallelization techniques for tree-based k-means method [5], a faster version of k-means clustering.

The k-means algorithm treats input patterns as points in a d dimensional space and employs Euclidean-distance based similarity metric between patterns and cluster centers. The algorithm chooses an initial set of cluster centers and then each pattern is assigned to the cluster represented by the closest cluster center. After all patterns processed and new clusters are formed, cluster centers are updated to represent new clusters. This process is repeated until no patterns need to be moved. The time complexity of k-means increases with the product of size of input space and number of clusters. This time complexity prohibits use

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 241–248, 2004.
© Springer-Verlag Berlin Heidelberg 2004

of k-means method on large data sets. The explosion of data almost in many application areas in recent years necessitated development of faster algorithms. The tree-based k-means clustering [5] is one of the such approaches to improve the efficiency of the (sequential) k-means algorithm. It is based on spatial decomposition of pattern space in such a way that finding the closest cluster center for a pattern or for a set of patterns by considering only some spatially close cluster centers rather than all the centers. The technique significantly reduces the number of distance calculations as number of clusters increases and results in a much faster k-means method. Although, the study is limited to low dimensional data, there exist many cases where either data set is low dimensional or reduced to low dimensions. Therefore, we believe fast clustering algorithms for large size of low-dimensional data is still important.

Even though, tree-based pruning in the above approach improve execution time, the sequential algorithm is still far from satisfactory performance for large data sets. In addition, large data sets can be physically distributed across multiple computers. Therefore, development of parallel algorithms for the tree-based k-means is important for such cases. Parallelization of the direct (or traditional) k-means method is relatively easier. However, in the case of tree-based k-means, the traverse of the irregular tree structure complicates parallelization and poses load balancing problems. In this paper, we discuss parallelization of the tree-based k-means method and propose three different schemes based on pattern decomposition and parallel pruning search of a k-d tree used in the tree-based algorithm. The main motivation behind this study is to develop and evaluate alternative data decomposition schemes.

The paper is organized as follows: The next section briefly discusses k-means and tree based k-means (sequential) method. Sect. 3 explains proposed parallel algorithms for the tree-based k-means. An analysis and preliminary performance study of the parallel algorithms are presented in Sect. 4 followed by the conclusion and future work.

2 K-means Clustering

We first briefly explain the direct k-means algorithm, then discuss the tree-based k-means. Let $P_1, P_2, ..., P_n$ be the set of n patterns, where P_i is the i^{th} pattern consisting of a tuple of describing features $(a_{i1}, a_{i2}, ..., a_{id})$. Each feature is mapped to a dimension, where d is the number of dimensions of the value space. The number of clusters, k, is supplied as input to the algorithm, and it is fixed during the execution. Let $C_1, C_2, ..., C_k$ be the clusters, and $c_1, c_2, ..., c_k$ are the associated centroids, such that c_j is the centroid of cluster C_j, for $1 \leq j \leq k$. Each pattern P_i is assigned to at most one cluster C_j. The direct k-means algorithm iteratively updates the cluster centroids as follows: Each pattern is assigned to the closest cluster. Then, cluster centroids c_i's are updated to be the centroid of all samples in C_i. The algorithm iterates until a predefined maximum number of iterations is reached or the convergence at a local minimum. The quality of the clustering is measured by the error function $\sum_{j=1}^{k} \sum_{P_i \epsilon C_j} ||P_i - C_j||^2$. The time

complexity of one iteration is $O(nkd)$ where nk term comes from calculating the similarity (or distance) between each pattern and cluster centroid to find the closest centroid. As the number of patterns and clusters increases, k-means becomes less practical. In order to improve efficiency of the direct k-means, one may try to reduce the number of distance calculations. One such approach is to organize patterns as a hierarchical spatial structure as done in [5].

2.1 Tree-Based K-means

In [5], an improvement to k-means clustering, tree-based k-means algorithm, is described which reduces the number of distance computations through a pruning approach. The algorithm organizes patterns in a k-d tree. The root of the tree represents all patterns and children nodes represent patterns in subspaces. In each iteration, the k-d tree is traversed in a depth-first manner starting at the root node. At the root level, all cluster centroids are candidates to be the closest centroid to any pattern in the space represented by the root node. As we traverse the tree, a pruning method (based on size of the subspace) is applied to eliminate some of the candidates for the subspace represented by each node visited. That is, the candidate set that comes from the parent node might contain some clusters centroids that cannot be closest to any pattern in the subspace (which is smaller than the subspace represented by the parent node). When the candidate set is reduced to one cluster centroid, all the patterns in the subspace is assigned to that cluster. Otherwise, a leaf node eventually is reached and pairwise (like in direct k-means) distance calculations are performed for all patterns in the leaf node and cluster centroids in the candidate set (possibly less than k clusters due to pruning in the internal nodes). Pruning requires calculation of minimum and maximum distance to any given subspace from a given centroid. The pruning is done as follows: Let min_i and max_i are the minimum and maximum distances to any point in the subspace for cluster centroid i from the current candidate set. Then, we can eliminate any centroid j where $min_j > \min_{\text{for all } i}(max_i)$. The cost of pruning at a node is independent of the number of patterns in the subspace and can be done efficiently. The results show that tree-based k-means is significantly faster than direct k-means. The reader is referred to [5] for further details. In our work, region-based quadtree is used as the substitute of k-d tree because our datasets are two dimensional. The term quadtree has taken a generic meaning for d-dimensional trees having same properties. For details of quadtree and alternative data structures reader should refer to [6].

3 Parallel K-means

In the following section, we will discuss parallelization of direct k-means briefly in order to lead to parallelization of the tree-based one. We present three data decomposition schemes for the tree based k-means: random, striped, and tree decomposition.

3.1 Parallel Direct K-means

In direct k-means, the basic computation is the distance calculation between patterns and cluster centroids. For each pattern, k distance calculations are performed, therefore, the amount of computation done for each pattern does not vary from pattern to pattern. In addition, the computations for each pattern are completely independent. A simple and effective parallelization scheme then would be assigning each processor to a set of patterns with equal number of patterns. With this decomposition, a processor performs the distance calculations and maintains the partial update to cluster centroids due to its patterns. When all the processors are done, the local results are merged to find out new cluster centroids. Since each processor has equal amount of work and there are no interactions (except the merging phase), the parallel scheme will scale almost linearly with the number of processors. A similar approach for distributed memory machines using MPI message passing library is described in [7], and for shared memory machines in [8].

3.2 Parallel Pruning for Tree-Based K-means

One approach is to partition patterns among processors such that each processor performs pruning in the space covered by its own patterns. First, the master node reads the patterns and divides it into p partitions. Then, each partition is sent to a processor together with initial cluster centroids. All processors apply k-means on their local patterns with pruning algorithm. After new pattern assignments are found, the cluster centroids are updated with MPI reduce operation. This is the only synchronization point among processors and amount of data is very small compared to number of patterns. Therefore, communication is not expected to be a bottleneck during iterations of the algorithm. Although the parallelization of the tree-based k-means is similar to the direct k-means, getting good speedups is more challenging due to the irregular tree decomposition of space (since it depends on pattern distribution), and varying computations during the traversal. The computations done during the traversal can be coarsely divided into two groups: internal-node computations and leaf computations. In the internal nodes, the space covered by a node is compared against the current candidate set of centroids. Since some of the cluster centroids might have been pruned in the upper levels, the number of distance calculations (which is proportional to the size of the candidate set) can vary across internal nodes. At the leaf nodes, similarly, distance calculations among a differing number of remaining patterns and number of candidates results in varying computation loads.

Pattern Decomposition Schemes. The mapping of patterns to processors will have significant impact on the execution time in two ways: load imbalance and possibly less pruning compared to the sequential case. The load balancing is due to varying computational loads in the tree traverse phase. Assigning equal number of patterns to each processor might not guarantee a balanced work. For

example, consider two processors, one with a set of patterns that are concentrated in a small space, and another one with the same number of patterns but scattered around much larger space. There will be more pruning of candidate cluster centroids at the upper levels of the local quadtree tree in the case of compact subspace (as shown in Fig. 1) because many cluster centroids possibly will be far from the compact space. In the case of sparse and larger subspace, the pruning might shift towards to the leaves which might result in more distance calculations. In the direct k-means, it does not matter which patterns are assigned to processors. However, in the tree-based case, we have to choose patterns such that the problems mentioned above are addressed. We have tried two different partitioning schemes:

Random Pattern Decomposition. Each processor gets equal number of patterns chosen randomly from the space covered by all patterns. With this scheme, we expect a balanced computational load on the average.

Spatial Decomposition. Each processor gets a number of patterns that belong to a compact space (that is, spatially nearby patterns). This scheme might achieve better pruning. We have tried two schemes: a) striped decomposition where the space is divided into strips in one dimension, b) tree decomposition where the space is divided by utilizing the quadtree representation.

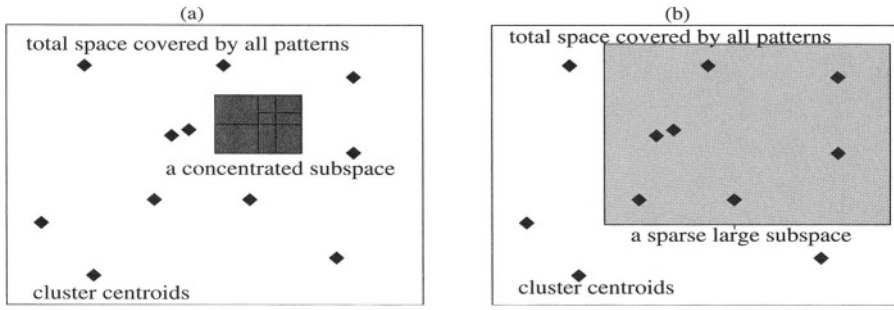


Fig. 1. Pattern decomposition: (a) concentrated small subspace will prune far cluster centroids quickly (b) sparse larger space (with the same number of patterns) will start pruning later towards to leaves

As mentioned above, as the subspace gets more compact, a processor can achieve more local pruning hence shorter execution time. The tree-decomposition is developed with the expectation of more compact subspaces compared to the ones produced by striped decomposition. The tree-decomposition is done as follows: The leaves of the quadtree is numbered according to a space filling curve. Then, the leaves of the quadtree is distributed to processors consecutively under this ordering. Figure 2 illustrates the spacefilling curve approach which is based on costzones approach used in the parallel n-body problem [9]. The computational load per processor can be estimated by the total number of patterns

assigned to processors as done in the random decomposition. However, a pattern in a lower level cell might cause more distance calculations since the cell covers bigger area. Therefore, in the tree decomposition, the computational load per cell is increased as the level is decreases for better load balancing.

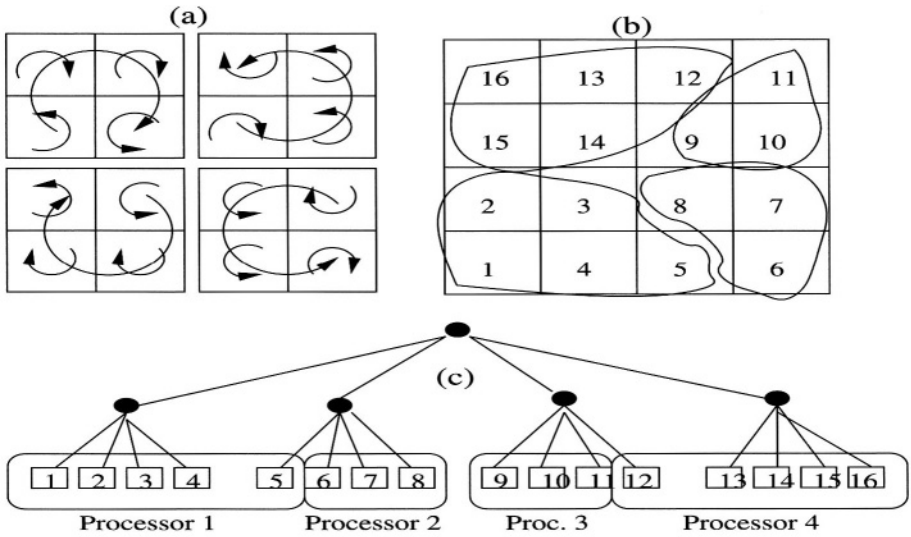


Fig. 2. Pattern decomposition: (a) nonuniform numbering scheme of cells (b) and (c) quadtree representation of space and a possible assignment of cells to processors

Table 1. Direct k-means versus pruning with random decomposition for data set D11 (parallel execution time in seconds)

Number of Processors	1	2	4	8	16	24
Direct K-means	144.38	103.05	53.59	27.78	14.83	10.54
Random Decomposition	11.05	9.25	5.71	4.01	2.91	2.48

Table 2. Parallel execution time in seconds for three decomposition schemes (D15)

Number of Processors	1	2	4	8	16	24
Random	43.10	35.75	21.02	13.25	8.70	6.93
Striped	42.78	28.11	17.30	10.31	6.70	5.13
Tree	42.75	25.65	12.80	7.42	4.65	3.54

Table 3. Total number of distance calculations ($x10^6$) on all processors (D15)

Number of Processors	1	2	4	8	16	24
Random	20.7	30.5	45.9	68.1	104.0	131.8
Striped	20.7	23.0	29.5	36.5	48.4	58.3
Tree	20.7	22.8	21.7	22.6	23.0	23.3

Table 4. Imbalance of distance calculations - percent difference from average (D15)

Number of Processors	2	4	8	16	24
Random	0	0	1	2	3
Striped	7	19	24	34	28
Tree	4	15	16	40	47

4 Evaluation of Decomposition Schemes

We have implemented the proposed algorithms with MPI message passing library in C++. Performance studies are conducted on 24 nodes PC cluster. The nodes are 400MHz Pentium II with 64 MB RAM and the interconnect is fast ethernet with gigabit uplink. The runs are done for 100000, 500000, and one million randomly generated two-dimensional pattern sets with 100, 200, and 400 clusters. The results shown here are for the 100000 patterns with 100 clusters (DS11) and 500000 patterns data set with 100 clusters (DS15) The other data sets produced similar results as well. Table 1 shows parallel execution time of direct k-means and randomly partitioned pruning based k-means. The pruning based k-means clustering is significantly faster than the direct k-means. However, the speedup is not as good as the direct one. Table 2 compares parallel execution time of random, striped, and tree-decomposition cases. Among these three, the random is the slowest, and the tree decomposition is the fastest one. Table 3 shows the total number of distance calculations performed by each processor. In the random decomposition scheme, the total amount of computation increases significantly as the number processors (that is, parallel algorithm performs much more computation than the sequential one). The reason, as anticipated, is that the random distribution covers a large area with fewer patterns and therefore it can't prune as much as the sequential one. As the number of processors increases, the amount of pruning decreases as well. Between striped and tree decomposition, clearly, tree decomposition achieves the best result, with higher parallel pruning among them. Table 4 shows the imbalance in the total number of distance calculations across processors. This number is closely related to the load of each processor (load imbalance). In random decomposition, the load is balanced quite well, only 3% load imbalance on 24 processors. In striped and tree decomposition cases, however, the load imbalance becomes severe as the number of processors increases. Although, the load imbalance is higher in tree decomposition case, it is the best as far as execution time is considered because of its compact form that allows it to prune more distance calculations than the others.

5 Conclusion and Future Work

We have developed a parallel tree-based k-means clustering algorithm and evaluated three data decomposition schemes on message passing computers. We have determined that spatial decomposition of patterns outperforms random pattern decomposition even though random decomposition has almost no load imbalance problem. The spatial decomposition, on the other hand, can be improved further by forming partitions in a more clever way and can be used also for running the algorithm on distributed memory machines. The current study is limited to low dimensional data sets, and, efficient parallelization schemes of k-means for higher dimensional sets need to be addressed as future work.

References

1. Jain, A.K., Murty, M.N., Flynn, P.J.: Data Clustering: A Review. ACM Computing Surveys, Vol. 31, No. 3 (1999) 264-323
2. Judd, D., McKinley, P.K., Jain, A.K.: Large-Scale Parallel Data Clustering. In Proc. of the 13th Int. Conf. on Pattern Recognition (1996)
3. Xue, L., Bajorath, J.: Molecular Descriptors for Effective Classification of Biologically Active Compounds Based on Principal Component Analysis Identified by a Genetic Algorithm. J. Chem. Inf. Comput. Sci. Vol. 40 (2000) 801–809
4. McQueen, J.: Some Methods for Classification and Analysis of Multivariate Observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability (1997) 173–188
5. Alsabti, K., Ranka, S., Singh, V.: An Efficient K-Means Clustering Algorithm. IPPS/SPDP 1st Workshop on High Performance Data Mining (1998)
6. Samet, H.: The Design and Analysis of Spatial Data Structures, Addison-Wesley (1989)
7. Dhillon, I.S., Modha, D.S.: A Data Clustering Algorithm on Distributed Memory Machines. In: Zaki, M.J., H0, C.T. (eds.): Workshop on Large-Scale Parallel KDD Systems. Technical Report 99-8. Computer Sci. Dept. Rensselaer Polytechnic Institute (1999)
8. Gursoy, A., Cengiz, I.: Parallel Pruning for K-Means Clustering Shared Memory Architectures. LNCS, Vol. 2150 (2001) 321–325
9. Singh, J.P., Holt, C., Totsuke, T., Gupta, A., Hennessy, J.: Load Balancing and Data Locality in Adaptive Hierarchical N-Body Methods: Barnes-Hut, Fast Multipole, and Radiosity. Journal of Parallel and Distributed Computing, 27 (1995) 118-141

On Generation of Permutations through Suffix/Prefix Reversing in a Cellular Network

Zbigniew Kokosiński

Cracow University of Technology, Faculty of Electrical & Computer Eng.,
ul.Warszawska 24, 31-155 Kraków, Poland; zk@pk.edu.pl

Abstract. In this paper a new permutation generator is proposed. Each subsequent permutation is generated in a cellular permutation network by reversing a suffix/prefix of the preceding permutation. The sequence of suffix/prefix sizes is computed by a complex parallel counter in O(1) time per generated object. Suffix/prefix reversing operations are performed at the same time when the permutation is actually produced.

1 Introduction

Generation of combinatorial objects belongs to a group of fundamental combinatorial problems that arise from many application areas. Many sequential and parallel generation algorithms were discovered [2,3,5,7,11] and analysed [1,10].

In this paper a new harware-oriented permutation generator is described in which the generation of permutations is equivalent to generation of interconnection patterns in a cellular array. We generalized the Zaks' permutation generation algorithm [11] by allowing both prefix and suffix reversals. It is shown that suffix/prefix reversals operations can be implemented in a triangular cellular network [4]. As a result the set of all n! permutations is generated in two different linear orders. If more then one generator is available the parallelization of the computations on the level of the set of objects also becomes possible and corresponding adaptive algorithm can be easily constructed [1].

2 A Representation of Permutations

Let $< A_i >_{i \in I}$ denote an indexed family of sets $A_i = A$, where: $A = I = \{1, \ldots, n\}$. Any mapping f which "chooses" one element from each set $A_1, \ldots, A_n$ is called a choice function of the family $< A_i >_{i \in I}$ [8]. With additional restrictions we can model by choice functions various classes of combinatorial objects. If for every $i \neq j$ a suplementary condition: $a_i \neq a_j$ is satisfied then any choice function $\alpha =< a_i >_{i \in I}$ that belongs to the indexed family $< A_i >_{i \in I}$ is called n-permutation of the set A. Set of all such choice functions represents the set of all permutations of the n–element set.

Prefix of the size p of the choice function $\alpha =< a_i >_{i \in I}$ is defined as the choice function $\pi =< a_i >_{i \in P}$, where $P=\{1, \ldots, p\}$ and $1 \leq p \leq n$.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 249–254, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Left cyclic shift of lenght t, 2≤t≤p, of the choice function $\pi =< a_i >_{i\in P}$ is the choice function $\tau^t =< a_i >_{i\in P}$ such that $\tau[i] = \pi[i+1]$, for 1≤i≤t-1, $\tau[t] = \pi[1]$, and $\tau[i] = \pi[i]$, for t+1≤i≤p.

Reversal of the choice function $\pi =< a_i >_{i\in P}$ is the choice function $\psi =< a_i >_{i\in P}$ such that $\psi[i] = \pi[p+1-i]$, for 1≤i≤p. We use also the notation $\psi = \pi^p_{rev}$. The reversal of the choice function $\pi =< a_i >_{i\in P}$ may be obtained by concatenation of left cyclic shifts of this choice function of lenght p,p—1, ... ,2 as stated in the following theorem (proof is omitted here):

Theorem 1. *If* $\alpha =< a_i >_{i\in I}$ *is a permutation and* $\pi =< a_i >_{i\in P}$ *is its prefix then* $\pi^p_{rev} = \tau^p \ldots \tau^t \ldots \tau^2$.

Suffix of the size s of the choice function $\alpha =< a_i >_{i\in I}$ is defined as the choice function $\sigma =< a_i >_{i\in S}$, where S={n-s+l, ... ,n} and 1≤s≤n.

Left cyclic shift of lenght t, 2≤t≤s, of the choice function $\sigma =< a_i >_{i\in S}$ is the choice function $\varphi =< a_i >_{i\in S}$ such that $\varphi[i] = \sigma[i+1]$, for n—t+1≤i≤n—1, $\varphi[n] = \sigma[n-t+1]$, and $\varphi[i] = \sigma[i]$, for n—s+1≤i≤n—t.

Reversal of the choice function $\sigma =< a_i >_{i\in S}$ is the choice function $\xi =< a_i >_{i\in S}$ such that $\xi[i] = \sigma[s+1-i]$, for n—s+1≤i≤n. We use also the notation $\xi = \sigma_{rev}$. Reversal of the choice function $\sigma =< a_i >_{i\in S}$ may be obtained by concatenation of left cyclic shifts of this choice function of lenght 2,3,...,s as stated in the following theorem (proof is omitted here):

Theorem 2. *If* $\alpha =< a_i >_{i\in I}$ *is a permutation and* $\sigma =< a_i >_{i\in P}$ *is its suffix then* $\sigma^p_{rev} = \varphi^2 \ldots \varphi^t \ldots \varphi^s$.

There is one to one correspondence between any linearly ordered set C of all choice functions α, with cardinality $|C| = c$, and linearly ordered set $\{0, 1, ..., c-1\}$. If $\alpha \in C$ then $\rho(\alpha)$ =x is called rank of α, where ρ is the ranking function. The function $\rho^{-1}(x) = \alpha$ is called unranking function.

3 Cellular Permutation Network

The triangular permutation array built from 2–state cells was invented by Kautz, Levitt and Waksman [4]. Each single cell has two inputs i_1, i_2, two outputs o_1, o_2, and performs identity permutation $(o_1 = i_1, o_2 = i_2)$ or transposition $(o_1 = i_2, o_2 = i_1)$. A group–theoretic formalism and an efficient O(n) algorithm for programming this cellular network in order to realize any n–permutation was proposed in [9]. Then the triangular network was applied for hardware generation of permutations in a new order [5]. By adding an extra cell state and proper control structure a versatile programmable hardware generator of n–permutations, (n,k)–combinations and all at most m–block partitions was constructed [6]. The triangular KLW networks enable easy suffix and prefix reversal operations (see Fig.1).

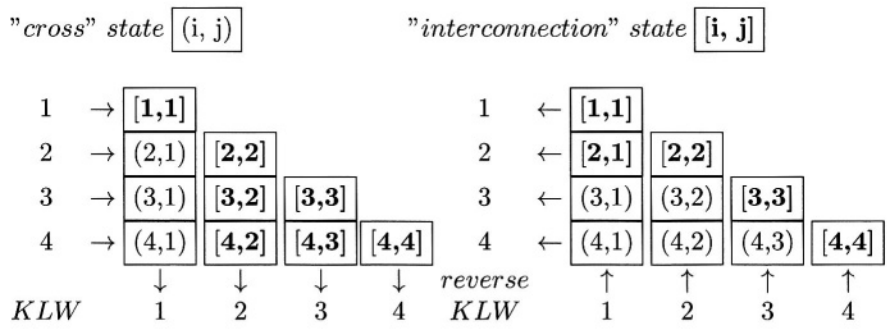


Fig. 1. Cellular permutation networks: a) KLW network performing suffix reversal (size=3); b) reverse KLW network performing prefix reversal (size=2);

In both networks any single column of cells in "interconnection" state performs a cyclic shift of lenght t, i.e. the choice function φ^t. Thus, according to Theorem 2, suffix reversal operation may be obtained in the network as a concatenations of column cyclic shifts (see Fig.1.a).

Property 1. *For any n–permutation α the suffix reversal of size s is performed in KLW and reverse KLW networks by setting all columns n–s+1,...,n in the "interconnection" state.*

In both networks any single row of cells in "interconnection" state performs a cyclic shift of lenght t, i.e. the choice function τ^t. Thus, according to Theorem 1, prefix reversal operation may be obtained in the network as a concatenations of row cyclic shifts (see Fig.1.b).

Property 2. *For any n–permutation α the prefix reversal of size p is performed in KLW and reverse KLW networks by setting all rows 1,...,p in the "interconnection" state.*

4 The Algorithm

Permutation generation method proposed by Zaks [11] produces the next n–permutation by reversing a certain suffix of the previous permutation. The initial permutation is <1,2, ... ,n>. The sequence of consecutive suffix sizes is described by the following recursive formula:

$$s_2 = 2;\ s_n = (s_{n-1}n)^{n-1}s_{n-1},\ \text{for n>2},$$

and is written as concatenation of its elements.

Property 3. *If the sequence of suffix sizes generated by the above formula is interpreted as the sequence of prefix sizes then the method of permutation generation by prefix reversal is complementary to Zaks' suffix reversal method.*

Input : n — size of the set.
Output: table P[n] with the consecutive permutations in a linear order.
Method: In steps 1–3 initial values of tables P, A and MAX are computed. In step 4 the first permutation is generated. If n ≠ 1 and not all permutations were generated, then in the while loop in step 5 subsequent values of the variable size are computed and the next n–permutation is produced.

```
1. for i:=1 to n do P[i]:=i;
2. for i:=1 to (n — 1) do
   2.1. A[i]:=0
   2.2. MAX[i]:=n — i;
3. i:=n — 1;
4. output P;
5. while i ≠ 0 do
   5.1. i:=n — 1;
   5.2. A[i]:=(A[i]+1) mod MAX[i];
   5.3. if A[i]=0 then
     5.3.1. repeat
              5.3.1.1. i:=i — 1;
              5.3.1.2. A[i]:=(A[i]+1) mod MAX[i];
            until A[i]≠0;
   5.4. size:=n — i+1;
   5.5. case of method :
            1: reverse suffix P[n — size+1] ... P[n] in table P;
            2: reverse prefix P[1] ... P[size] in table P;
   5.6. output P;
```

Fig. 2. Algorithm PERMGEN.

In [11] an iterative algorithm for computing the s_n sequence was developed. Our permutation generation algorithm computes suffix/prefix sizes by a counting process. According to the adopted method the consequtive permutations are obtained by suffix or prefix reversals. Sample sequences generated by the algorithm PERMGEN, for n=4, are depicted in Table 1. Additional sequence in column C (counter carries) reflects a method of hardware generation of suffix sizes as described in next section.

Property 4. *For any given choice function α representing a permutation, ranks $\rho(\alpha)$ and $\rho'(\alpha_{rev})$ in linear orders defined by sequences of suffix and prefix reversals, respectively, satisfy relation: $\rho(\alpha)+\rho'(\alpha_{rev}) = c-1$, where $c = n!$ (compare sequences in the two last columns of the Table 1).*

Property 5. *The α–sequence obtained by prefix reversals from the initial choice function $< 1, 2, \ldots, n >$ is identical as the α–sequence obtained by suffix reversals from the initial choice function $< 1, 2, \ldots, n >_{rev}$, and* vice versa.

Table 1. Sequences generated by algorithm PERMGEN (n=4)

No.	table A (counter output)	table C (counter carries)	suffix/ prefix size	table P1 (suffix reversal)	table P2 (prefix reversal)
1	0 0 0	0 0 0	2	1 2 3 4	1 2 3 4
2	0 0 1	0 0 1	3	1 2 4 3	2 1 3 4
3	0 1 0	0 0 0	2	1 3 4 2	3 1 2 4
4	0 1 1	0 0 1	3	1 3 2 4	1 3 2 4
5	0 2 0	0 0 0	2	1 4 2 3	2 3 1 4
6	0 2 1	0 1 1	4	1 4 3 2	3 2 1 4
7	1 0 0	0 0 0	2	2 3 4 1	4 1 2 3
8	1 0 1	0 0 1	3	2 3 1 4	1 4 2 3
9	1 1 0	0 0 0	2	2 4 1 3	2 4 1 3
10	1 1 1	0 0 1	3	2 4 3 1	4 2 1 3
11	1 2 0	0 0 0	2	2 1 3 4	1 2 4 3
12	1 2 1	0 1 1	4	2 1 4 3	2 1 4 3
13	2 0 0	0 0 0	2	3 4 1 2	3 4 1 2
14	2 0 1	0 0 1	3	3 4 2 1	4 3 1 2
15	2 1 0	0 0 0	2	3 1 2 4	1 3 4 2
16	2 1 1	0 0 1	3	3 1 4 2	3 1 4 2
17	2 2 0	0 0 0	2	3 2 4 1	4 1 3 2
18	2 2 1	0 1 1	4	3 2 1 4	1 4 3 2
19	3 0 0	0 0 0	2	4 1 2 3	2 3 4 1
20	3 0 1	0 0 1	3	4 1 3 2	3 2 4 1
21	3 1 0	0 0 0	2	4 2 3 1	4 2 3 1
22	3 1 1	0 0 1	3	4 2 1 3	2 4 3 1
23	3 2 0	0 0 0	2	4 3 1 2	3 4 2 1
24	3 2 1	1 1 1		4 3 2 1	4 3 2 1

Theorem 3. *Algorithm PERMGEN generates, in the form of α–sequences stored in table P, all n–permutations in a linear order that correponds to the lexicographic order of A sequences in constant time per object. Thus, the algorithm is correct.*

5 Construction of the Permutation Generator

The permutation generator consists of a control block and a commutative block. The control block is a programmable complex parallel counter. There is one to one correspondence between generation of permutations and counting in the code computed in the table A by the algorithm PERMGEN. Counter function is designed to control the permutation array which is the main component of the the commutative block (the array itself is a combinatorial circuit). Suffix/prefix

operations defined on the n–element set do correspond to interconnection patterns between inputs and outputs in the array. In hardware implementation the suffix/prefix size is not computed directly as in the step 5.4 of the algorithm but indirectly by computing counter carries stored in table C:

$$C[i] = 1 \text{ iff } (A[i] = MAX[i]) \cap (C[i+1] \cap \ldots \cap C[n-1] = 1), \text{ for } 1 \leq i \leq n-1,$$
$$\text{otherwise } C[i] = 0.$$

Every C[i], $2 \leq i$, controls states of all cells in columns (i—1) of the permutation network while suffix reversal is performed. Similarly, each C[i], $2 \leq i$, controls states of all cells in rows (n+2—i) of the permutation network while prefix reversal is performed. Each next permutation is obtained from the preceding one by single pass of permutation elements through the network in a constant time dependent on the size of the network size, which is O(n).

6 Concluding Remarks

In this paper new permutation generator has been described. In opposition to the original Zaks' algorithm our method of permutation generation is hardware oriented, uses either suffix or prefix reversal operations, provides parallelization of computations on the single object level and produces permutations in the form of interconnection patterns in the permutation network. Due to these properties the proposed solution can be located among other hardware generators of combinatorial objects having similar characteristics and applications [5,6].

References

1. Akl, S.G.: Design and analysis of parallel algorithms. Prentice Hall (1989) 148-150
2. Akl S.G.: Adaptive and optimal parallel algorithms for enumerating permutations and combinations. The Computer Journal **30** (1987) 433–436
3. Akl S.G., Meijer H., Stojmenović I.: Optimal parallel algorithms for generating permutations. TR 90-270, Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, Canada (1990)
4. Kautz W.H., Levitt K.N., Waksman A.: Cellular interconnection arrays. IEEE Transactions on Computers **C-17** (1968) 443–451
5. Kokosiński Z.: On generation of permutations through decomposition of symmetric groups into cosets, BIT **30** (1990) 583–591
6. Kokosiński Z.: Mask and pattern generation for associative supercomputing. Proc. 12th Int. Conference "Applied Informatics" AI'94, Annecy, France (1994) 324–326
7. Lin C-J.: Parallel algorithm for generating permutations on linear array. Information Processing Letters **35** (1990) 167–170
8. Mirsky L.: Transversal theory. Academic Press (1971)
9. Oruç. A.Y., Oruç A.M.: Programming cellular permutation networks through decomposition of symmetric groups. IEEE Trans. Computers **C-36** (1987) 802–809
10. Sedgewick R.: Permutation generation methods. Comp. Survey **9** (1977) 137–164
11. Zaks S.: A new algorithm for generation of permutations, BIT **24** (1984) 196–204

A Parallel Dynamic Programming Algorithm for Unranking t–ary Trees

Zbigniew Kokosiński

Cracow University of Technology, Faculty of Electrical & Computer Eng.,
ul. Warszawska 24, 31-155 Kraków, Poland; zk@pk.edu.pl

Abstract. In this paper an O(n) parallel algorithm is presented for fast unranking t–ary trees with n internal nodes in Zaks' representation. A sequential O(nt) algorithm is derived on the basis of dynamic programming paradigm. In the parallel version of the algorithm processing is performed in a dedicated parallel architecture containing certain systolic and associative features. At first a coefficient table is created by systolic computations. Then, n subsequent elements of a tree codeword is computed in O(1) time through associative search operations.

1 Introduction

Many different representations and sequential generation algorithms were invented and used for binary and t–ary trees, f.i. bitstrings, x–sequences, y–sequences, w–sequences, z–sequences etc. [3,4,7,12,14,15,17,19,20]. Two parallel generation algorithms in a linear array model and in an associative model were proposed providing parallelization of computations on the object level [2,11]. Sequential ranking and unranking algorithms were developed for binary trees [7] and t–ary trees [4,15,17,20]. Ranking and unranking of combinatorial configurations is applied in adaptive and random generation algorithms, genetic algorithms etc. [1,18].

In the present paper we propose a new parallel algorithm for unranking t–ary trees developed on the basis of a dynamic programming technique. Dynamic programming was successfully employed in many application areas including unranking combinations [9], partitions [8] and some other combinatorial objects [5, 6]. Although unranking problems are inherently sequential, a portion of computations can be parallelized. However, many sequential unranking algorithms are not suitable for parallelization. Till now, parallel dynamic programming algorithms were proposed for unranking combinations [10].

2 Representations of t–ary Trees

Let $< A_i >_{i \in I}$ denote an *indexed family of sets* $A_i = A$, where: A ={1, ..., m}, I ={1, ... , n}, $1 \leq$ m,n. Any mapping f which "chooses" one element from each set $A_1, ..., A_n$ is called a *choice function* of the family $< A_i >_{i \in I}$ [13].

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 255–260, 2004.
© Springer-Verlag Berlin Heidelberg 2004

With additional restrictions we can model by choice functions various classes of combinatorial objects [5,8].

If a suplementary condition: $a_i < a_j$, for i < j, and i, j $\in I$, is satisfied then any choice function $\kappa =< a_i >_{i\in I}$, that belongs to the indexed family $< A_i >_{i\in I}$, is called *increasing choice function* of this family (k–sequence). If n≤m, then all increasing choice functions $\kappa =< a_i >_{i\in I}$ are representations of all *n–subsets (combinations)* of m–element set *A*. In conventional representation of combinations with repetitions we deal in fact with indexed sets $K_i = \{i, \ldots, m-n+1\} \subset A_i$.

Below we define choice functions ζ and χ corresponding to the notions of z–sequences and x–sequences [12,19].

If suplementary conditions: 1. m=(n-1)t+1; 2. $a_i < a_j$, for i < j and i, j $\in I$; and 3. $a_i \in \{1, \ldots, (i-1)t+1\}$, for i $\in I$; are satisfied then any choice function $\zeta =< a_i >_{i\in I}$, that belongs to the indexed family $< A_i >_{i\in I}$, is called *increasing choice function with restricted growth of this* family (z–sequence [19]). In the above mappings we deal in fact with indexed sets $T_i = \{1, \ldots, (i-1)t+1\} \subset A_i$.

If $A_i = \{0,1\}$ and $I = \{1, \ldots, tn\}$ then any choice function $\chi =< a_i >_{i\in I}$, that belongs to the indexed family $< A_i >_{i\in I}$, is called *binary choice function* of this family (x–sequence [19]). All binary choice functions, with the number of $a_1 + \ldots + a_i \geq i/t$, for $1 \leq i \leq tn$, are bitstring representations of all t–ary trees of the set *A*. A simple transformation converts choice functions ζ into corresponding choice functions χ.

The number of all t–ary trees with n internal nodes is denoted by $B(n,t) = (tn)!/n!(tn-n+1)!$. The number of binary trees $B(n,2)$ is known as Catalan number $C(n)$.

For given n and t, the number of all choice functions ζ is a fraction $C_n^{nt}/(nt-n+1)C_n^{nt-t+1}$ of the set of all choice functions κ and a fraction C_t^{nt}/nC_t^{nt-n+1} of the set of all choice functions κ with $a_1 = 1$.

Let us introduce now the concept of Ruskey numbers [15]. The number of different (n,t)–trees(i.e. trees with n internal nodes) less then or equal to $\zeta =< z_1, \ldots, z_{n-i+1}, z_{n-i+2}, \ldots, z_n > \in < T_i >_{i\in I}$ with fixed $< z_1, \ldots, z_{n-i+1} >$ (in increasing lexicographic order) is called Ruskey number:

$$1.\ R_n^t(1,j) = j, \qquad \text{for } 1 \leq j \leq n(t-1)-1\ ;$$

$$2.\ R_n^t(i,(i-1)(t-1)+1) = B(n,t), \qquad \text{for } 1 \leq i \leq n\ ;$$

$$3.\ R_n^t(i,j) = R_n^t(i,j-1) + R_n^t(i-1,j),$$
$$\text{for } 1 \leq i \leq n, \text{ and } (i-1)(t-1)+1 \leq j \leq n(t-1)-1\ .$$

The above recursive formula describes construction of Ruskey tables for different values n,t. The table RT containing a part of Ruskey table, for n=5 and t=3, is shown in Table 1. Ruskey numbers $R_5^3(i,j)$ are stored in corresponding elements RT[i, j] of the table RT, while all remaining cells are filled with zeros.

Table 1. Construction of the table RT for (n,3)–trees, n ≤ 5, B(5,3)=273.

i / j	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	0	0	3	7	12	18	25	33	42
3	0	0	0	0	12	30	55	88	130
4	0	0	0	0	0	0	55	143	273
5	0	0	0	0	0	0	0	0	273

3 Sequential Dynamic Programming Algorithm

In this section we assume t–ary trees to be represented by increasing choice functions with restricted growth (z–sequences). In the algorithm UNRANKTREE presented in Fig.1 a table RT is used, which includes a part of Ruskey table.

Input : n — number of internal nodes, t — degree of tree nodes, Index — rank of the choice function ζ representing t–ary tree ($1 \leq$ Index $\leq$ B(n,t)), table RT with elements RT[i, j] containing Ruskey numbers $R_n^t(i,j)$.
Output: table T with choice function ζ.
Method: computations proceed with tree ranks in decreasing lexicographic order. In step 4 the maximum elements RT[i,m] satisfying inequality are selected in each row. The next value T[n—i+1] is computed in step 4.1.2. After O(nt) iterations we obtain the required t–ary tree ζ.

```
1. Index':=B(n,t)—Index + 1;
2. for I:=1 to n do T[i]:=n;
3. i:=n; j:=(n(t—1)—1; m:=j;
4. while (Index' > 0) do
   4.1. if RT[i,m] ≤ Index'
        then
        4.1.1. Index':=Index'—RT[i,m];
        4.1.2. T[n—i+1]:=T[n—i+1]+n(t—1)—1—m;
        4.1.3. m:=m+1;
        4.1.4. i:=i—1;
        else
        4.1.5. m:=m—1;
        4.1.6. if RT[i,m]=0 then i:=i—1;
5. return T.
```

Fig. 1. Algorithm UNRANKTREE.

Each coefficient $R_n^t(i,j)$ is mapped to the cell RT[i, j]. In the dynamic programming approach we assume that the table RT with Ruskey numbers is precomputed.

Theorem 1. *Algorithm UNRANKTREE is correct ind its asymptotic computational complexity is O(nt).*

Proof. The set of all B(n,t) trees can be shown in the form of a rooted ordered tree of height n (see Fig.2). There are n(t-1)-1 nodes with depth n. Each node with depth i, 0≤i≤n-1, has $it - k + 1$ descendants, where k is an integer label for edge connecting given node with its ancestor (for root that has no ancestor we assume $k = 0$), and edges connecting given node with its descendants are labeled by k+1, k+2, …, it + 1, respectively. In this way all nodes with depth i as well as all paths are ordered in the tree. Traversing the tree in preorder and listing all paths from the root to subsequent leaves — by sequences of edge labels — is equivalent to generation (enumeration) of all B(n,t) trees in increasing lexicographic order. Let us assign to all such paths their ranks in decreasing lexicographic order. Unranking the object with rank Index in the tree is equivalent to finding in the tree the path with rank Index'=B(n,t)—Index—1, 0≤Index, Index'≤B(n,t)—1. Every node of the tree with depth i has an integer label equal to the sum of all leaves of ordered subtrees rooted i in this node and all its siblings with depth i following it. Each node label is a Ruskey coefficient. We determine the path with rank Index' by determining a proper subtrees on the consequtive levels starting from the root. Rooted subtrees on ith level are viewed in the decreasing order of their size (size means in this case the number

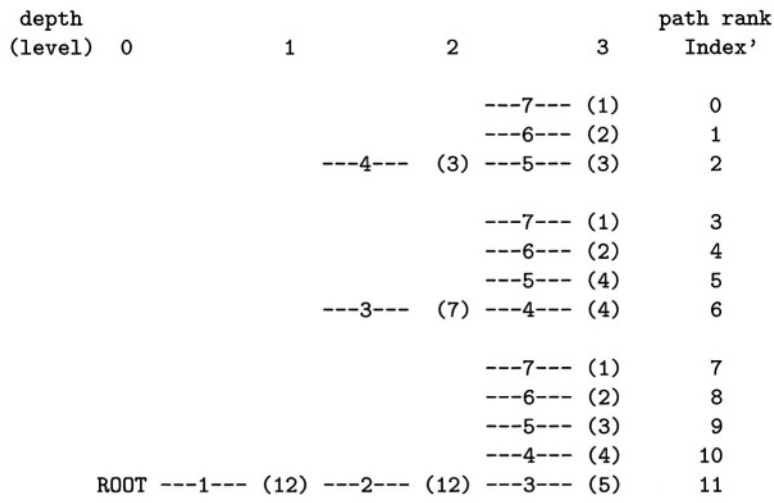


Fig. 2. Rooted ordered tree of all B(3,3) trees, where —x— denotes edge label and (x) denotes node label.

of subtree leaves). In order to do this the current Index' of the choice function is compared with node labels $R_n^t(i,j)$ and taken from the cell RT[i, j]. In each level i no more then (n—i+1)(t—1)—1 comparisons are made and before the next step rank Index' is modified (step 4.1.1 of the algorithm). Step 4 with complexity O(1) is repeated O(nt) times. Condition RT[i,j]≤Index' is satisfied n times in each step 4, and next item of the required object is obtained. Hence, the total complexity of the algorithm is O(nt).

4 Parallel Dynamic Programming Algorithm

In the algorithm UNRANKTREE two computational processes can be parallelized: 1) creation of the coefficient table RT, and 2) searching in the coefficient table RT.

Let us notice that elements in ith row of RT form a sequence which is increasing with column index j. This property is essential for speeding up the search in RT rows. For given pair {n,t} generation of RT requires O(nt) steps. Generation of the table RT from recursive formulas presented in section 2 may be parallelized through systolic computations.

A simple parallel unranking algorithm for t–ary trees implementing associative search operations *(no greater then* and *maximum value)* may be sketched as shown in Fig 3.

Input : identical as in algorithm UNRANKTREE.
Output: identical as in algorithm UNRANKTREE.
Method: computations proceed with tree ranks in decreasing lexicographic order. In order to determine table T an associative search is used. In each step 2.1 all elements RT[i,m] satisfying given inequality are selected. Then the element with maximum m coordinate is selected and the index value is updated. Next value T[n—i+1] is obtained in step 2.4. After n iterations we obtain the required function ζ in the table T.

1. Index':=B(n,t)—Index + 1;
2. **for** i:=1 **to** n **do** T[i]:=n;
3. j:=n(t—1)—1;
4. **for** i=n **downto** 1 **do**;
   4.1. **search in parallel** for all m ≤ j : RT[i,m] ≤ Index';
   4.2. **select maximum** m ;
   4.3. Index':=Index'—RT[i,m];
   4.4. T[n—i+1]:=T[n—i+1]+n(t—1)—1—m;
5. **return** T.

Fig. 3. Algorithm UNRANKTREE–PAR.

Theorem 2. *Algorithm UNRANKTREE–PAR is correct ind its asymptotic computational complexity is $O(n)$.*

Proof. The unranking algorithm is a variant of algorithm UNRANKTREE. Correctness of the method results from the Proof of Theorem 1.

References

1. Akl, S.G.: Parallel computation: models and methods. Prentice Hall (1997) 475–509
2. Akl, S.G., Stojmenović, I.: Generating t–ary trees in parallel. Nordic J. of Computing **3** (1996) 63–71
3. Er, M.C.: Lexicographic listing and ranking t–ary trees. The Computer Journal **30** (1987) 559–572
4. Er, M.C.: Efficient generation of k–ary trees in natural order. The Computer Journal **35** (1992) 306–308
5. Kapralski, A.: New methods for the generation of permutations, combinations and other combinatorial objects in parallel. Journal of Parallel and Distributed Computing **17** (1993) 315–326
6. Kapralski, A.: Modelling arbitrary sets of combinatorial objects and their sequential and parallel generation. Studia Informatica 21 (2000)
7. Knott, G.D.: A numbering system for binary trees. Comm. ACM **20** (1977) 113–115
8. Kokosiński Z.: Circuits generating combinatorial objects for sequential and parallel computer systems. Monografia 160, Politechnika Krakowska, Kraków (1993) [in Polish]
9. Kokosiński Z.: Algorithms for unranking combinations and their applications. Proc. Int. Conf. PDCS'95, Washington D.C., USA (1995) 216–224
10. Kokosiński Z.: Unranking combinations in parallel. Proc. Int. Conf. PDPTA'96, Sunnyvale, CA, USA, Vol.I (1996) 79–82
11. Kokosiński Z.: On parallel generation of t–ary trees in an associative model. PPAM 2001, Lecture Notes in Computer Science 2328 (2002) 228–235
12. Makinen, E.: A survey of binary tree codings. The Computer Journal **34** (1991) 438–443
13. Mirsky, L.: Transversal theory. Academic Press (1971)
14. Roelants van Baronaigien, D., Ruskey, F.: Generating t–ary trees in a–order. Information Processing Letters **27** (1988) 205–213
15. Ruskey, F.: Generating t–ary trees lexicographically, SIAM Journal of Computing **7** (1978) 424–439
16. Stojmenović, I.: On random and adaptive parallel generation of combinatorial objects. Int. Journal of Computer Mathematics **42** (1992) 125–135
17. Trojanowski, A.E.: Ranking and listing algorithms for k–ary trees. SIAM Journal of Computing **7** (1978) 492–509
18. Üçoluk G.: A method for chromosome handling of r–permutations of n–element set in genetic algorithms, Proc. ICEC'97, Indianapolis, USA (1997) 55–58.
19. Zaks, S.: Lexicographic generation of ordered trees. Theoretical Computer Science **10** (1980) 63–82
20. Zaks, S.: Generating and ranking t–ary trees. Information Processing Letters **14** (1982) 44–48

Adaptive Pareto Differential Evolution and Its Parallelization

Daniela Zaharie and Dana Petcu

Computer Science Department, West University of Timişoara, B-dul V.Pârvan 4, 300223 Timişoara, Romania,
{dzaharie,petcu}@info.uvt.ro

Abstract. An adaptive Pareto differential evolution algorithm for multi-objective optimization is proposed. Its effectiveness on approximating the Pareto front is compared with that of SPEA[9] and of SPDE[2]. A parallel implementation, based on an island model with a random connection topology, is also analyzed. The parallelization efficiency derives from the simple migration strategy. Numerical tests were performed on a cluster of workstations.

Keywords: Multi-objective evolutionary algorithms, Pareto front, parameter adaptation, island model, parallel computing, PC cluster.

1 Introduction

From a mathematical viewpoint a multi-objective problem (MOP) can be formulated as follows: find $x_* \in D \subset \mathbb{R}^n$ which optimizes a vector function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^r$, $\mathbf{f}(x) = (f_1(x), f_2(x), \ldots, f_r(x))$, i.e. it minimizes or maximizes each of the r objectives. In the following we shall consider that all the objectives have to be minimized. There are few situations when a solution, $x \in D$, can be found such that it optimizes all the objectives. Usually we are searching for "trade-off" solutions which assure a good compromise between the objectives. In the optimization theory such solutions are called Pareto optimal and are defined as follows: x_* is Pareto optimal if there does not exist $x \in D$ such that $f_i(x) \leq f_i(x_*)$ for all $i \in \{1, \ldots, r\}$ and $f_j(x) < f_j(x_*)$ for at least one $j \in \{1, \ldots, r\}$. The set of all Pareto optimal vectors is called *Pareto optimal set* and the set of the corresponding objective values is called *Pareto front.* For a set $S \subset D$ all the elements $x \in S$ having the property that there does not exist $y \in S$ such that $f_i(y) \leq f_i(x)$ for all $i \in \{1, \ldots, r\}$ and $f_j(y) < f_j(x)$ for at least one $j \in \{1, \ldots, r\}$ are called *non-dominated* with respect to S.

The evolutionary algorithms (EAs) are adequate in finding in a single run the entire Pareto optimal set because they work with populations of candidate solutions. Moreover, unlike traditional mathematical programming techniques, EAs are less susceptible to the convexity or continuity of the Pareto front. The basic characteristic of EAs is that the population of candidate solutions are iteratively transformed by some nature-inspired operators: *mutation, recombination* and *selection.* While for EAs developed for single-objective optimization

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 261–268, 2004.
© Springer-Verlag Berlin Heidelberg 2004

problems the critical operators seem to be mutation and recombination, in the multi-objective case the selection operator has the biggest influence. In the design of an EA for multi-objective optimization two main problems arises: (i) how to accomplish selection in order to guide the search towards the Pareto set; (ii) how to maintain a diverse population in order to prevent premature convergence and achieve a well distributed Pareto front. Some specific techniques are: Pareto ranking, archiving and elite preservation, fitness sharing. Most of Pareto-based multi-objective evolutionary algorithms (e.g. NSGA - Non-dominated Sorting Genetic Algorithm [3], SPEA - Strength Pareto Evolutionary Algorithm [9]) differ only on the manner that they implement the above mentioned techniques, the variation operators (mutation and recombination) being almost identical.

A somewhat different approach on designing an EA for multi-objective optimization is that of Abbas et al. [1] and Madavan [5] which extended to MOPs the differential evolution algorithm (DE), a powerful EA for single-objective problems on continuous domains. As the original DE, these algorithms are sensitive to the parameters values. To solve this problem Abbas designed in [2] a self-adaptive variant called SPDE. Following the same idea as in [5] we extend an adaptive differential evolution designed to preserve the population diversity during the evolution [11] to the multi-objective case obtaining an Adaptive Pareto Differential Evolution (APDE).

On the other hand, when the problem space is high-dimensional and/or many objectives are involved, solving a MOP using an EA requires extensive computational resources. A solution is the parallel implementation. Different parallel models for EAs have been proposed. They differ mainly on the level at which the parallelization is applied. Due to the particularities of Pareto-based selection the multi-objective EAs (MOEAs) have been parallelized mainly at the objective functions level.

The main aim of this work is to study the effectiveness of the proposed APDE on approximating the Pareto front and to analyse a parallelization based on the island model with a random connection topology.

The paper is organized as follows. Section 2 presents the Adaptive Pareto Differential Evolution algorithm and its numerical validation on some test functions. Section 3 analyzes the applicability of the island model to parallelize APDE. Numerical tests were performed on a PC cluster. Some concluding remarks are presented in Section 4.

2 Adaptive Pareto Differential Evolution

Adaptive Differential Evolution – ADE. In the classical DE algorithm [6], at each generation from the current population, $X = \{x_1, \ldots, x_m\}$ is obtained a new population $Z = \{z_1, \ldots, z_m\}$, by recombination rule as follows:

$$z_l^i = \begin{cases} x_{\alpha^l}^i + F \cdot (x_{\beta^l}^i - x_{\gamma^l}^i) \cdot N(0,1), & \text{with probability } p, \\ x_{\alpha^l}^i, & \text{with probability } 1-p, \end{cases} \quad l = \overline{1,m},\ i = \overline{1,n}, \tag{1}$$

with α^l, β^l and γ^l values selected without replacement from $\{1,\ldots,m\}$, independently for each i and l. $N(0,1)$ denotes a random value with standard normal distribution. In the selection step the *better* element between x_l and z_l is retained. The algorithm behavior is highly influenced by the parameters $F \in [0,2)$ and $p \in [0,1]$. In [11] is proposed a parameter adaptation based on the idea of controlling the population diversity. Preserving the population diversity is an important issue both in the case of single objective problems (in order to avoid premature convergence) and in the case of multi-objective problems (in order to obtain a diverse Pareto front). As measure of the population diversity are used the statistical variances computed for each component. The adaptation idea is based on a theoretical result [11] which establishes the influence of the recombination operator on these variances $E(\mathrm{Var}(z^i)) = \left(1 - \frac{2p}{m} + \frac{p^2}{m} + 2pF^2\right)\mathrm{Var}(x^i)$, with $E(\cdot)$ denoting the mean of a random variable. By adequately choosing p and F we can obtain a desired level for the population variance after the recombination step. Thus the original algorithm is modified as follows. The parameters F and p are replaced with two sets of parameters: $\{F_i\}_{i=1,\ldots,n}$ and $\{p_i\}_{i=1,\ldots,n}$ (a pair of parameters, (F_i, p_i), for each component). At each generation are computed the variances for all n components, $\mathrm{Var}(x^i(g))$, and the new values of the parameters F_i *or* p_i are determined such that the next recombination will compensate the effect of the last selection. To do this, the factor $c_i(g+1) = \gamma\mathrm{Var}(x^i(g))/\mathrm{Var}(x^i(g+1))$, $\gamma > 0$ is computed and $F_i(g+2)$ or $p_i(g+2)$ are determined by solving the equation

$$1 - \frac{2p}{m} + \frac{p^2}{m} + 2pF^2 = c_i(g+1), \tag{2}$$

with respect to F or p, respectively. At each generation only one set of parameters is modified. For instance, at even generations the values F_i are modified as follows:

$$F_i = \begin{cases} \sqrt{[m(c_i-1) + p_i(2-p_i)]/2mp_i} & \text{if } m(c_i-1) + p_i(2-p_i) \geq 0, \\ F_{inf} & \text{otherwise}, \end{cases} \tag{3}$$

with F_{inf} a minimal value for F (we used $F_{inf} = 1/\sqrt{m}$). Un upper value is also imposed: $F_{sup} = 2$. At odd generations are adapted the parameters p_i:

$$p_i = \begin{cases} -(mF_i^2 - 1) + \sqrt{(mF_i^2-1)^2 - m(1-c_i)}, & \text{if } c_i \geq 1, \\ p_{inf} & \text{otherwise}, \end{cases} \tag{4}$$

p_{inf} being the minimal value for p_i (e.g. $p_{inf} = 0.1$).

Adaptive Pareto Differential Evolution – APDE. The APDE is obtained by modifying only the selection step in ADE. The recombination is kept identically in order to apply the same parameters adaptation rule. However since we are interested in preserving both the diversity of the population and that of the Pareto front the right hand side member in Eq.(2) is

$$c_i(g+1) = \gamma\frac{\mathrm{Var}(x^i(g))\mathrm{Var}(\mathbf{f}(g))}{\mathrm{Var}(x^i(g+1))\mathrm{Var}(\mathbf{f}(g+1))} \quad \text{with } \mathrm{Var}(\mathbf{f}(g)) = \frac{1}{r}\sum_{j=1}^{r}\mathrm{Var}(f_j(g)).$$

The parameter γ allows us to control the algorithm behavior. Values of γ near 1 lead to parameters values for which next recombination compensates the effect on population diversity of the last selection. Choosing $\gamma < 1$ one could accelerate the convergence, but with the risk of inducing premature convergence. Usually values equal to 1 or slightly greater than 1 (e.g. $\gamma = 1.25$) are adequate.

The selection step is similar with that proposed in [3] and used in [5]: the joined offsprings and parents population is decreasingly sorted and the first m elements are selected. The first sorting criteria is the non-domination Pareto rank and the second one is the crowding distance proposed in [3]. This type of selection proved to be more elitist than that used in [1].

Numerical validation of APDE. We tested the APDE effectiveness on approximating the Pareto front on some test functions used in [10] to test SPEA (Table 1).

All test functions are characterized by $r = 2$, $\mathbf{f}(x) = (f_1(x_1), f_2(x_2, \ldots, x_n))$ with $f_2(x_2, \ldots, x_n) = g(x_2, \ldots, x_n)\ h(f_1(x_1), g(x_2, \ldots, x_n))$, $x_i \in [0, 1]$, $i = \overline{1, n}$ (for z_4, $x_i \in [-5, 5]$, $i = \overline{2, n}$). For z_1, z_2, z_3 and z_4 the first objective function is $f_1(x_1) = x_1$ while for z_6 it is $f_1(x_1) = 1 - \exp(-4x_1)\sin^6(6\pi x_1)$. The true Pareto fronts are $\{(x, 1 - h(x, 1)); x \in [0, 1]\}$.

Table 1. Test functions

Function	$g(x_2, \ldots, x_n)$	$h(u, v)$
z_1	$1 + 9\sum_{i=2}^{n} \frac{x_i}{n-1}$	$1 - \sqrt{u/v}$
z_2	$1 + 9\sum_{i=2}^{n} \frac{x_i}{n-1}$	$1 - (u/v)^2$
z_3	$1 + 9\sum_{i=2}^{n} \frac{x_i}{n-1}$	$1 - \sqrt{u/v} - u/v\sin(10\pi u)$
z_4	$1 + 10(n-1) + \sum_{i=2}^{n}(x_i^2 - 10\cos(4\pi x_i))$	$1 - \sqrt{u/v}$
z_6	$1 + 9\left(\sum_{i=2}^{n} \frac{x_i}{n-1}\right)^{0.25}$	$1 - (u/v)^2$

For z_1, z_2, z_3 the dimension is $n = 30$ while for z_4 and z_6 it is $n = 10$. Comparative results concerning the behavior of APDE, SPEA [9], SPDE[2] and PDE (a non-adaptive variant similar with that proposed in [5]) are presented in Table 2.

Table 2. Comparison between APDE ($\gamma = 1.25$), SPEA, SPDE and PDE ($p = F = 0.3$). Parameters: $m = 100$, 250 generations.

Fct.	APDE		SPEA		SPDE		PDE	
	GD	*CR*	*GD*	*CR*	*GD*	*CR*	*GD*	*CR*
z_1	0.0010	0.77	0.0362	0.79	0.0360	0.67	0.0009	0.73
z_2	0.0012	0.76	0.0533	0.52	0.0300	0.44	0.0009	0.72
z_3	0.0026	0.92	0.0103	0.57	0.0124	0.46	0.0013	0.71
z_4	0.0126	0.76	1.3329	0.34	–	–	0.1140	0.65
z_6	0.0725	0.64	0.3796	0.07	2.767	0.1	0.0713	0.65

As performance metrics we used the *generational distance - GD* (the average of the euclidean distances between the elements of the approximated Pareto front and a discretized version of the true Pareto front) and the *cover ratio - CR* [4]. The cover ratio is computed as follows. For each objective function the interval determined by the minimal and maximal function value corresponding to the true Pareto front is divided in m sub-intervals. The number of intervals which contain components belonging both to the approximated and to the true Pareto front is determined and it is divided by the number of intervals which contain values of the true Pareto front. *CR* is the average of these values computed for all objectives. A value of *CR* near 1 indicates a high diversity of the Pareto front.

The numerical results for SPDE and PDE have been obtained by simulations while for SPEA have been downloaded from http://www.tik.ee.ethz.ch/~zitzler/testdata.html. For SPEA and SPDE results from five independent runs have been gathered together while for APDE and PDE the metrics have been obtained by averaging results obtained in ten independent runs. The results suggest that APDE outperforms SPEA and SPDE (SPDE cannot be applied to z_4). In most cases PDE gives a better approximation of the Pareto front but with a lower diversity than APDE (for z_4 APDE behaves better than PDE). However PDE is sensitive to the parameter choice (for instance when $p = F = 0.5$ for z_4 it prematurely converges in 99% cases while APDE prematurely converges only in 48% cases).

3 Parallel Implementation of APDE

A short overview on parallel MOEA. The parallel implementations of EAs differ mainly on the level at which the parallelization is applied [7]: objective function evaluation level *(master-slave model),* population level *(multi-population model,* called also *island model* or *migration model),* elements level *(cellular model).* Due to the particularities of Pareto-based selection, the MOEAs have been parallelized mainly at the objective functions level.

In the multi-population model, the population is divided into sub-populations called *islands.* In each island a standard sequential evolutionary algorithm is executed. The communication between sub-populations is assured by a migration process: after some generations some elements leave their island and migrate to another. This process has an important role in preserving the population diversity. Its effectiveness depends on the communication topology (a graph structure in which the islands are nodes and the connections indicate the communications). Using this model, successful parallel implementations of EAs for single objective problems have been obtained [12]. At first sight for MOEAs an island based approach would decrease the cost of the Pareto ranking process (if the population is divided in s sub-populations the complexity order becomes $\mathcal{O}(rs(m/s)^2)$). However, this gain is true only if the migration process is a simple one.

Most of the existing island approaches for MOEAs (see for instance [4],[8]) are characterized by the presence of a global processing step when the elements

of all sub-populations are gathered together and sorted by some criteria. The parallelization which we propose tries to avoid such a global processing step.

The multi-population APDE. We consider now a multi-population approach for APDE which consists in dividing the population in s sub-populations of the same size, μ. On each sub-population an APDE is executed for a fixed number, τ, of generations. Each APDE corresponding to a sub-population works with its own sets of randomly initialized adaptive parameters. After each τ generations a migration process, based on a random connection topology, is started. More specifically: each element from each sub-population can be swaped (with a given *migration probability,* p_m) with a randomly selected element from a randomly selected sub-population. A global sorting step is *not* used. This allows obtaining a real gain by reducing the complexity order of the Pareto ranking process, which is applied only at sub-populations level. The question is if this strategy alters the ability of the algorithm to approximate the Pareto front. We analyzed the influence of the migration process on the ability of a multi-population APDE to approximate the Pareto front (see Table 3) and the main remark is that for adequately chosen values of τ (e.g. $\tau = 25$) and p_m ($p_m = 0.5$) the island model works appropriately but does not preserve the front diversity as well as the single population version.

Table 3. Influence of migration. Test function z_1, $n = 100$, $m = 180$, $\gamma = 1$, 500 generations. Averages and standard deviations for GD and CR are computed.

μ	s	p_m	τ	GD		CR	
				AVG	STDEV	AVG	STDEV
180	1	–	–	0.00130	0.00014	0.96	0.017
90	2	0	–	0.00305	0.00024	0.92	0.017
90	2	0.25	25	0.00127	0.00015	0.92	0.015
90	2	0.5	25	0.00116	0.00009	0.92	0.019
90	2	0.75	25	0.00130	0.00006	0.93	0.021
90	2	0.5	10	0.00296	0.00015	0.93	0.011
90	2	0.5	50	0.00313	0.00035	0.93	0.016
45	4	0.5	25	0.00145	0.00021	0.89	0.023
30	6	0.5	25	0.00249	0.00022	0.88	0.017

Numerical tests concerning the parallel APDE. In this section we present the results obtained running a multi-population implementation of the APDE on a PC cluster: 8 PC IV 1500 MHz with 256 Mb RAM interconnected via a Myrinet switch and optical fiber cables ensuring a transmission of 2 Gb/s. Such a system is suited for a random communication topology between the processes of a parallel code. The code is written in C and PVM (http://netlib.org/pvm).

In order to implement the multi-population model we have adopted the following strategy. The user can decide if the sub-populations will be treated in one or more processes. One processor of the cluster system can treat one or more

processes. A random communication topology is generated in the migration process. In the migration process an individual can be moved with probability p_m in a random position of a randomly selected subpopulation. The selected position being occupied by another individual, the later one will migrate to the former position of the incoming individual. If the destination subpopulation is treated by the same process it suffices a simple exchange; otherwise the individual will be gathered in a message buffer with the others which will migrate from the current process. This message buffer is sent to all other processes which will extract the individual data. The data corresponding to the individuals being replaced are sent back.

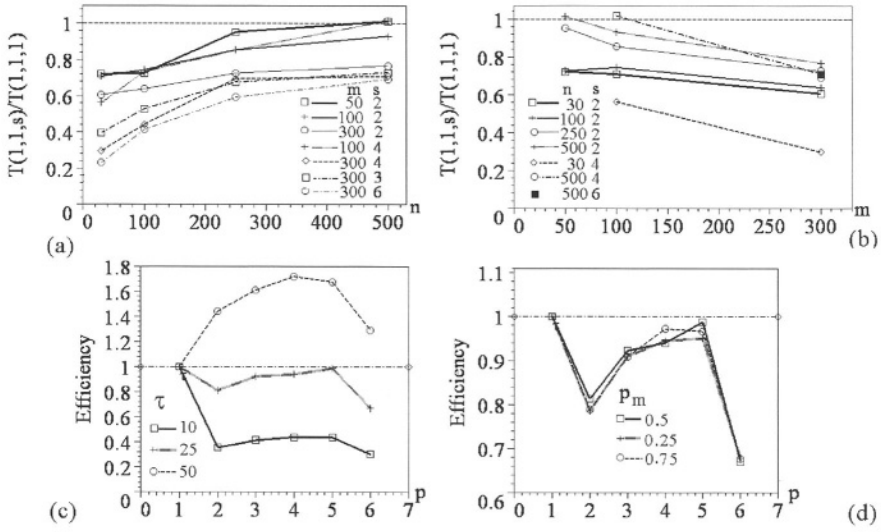


Fig. 1. Running times and efficiency for z_1: (a),(b) differences in time between single population and multi-population cases ((a) n is variable; (b) m is variable); (c) efficiency for $p_m = 0.5 - \tau$ variable; (d) efficiency for $\tau = 25 - p_m$ variable.

In order to determine the speedup of the code due to the parallel implementation we have measured the time spent by p processors treating t processes dealing with s subpopulations (denoted by $T(p,t,s)$). Figures 1 (a) and (b) show that the algorithm with several subpopulations is faster than the algorithm which does not use subpopulations. The difference is more relevant when n is small or m is large. This is due to the fact that the most costly operation (Pareto ranks computation) is done at the subpopulation level and a global sorting is not used. The migration step is $\mathcal{O}(m)$ thus it does not induce a high cost.

To analyze the efficiency of the parallel implementation we used the algorithmic speedup: $S_p = T(1,1,1)/T(p,p,p)$. The influence of the migration parameters τ and p_m on the efficiency ($E_p = S_p/p$) is illustrated in Fig. 1 (c) and (d).

The increase of efficiency for $p \in \{2, \ldots, 5\}$ can be explained by the fact that working with subpopulations reduces the computational cost. The results also suggest that $p_m = 0.5$ and $\tau = 25$ are adequate both from the viewpoint of the algorithm behavior (see Table 3) and of the parallel implementation.

4 Conclusions

The numerical tests suggest that the proposed APDE algorithm is competitive in solving MOPs on continuous domains when is compared with algorithms like SPEA and SPDE. On the other hand, using the island model with a simple migration strategy an efficient parallel implementation can be obtained.

References

1. H. A. Abbass, R. Sarker and C. Newton. PDE: A Pareto-frontier Differential Evolution Approach for Multi-objective Optimization Problems. In *IEEE Proc. of the Congress on Evolutionary Computation 2001* (CEC'2001), vol. **2**, 971–978, 2001.
2. H. A. Abbass. The Self-Adaptive Pareto Differential Evolution Algorithm. In *IEEE Proc. of Congress on Evolutionary Computation* (CEC'2002), vol. **1**, 831–836, 2002.
3. K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization. NSGA-II, KanGAL report 200001, Indian Institute of Technology, Kanpur, 2000.
4. Tomoyuki Hiroyasu, Mitsunori Miki and Sinya Watanabe. The New Model of Parallel Generic Algorithm in Multi-Objective Optimization Problems - Divide Range Multi-Objective Genetic Algorithm. In *IEEE Proc. of Congress on Evolutionary Computation* (CEC'2000), vol. **1**, 333–340, 2000.
5. N. K. Madavan. Multiobjective Optimization using a Pareto Differential Evolution Approach. In *IEEE Proc. of Congress on Evolutionary Computation* (CEC'2002), vol. **1**, 1145-1150, 2002.
6. R. Storn and K. Price. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Techn. Rep. TR-95-012, ICSI, 1995.
7. M. Tomassini. Parallel and Distributed Evolutionary Algorithms: A Review In *Evolutionary Algorithms in Engineering and Computer Science,* eds. K. Miettinen et al., J. Wiley and Sons, Chichester, 113–133, 1999.
8. F. Toro, J. Ortega, J. Fernandez, A. Diaz. PSFGA: A Parallel Genetic Algorithm for Multiobjective Optimization. In *Proc. 10th Euromicro Workshop on Parallel, Distributed & Network-based Processing* (EuroMicro-PDP 02), 2002.
9. E. Zitzler and L. Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Tech. Rep. 43, Computer Eng. and Comm. Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), 1998.
10. E. Zitzler, K. Deb and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* **8(2)**, 173–195, 2000.
11. D. Zaharie. Control of Population Diversity and Adaptation in Differential Evolution Algorithms. In *Proc. of Mendel 2003, 9th International Conference on Soft Computing,* eds. R. Matoušek, and P. Ošmera, 41–46, 2003.
12. D. Zaharie and D. Petcu. Parallel Implementation of Multi-population Differential Evolution. In *Proc. 2th Workshop on Concurrent Information Processing and Computing* (CIPC'03), Sinaia, eds. D. Grigoraş et al, in print, 2003.

Global Predicates for Online Control of Distributed Applications

Janusz Borkowski

Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warsaw, Poland
janb@pjwstk.edu.pl

Abstract. This paper proposes on-line synchronization and control of parallel applications based on global predicates. It reexamines existing theories, that have been developed mainly for application monitoring and debugging. Predicate modality *Instantly* is selected as the one that best meets the new requirements. Predicate parameterization is introduced to differentiate reactions of processes to predicate satisfaction. It is shown, that signals indicating predicate satisfaction sent to processes should be timestamped and should have relevant global state information attached to enable efficient application control. A correct simplified algorithm for construction of Strongly Consistent Global States, on which evaluation of predicates under modality *Instantly* is based, is presented.

1 Introduction

Application/system monitoring has been treated so far as a crucial method in testing and verification. In order to monitor an application/system we need to observe its consecutive states. In distributed environments (no global clock, no shared memory, only message passing) global states are not accessible by an observer in a direct way. Processes can report their local states to a monitor, but special techniques must be applied to order these states properly and reconstruct their (close to) original sequence [11,5]. Then global predicates can be defined on obtained global states and they can be used for monitoring and debugging parallel and distributed applications [6,12,9,13]. In this paper we propose using global predicates in a much different way. The predicates are an integral part of a parallel application and they control the application execution on the fly. When a predicate is met the monitor should influence the application control flow in an appropriate way. There are strong motivations supporting this idea. Global state theories are already well developed, so we have a solid ground to stand on. Many control and synchronization problems have an obvious solution when application global state is available, while they are tedious to implement and more error prone when using classic methods e.g. message passing. The code responsible for synchronization/control can be put concisely in one place: in the predicate definition. Global predicates, as mentioned above, are used to verify whether synchronization and control imposed by other means work as intended.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 269–277, 2004.
© Springer-Verlag Berlin Heidelberg 2004

We argue that a much more natural and simple solution would be to impose synchronization/control directly with the use of the predicates, making application behavior correct by construction and immediately verifiable. An integrated parallel programming environment incorporating these ideas is under construction in our laboratory [3,4]. There is only one other project employing some form of control based on global predicates, according to author's knowledge. The META [15] toolkit is designed for managing reactive distributed systems and it is a low level coordination-like language.

In the next section we describe the problem of observing global states in distributed systems. We present also different predicate modalities and evaluate their suitability for parallel application control. Section 3 shows, that application control based on standard global predicates is not flexible enough. Parameterized predicates and global state information distribution are introduced there. In Section 4 a simplified algorithm for Strongly Consistent Global State construction is presented. Section 5 contains a short summary.

2 Global States and Predicates

Consider a parallel/distributed system without a common clock and without shared memory. Message passing is the only communication mean. Messages travel along FIFO channels. The system consists of N processes $P_1, .., P_N$. A process state s has a start event $S(s)$ and a termination event $T(s)$. A global state is a tuple $<s_1, .., s_N>$, where s_i is a local state of P_i. A monitoring process can learn about states of application processes thanks to messages the processes send to it. The messages must have timestamps to let the monitor restore the original state order. Generally, it is impossible to restore the order precisely. It is possible to know event (state) casual relation [10,11,5], or approximated time of event occurrence [14,17] when process local clocks are partially synchronized. Using timestamps a monitor can construct a precedence relation on process states. $s_k \xrightarrow{r} s_l$ means, that a local state s_k on process P_k took place before a local state s_l on process P_l according to a chosen precedence relation r. Now we can define Consistent Global States (CGS) : $CGS^r = \{s : s = <s_1, .., s_N> \forall_{i,j=1..N}\, i \neq j \Rightarrow \neg s_i \xrightarrow{r} s_j\}$, which means, that a global state is consistent iff it contains pair wise concurrent local states [17]. There are other CGS definitions, which are equivalent [6,13,5,9]. Each global state, which actually took place, is a CGS, but it is unknown which of all possible CGSs actually took place. Because it is impossible to tell through which CGSs an application has actually passed, one cannot simply ask whether an application has passed through a global state satisfying a given predicate. Predicate modalities solve this problem. Five modalities have been defined in the literature so far. All of them were considered only in terms of passive application monitoring. Here we will reconsider them to tell how suitable they are for active on-line application control. Modalities $Possibly(\varphi)$ and $Definitely(\varphi)$ [6] - is it possible/sure, that the application has passed through a CGS in which φ is true? *Possibly* cannot tell wheather φ has been actually met, *Definitely* can miss a satisfaction. Both modalities cannot tell when a satisfaction happened.

To detect them it is necessary to construct the full CGS lattice and examine each CGS. The cost of it is $O(E^N)$, where E is the maximal number of events in a single process, N is the number of processes. This cost is prohibitive. Modality $Currently(\varphi)$ [6] - does the application state satisfy φ just now? Here each application process must wait for an acknowledgment from the monitor before changing its local state. The waiting is very intrusive and the waiting time is wasted. The former problem caused that *Currently* has not been considered widely for monitoring purposes; the latter makes this modality unacceptable for us. Modality $Properly(\varphi)$ [8] - was φ satisfied in such a CGS (so called *common* CGS), that the execution had to pass through it ? A special kind of CGS is defined here: *common* CGS. It is a global state, which must be passed through. If we define a predicate φ on common states, then a) we need to examine only common states, which can be done in a polynomial cost [8] b) if $Properly(\varphi)$ is satisfied, then φ is satisfied in reality c) the detection moment of $Properly(\varphi)$ corresponds to the moment φ was actually satisfied d) the state in which φ was satisfied is captured. However, if φ is satisfied in a non-common state then it can remain undetected. In [8] this serious problem is partially solved by using only conjunction of local predicates, which, by construction, can be satisfied only in common states. We are interested in using arbitrary predicates, that is why *Properly* is of very limited interest to us.

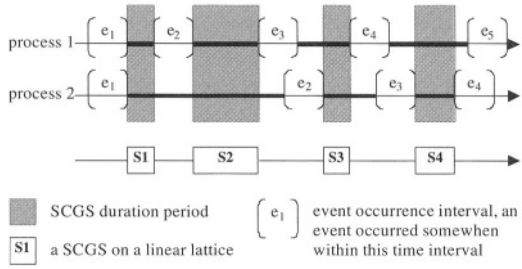


Fig. 1. Event occurrence history for two application processes with SCGSs depicted

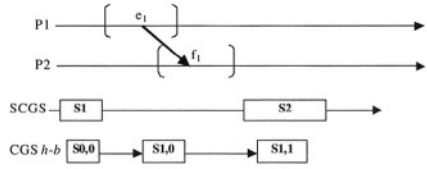


Fig. 2. Global states lattices obtained using different event orderings

Modality $Instantly(\varphi)$ [17] - checks if there was a period in real time, when the application was in such a state, that φ was satisfied. The previously discussed modalities are implemented usually with the use of vector clock timestamps [11,

5] as a way to reconstruct the causality relation between events and can be used in asynchronous systems. Now we need process local clocks to be partially synchronized [7,16] to use timestamps based on real time intervals (TRTI). TRTI are attached only to process-to-monitor messages and contain only two numbers. Processes may arbitrary select events about which they want to inform the monitor. These features lead to a low overhead of monitoring based on TRTI. CGSs based on TRTI are called Strongly CGS (SCGS) and they have interesting properties. For $s \in SCGS, s = <s_1, .., s_N>$, it is guaranteed that there was a time period when simultaneously $\forall_{i=1..N}$: process P_i was in state s_i. SCGS lattice is linear, so every SCGS is a common state. All the positive features of *Properly* apply here also. But, when applying *Instantly* there is no problem with φ being satisfied in non-common states - there are no such states. On Figure 1 we see four SCGSs. There are time spans, where no SCGS can be reported, because of uncertainty in events timing. If the clocks synchronization is done with accuracy ε, then we will be able to detect all SCGSs lasting longer than 2ε. So, the only problem here is a possibility to miss short-lived states, in which φ can be satisfied. Although it is a limitation, it can be diminished by more accurate clock synchronization and it is a well-defined limitation, so one can deal with it reasonably e.g. by increasing parallel task granularity. The detection algorithm cost is *O(EN* log *N)* [17]. It is far better than in the case of other modalities. It turns out, that modality *Instantly* is the best candidate for parallel/distributed application control purposes, as rough clock synchronization is widely applied nowadays. Unfortunately, TRTI do not always reflect the happened-before relation. See Figure 2. We cannot not tell which event was first: e_1 or f_1 (time intervals overlap). The ordering based on the happened-before relation leads to a lattice depicted as CGS h-b and clearly guarantees correctness. Please notice, that if message transfer time l is longer that 2ε, then send and receive events will get non-overlapping time intervals. The order imposed by happened-before relation will be observed! RBS protocol [7] can provide $\varepsilon \approx l/100$ on networks with hardware broadcast, e.g. on Ethernet. Also accurate GPS clocks can be employed. As a result, in properly designed environments, it is possible to use TRTI to produce observations consistent with happened-before relation.

3 Flexible Control over Application Processes

Our target is to have a flexible control over parallel application based on global predicates. We assume, that the monitoring process observes application global states and it informs application processes whenever a predicate is satisfied. The information takes form of a synchronization signal. Here we will not delve into the nature of the signals. It is enough to say that they are kind of messages. A detailed proposal concerning reactions signals can cause and how to implement them is given in [2,3].

The simplest type of association between predicate satisfaction and process reactions can be named “global predicates cause global reactions”. It means, that all processes react on predicate satisfaction in the same way. Barrier is a well-

known synchronization primitive, we will use it as an example: Predicate φ=have all the processes reached the barrier? *Reaction* (the same for every process) = cross the barrier. However, in many cases this simple association is not enough. Another example refers to load balancing. The monitor is informed about the load of application processes. φ= is there a load imbalance? *Reaction*=?? Not every process is to take part in the load balancing procedure, and those which are involved should know if to take or give their load, and how much. Here we are far from uniform global reactions. We propose to parameterize predicates with a process name to be able to differentiate process reactions. For each fixed value of i in $\varphi(i)$, where $i = 1..N$ is a process number, we get a predicate variant. The monitor evaluates each variant conceptually separately. If $\varphi(k)$ is satisfied, then it means that the predicate is satisfied in respect to process P_k and P_k should react on it. An example deals with parallel branch-and-bound algorithms[1,4]. A solution space can be divided into subregions; each subregion has an estimation of solutions contained in it. Processes examine subregions one after another. There is no point to examine a subregion, for which the estimation is worse than a solution found so far. $\varphi(i)$= is P_i examining a subregion with estimation worse the global best solution found so far? *Reaction*= abandon the examination of the current subregion and take the next one. Parameterization does not solve all the problems. Let us suppose, that a process P_i has L subregions to examine, and $K < L$ of them have their estimation worse than a currently known solution. P_i will start K times subregion examination and K times $\varphi(i)$ will evaluate to true causing P_i to abandon the current search. It would be much more effective if the process knows the best solution found so far, so it could simply omit uninteresting subregions by itself. It would not start unnecessary calculations, and the monitor would not have to trigger reactions.

Problems pointed out above stem from the fact, that application processes do not know the state of the whole system. This knowledge is required to choose proper reactions in more complicated cases. The monitor has that knowledge. The solution is to have the monitor pass necessary information to processes along with synchronization signals. In this way the signals have two components 1) identification of the predicate, whose satisfaction caused the signal, 2) a proper fragment of the global state, in which the predicate has been satisfied, necessary for choosing a proper action by the receiving process. An example: $\varphi 1(i)$= is P_i overloaded? *Attached information*= identifiers of underloaded processes and amount of work to transfer, *Reaction* = P_i transfers a proper part of its load to the appointed process(es). The next example brings us back again to the branch-and-bound parallel search. $\varphi(i)$= is P_i examining a subregion with estimation worse the global best solution found so far? *Attached information*= the current global best solution, *Reaction*= abandon the examination of the current subregion and take the next one. Now, a process having learned the value of the current best solution will be able to skip examinations of other uninteresting regions autonomously.

Message transmissions and SCGS detection take time. There must be a delay between entering a state satisfying a condition and an arrival of a signal caused

by that condition. For stable predicates this delay is not crucial. However, we want to deal with unstable predicates, too. The branch-and-bound example is useful here again. Imagine that process P_i is examining a subregion s_k, while a new best solution was found by P_m, making further investigation of s_k unnecessary. A signal is sent to P_i to abandon s_k, but it arrives when P_i, is exploring s_{k+1} already. Subregion s_{k+1} may contain a better solution, but its examination will be erroneously canceled. We have to deal with such situations. The monitor knows the real time occurrence period of each SCGS. A signal can have a timestamp marking the SCGS, which has led to the generation of the signal. A process can decide, checking the timestamp, whether to react on the signal, or the signal is too much delayed [2,3].

4 Simplified SCGS Detection Algorithm

Our starting point was an algorithm published in [17]. It explored an equivalence between SGCSs and common states. It was shown, that the common state detection algorithm [8] leads to SCGS detection, additionally performance improvements can be obtained. In this algorithm we start from an arbitrary CGS, and using a common state testing condition we advance processes by one event until we reach a common state (which is a SCGS). However, after a SCGS is found, we need to start over to find the next one. Here the descriptions and proofs are vague. The idea is to step by one event at a well-chosen process to leave the current SCGS and to enter one of the next CGSs. One has to find a proper process (additional cost) and this step has not been well explained. We will formulate the algorithm using different approach. On Fig. 1 one can see, that SCGS duration periods can be graphically interpreted as nonempty intersections of segments representing state duration at each process. Each process has a sequence of segments corresponding to its local state history. We are to find all non-empty intersection of segments, containing one segment from each sequence. The segment positions in each sequence are sorted. At first we assume, that intervals between consecutive events at one process are longer than 2ε, where ε is the clock synchronization accuracy. The notation is as follows:

SEQ_i	segment sequence for process P_i,
CS	the currently examined set of segments containing one segment from each sequence,
s_i	segment from SEQ_i in CS,
$S(s)$	segment s start position
$T(s)$	segment s terminate position
$next(s_i)$	the next segment after s_i in SEQ_i

The intersection is non-empty if $\forall_{i,j=1..n} : S(s_i) < T(s_j)$, which can be simplified as $\max_{i=1..N}(S(s_i)) < \min_{i=1..N}(T(s_i))$ (condition C1). Initially CS contains the first segment from each sequence. If the condition is not met, then there exist k and l, such that $S(s_k) \geq T(s_l)$. Assuming that we have checked all segments lying before the ones currently in CS (which is true initially) there

is no point to attempt decreasing $S(s_k)$, instead we can take the next segment in SEQ_l to increase $T(s_l)$. In such a way we proceed forward moving by one segment from one sequence at a time, preserving the assumption. If a nonempty intersection is found, we have to restart the search. There exists k, such that $T(s_k) = T(CS)$. It is enough to notice, that the next non-empty intersection cannot start earlier than $S(next(s_k))$. So we take the next segment from SEQ_k and run the procedure further, keeping the assumption. By induction, no non-empty intersection will be missed and all segments will be checked.

When we allow short intervals between events in a single process, we can have a situation as the one depicted on Fig. 3, left. There is a "negative length" segment between events e_1 and e_2. It should be interpreted that we do not know the exact start/end points of the segment, yet we know there is one here. For $P1$ in a state between events e1 and e2 and P2 between f1 and f2 condition C1 is false. However, we do have a nonempty intersection here. The right-hand side of Fig. 3 shows a similar situation, but this time the intersection is empty. The algorithm presented in [17] will declare an SCGS at this point incorrectly. We formulate the correct full condition as follows:

$$C1 \vee S(s_i) = \max_{k=1..N}(S(s_k)) \wedge T(s_i) = \min_{k=1..N}(T(s_k)) \wedge$$

$$\max2_{k=1..N}(S(s_k)) < T(s_i) \wedge \min2_{k=1..N}(T(s_k)) > S(s_i))$$

where min2() and max2() give the second minimal and maximal value, respectively. The condition says, that if we have a "negative length" segment, then it must be fully contained in other segments.

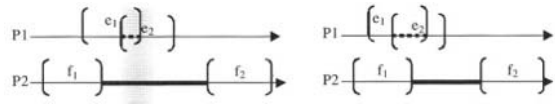


Fig. 3. Short-lasting local states with overlapping event intervals can lead to an SCGS (left) or not (right)

To speed up min/max (also min2/max2) operations priority queues can be employed. $T(SC) = \min_{i=1..N}(T(s_i))$, so the search for $k{:}T(s_k) = T(SC)$ can be accelerated by using one of the queues. Now, when we substitute all the symbols with symbols referring to states, events and timestamps as defined in section 2, we will obtain an algorithm very similar to the one presented in [17] and its cost is $O(EN\log N)$. In our approach, however, it is apparent what to do after a SCGS is found, short-lasting local states with overlapping event intervals are handled correctly, we need only two priority queues and the theory supporting the correctness of the algorithm is much simpler.

5 Conclusions

The paper desribes a method of using global predicates for parallel/distributed application control. Strong motivations are given to support the proposed methodology. We reexamined predicate modalities to determine how well they are suited for on-line application control. It turned out that modality *Instantly* satisfies our requirments. Although it is based on real-time timestamps, the happened-before relation can be still observed in systems with local clocks skews less than half of the minimal message transfer time, which is feasible. Uniform process reactions on predicate satisfaction have limited use, so we introduced parameterized predicates to differentiate process reactions. Part of a global state data should be made available to processes along with information about predicate satisfaction. This information itself can arrive with a delay, therefore it should be timestamped to let processes decide whether to accept or reject it. Finally, we describe a corrected and clarified SCGS detection algorithm, indispensable when using modality *Instantly*. The presented concepts are being tested practically - we are building a parallel programming environment supporting them.

Acknowledgements. The work has been sponsored by KBN grant 4T11C 007 22 and an internal PJIIT research grant.

References

1. I. P. Androulakis and C. A. Floudas, "Distributed branch and bound algorithms for global optimization", The IMA Volumes in Mathematics and its Applications, vol 106 Parallel Processing of Discreet Problems, Springer 1999, pp 1-37
2. J. Borkowski, "Interrupt and Cancellation as Synchronization Methods", in Proc. of 4th Int. Conf. Parallel Processing and Applied Mathematics PPAM 2001, Nalecz-ow, Poland, Springer 2001, LNCS 2328
3. J. Borkowski, D. Kopanski, M.Tudruj, "Implementing Control in Parallel Programs by Synchronization-Driven Activation and Cancellation", Proc. of the 11th Euromicro, Genova, Italy, IEEE 2003
4. J. Borkowski, D. Kopanski, M. Tudruj, "Adding Advanced Synchronization to Processes in GRADE", in Proceedings of the Int. Conf. on Parallel Processing in Electrical Engineering, PARELEC 2002, Warsaw, Poland, IEEE 2002
5. O. Babaoglu and K. Marzullo, "Consistent global states of distributed systems: fundamental concepts and mechanisms", Distributed Systems, Addison-Wesley, 1995
6. R. Cooper and K. Marzullo, "Consistent detection of global predicates," Proc. ACM/ONR Workshop on Parallel Distributed Debugging, pp 163-173, 1991.
7. J. Elson, L. Girod and D. Estrin. "Fine-grained network time synchronization using reference broadcasts." In Proc. of the 5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, Dec 2002
8. E. Fromentin and M. Raynal, "Characterizing and detecting the set of global states seen by all observers of a distributed computation," Proceedings of the 15th International Conference on Distributed Computing Systems, pp 431-438, 1995.

9. V. K. Garg, B. Waldecker, "Detection of Strong Unstable Predicates in Distributed Programs", IEEE Trans, on Parallel and Distrib. Systems, Vol. 7, No. 12, Dec 1996 (pp. 1323-1333)
10. L. Lamport, "Time, clocks and the ordering of events in a distributed system", Commun. ACM 21, 7, pp. 558-564, 1978
11. F. Mattern. "Virtual Time and Global States in Distributed Systems". Proc. Workshop on Parallel and Distributed, Algorithms, Chateau de Bonas, Oct. 1988, M. Cosnard et al. (eds.), Elsevier / North Holland, pp. 215-226, 1989.
12. M. Minas, "Detecting Quantified Global Predicates in Parallel Programs", Europar 95 , Stockholm, Sweden. Proceedings. LNCS, Vol. 966, Springer, pp 403-414
13. N. Mittal and V. K. Garg. "On Detecting Global Predicates in Distributed Computations". In Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS), pages 3-10, Phoenix, Arizona, April 2001.
14. K. Marzullo and G. Neiger, "Detection of Global State Predicates", Proc. of 5th Int. Workshop WDAG '91, Delphi, Greece, 1991, LNCS 579 Springer 1992
15. K. Marzullo and M. D. Wood. "Tools for Constructing Distributed Reactive Systems" , TR Cornell University, Dep. of Computer Science, Ithaca, New York, Feb. 1991, http://citeseer.nj.nec.com/145302.html
16. D. Mills, "Network Time Protocol (ver. 3) Specification, Implementation and Analysis", RFC1305
17. S. D. Stoller: "Detecting Global Predicates in Distributed Systems with Clocks", Distributed Computing, Volume 13 Issue 2 (2000) pp 85-98

A Thread Partitioning Algorithm for Data Locality Improvement

Alexander Chernov, Andrey Belevantsev, and Oleg Malikov

Faculty of Computational Mathematics and Cybernetics,
Moscow State University,
Leninskie Gory, Moscow, 119899, Russia
{cher, bonzo, malikov}@unicorn.cmc.msu.ru

Abstract. This paper addresses the problem of automatic thread partitioning of sequential programs. We propose a new algorithm, which has the main features as follows: 1) non-loop regions of programs may be parallelized; 2) partitioning is performed in the way that data locality in the new threads is improved. The algorithm is preliminarily evaluated using one handcrafted example and a generator of data dependence graphs, tuned to generate random graphs with characteristics close to those of real-world programs, and a cache simulator. The results of evaluation are presented which shows feasibility of our approach.

1 Introduction

Workstations and PCs, which contain more than one CPU, are widely spread. These computers typically contain 2, 4, or more CPUs operating on shared memory. One approach to exploit the full power of multi-processor computers is to use "light" processes (threads). As all the threads share one address space, no data transfer is necessary, and thread synchronization primitives are often simple and fast.

On the other hand, performance of programs may be significantly improved, if program locality properties are considered. Most programs naturally exhibit good locality properties, i.e. they execute close groups of instructions, or they access closely coupled data. Processor cache is the hardware feature, which benefits the most from the good locality properties of programs. Improvement of data locality properties of the program is essential for effective cache utilization.

In this work we present a new algorithm, which combines the two optimizations mentioned above. A sequential program is partitioned into threads so that each thread uses its cache better, than the original program. The algorithm extensively uses information that is gathered during program profiling. A preliminary evaluation of this algorithm on real multiprocessing systems and on a developed simulator, is also presented.

This paper has the following structure. In the section 2 the proposed algorithm for thread separation and locality improvement is described. Section 3 contains evaluation results of our algorithm. Section 4 provides discussion of related work. Then we summarize the results and outline the directions for further research.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 278–285, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Algorithm Description

Our goal is to develop an algorithm for program separation into threads that works for SMP boxes with several CPUs. The algorithm construct a parallel program that consists of several threads each running on its own CPU. First, we will make a sketch of our algorithm, and then we will give its formal description.

For the example we chose a function that calculates the roots of quadric equation $x^4 + ax^3 + bx^2 + cx + d = 0$. The function does not contain loops and cannot be parallelized by most existing compiler parallelizing techniques.

The algorithm consists of the following steps.

2.1 Construction of Appropriate Intermediate Representation

For the effective program parallelization, one should use a suitable intermediate representation (IR) of the source program. Such an IR must contain program properties that are essential for the algorithm and easy to handle. We use data dependence graph (DDG) as such IR. This IR is also used in [2]. DDG is a hierarchical data structure. The algorithm operates on this graph and is recursively applied to each level of hierarchy.

The nodes of DDG are simple statements (such as assignment or addition statement) or complex statements (such as loops or conditional statements) of a source function. These complex nodes are in turn DDGs that may be expanded on the below level. An edge exists between two nodes of the graph, if there is data dependency between them. We take into account " read-write", "write-read" and "write-write" dependencies. To present side effects of the source function on the graph we introduce special node corresponding to the return statement of the function, called EXIT. If a node generates side effects, there is an edge between this node and EXIT node in the graph.

It can be noted that loop data dependencies of the nodes are hidden on different levels of hierarchy. Therefore, on each level of DDG it is directed acyclic graph (DAG). DDG for our example function is shown in the Fig. 1. The construction of DDG involves performing of reaching definitions analysis, which in turn needs alias analysis to be done.

2.2 Construction of the Cost Model

Our goal is to construct such a program separation onto threads that would increase the number of locality events as much as possible. We introduce a cost model to estimate locality properties of a separation. The model includes weights of DDG nodes, the number of locality events and synchronization costs.

Each node of DDG gets its weight based on execution time of this node. The time is estimated by program profiling. The processor cache is flushed where necessary during profiling to allow modeling of locality events at later stages of the algorithm. To obtain precise timings high-resolution hardware performance counters are used.

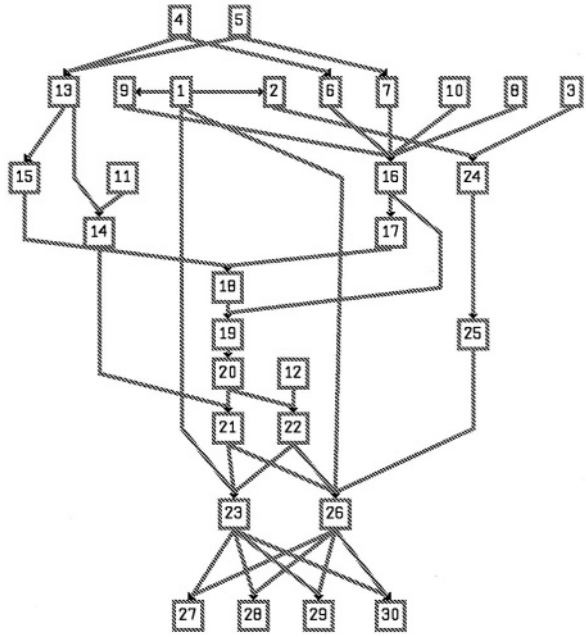


Fig. 1. Data dependence graph of an example function

Initial weights of DDG nodes are based on execution times only and do not reflect locality properties. Locality events are considered during the thread separation stage of the algorithm. The process of recalculating the weights of the nodes is described in the next section. Weights for DDG of example function are shown in the Fig. 1.

2.3 Thread Partitioning

On this stage of the algorithm actual thread partitioning is performed. We iterate over the list of unassigned nodes. On each step we try to assign a thread for some node from the list (i.e., we perform list scheduling). The best fitting thread for the node is a thread, which minimizes the criterion function that is calculated using currently inserted node. The function considers total execution time of the program, i.e. maximum of thread execution times, and mean difference of thread execution times.

When adding a node to a thread, we correct our cost model. It means that thread execution time is adjusted accordingly to happened locality and synchronization events. If some data dependencies are not satisfied before execution of the node, synchronization primitives are to be inserted into the current threads and dependent threads. Current node may start execution only after the end of synchronization(s); thread execution time increased by the time spent for waiting for other threads and synchronizing with them. For correctness of such

algorithm the predecessors of current node in DDG must be processed before the current node. Such traversal is possible because DDG is acyclic graph on one level of hierarchy.

We must also consider locality issues while adding a node to a thread. For this purpose we perform modeling of a cache for each thread. For each use of a node, we model a cache query. If there was a cache hit, thread execution time is decreased by amount of time we save while fetching or saving node's operand.

2.4 Additional Optimizations

Program separation onto threads is already done on this stage. After that, it is possible to perform additional program transformations to increase locality of these threads. We use data packing [5] as such optimization. This transformation is done by packing closely in the memory all objects that are accessed by sequential instructions. The optimization improves spatial locality of the thread. It must be performed for all threads at once.

2.5 Formal Description

A pseudocode of the algorithm is as follows:

```
thread_partitioning(DDG g = (V, E), int thread_count) {
  Threads = new Thread[thread_count];
  foreach(n in V) {
    best_thread = none;
    foreach(thread in Threads) {
      thread.add(n);
      if (thread.better(best_thread)
        best_thread = thread;
      thread.remove(n);
    }
    best_thread.add(n);
  }
  return Threads;
}

Thread.add(DDGNode node) {
  //adjust synchronization issues
  new_time = this.time;
  foreach (p in Pred(node)) {
    if (p.thread != this &&
    not_synchronized(p.thread, this) {
      //add synchronization time to other thread
      wait = p.thread.add_synchronization(this);
      //find thread's new time
      new_time = MAX(new_time, wait);
```

```
    }
  }
  this.time = new_time;
  //add node execution time
  this.time += node.time;
  //adjust locality issues, considering cache hits
  foreach (v in Uses(node)) {
    if (this.cache.hit(v))
      this.time -= Cache.HIT_PRICE;
  }
}
```

3 Evaluation of the Algorithm

At first, we have manually applied the algorithm to the function `expand_start_bindings_and_block` from the source code of GNU C compiler, version 3.2. We chose the function, which is only 20 lines long for simplicity. The function does not contain loops.

All instruction execution times were taken from the TSC (time stamp) counter of Intel Pentium III processor. All locality events are counted as happened in L1 cache. We assumed that it takes 20 cycles to fetch an operand from L2 cache and 2 cycles to fetch it from L1 cache, so each cache hit saves 18 cycles. We also assumed that synchronization takes 30 cycles to complete. Our results showed that total execution time decreased from 2589 cycles to 1522 cycles on three threads (1.7 times speedup).

Then the algorithm was implemented in Java programming language. An input for the algorithm is the DDG of a program. For reasons of simpler modeling we consider all variables as having the same size. N-way set associative cache is modeled for counting locality events. Basic parameters of the implementation are number of threads, synchronization price, cache hit price, cache size, cache line size, and cache set size.

Evaluation of the algorithm was performed on 4-processor Itanium machine running RedHat Linux 7. For evaluation we used our example function, which intensively uses complex arithmetic. During function partitioning we made assumptions similar to those described above.

The methodology of evaluation is as follows. Both original and partitioned versions of the quadric program were run for 100 times, and execution time was measured. We use AR.ITC register on Itanium for accurate measurement of elapsed processor ticks for each run. Average μ and root-mean-square deviation σ of each sequence was calculated. Then all values that do not belong to the range $[\mu - 2\sigma, \mu + 2\sigma]$ were removed from the sequence. Then the average was recalculated. This value was used for speedup calculation.

The test program on Itanium was compiled with GCC 3.3.1 with different optimization level (-O0, -O1, -O2). The results are shown in the Fig. 2.

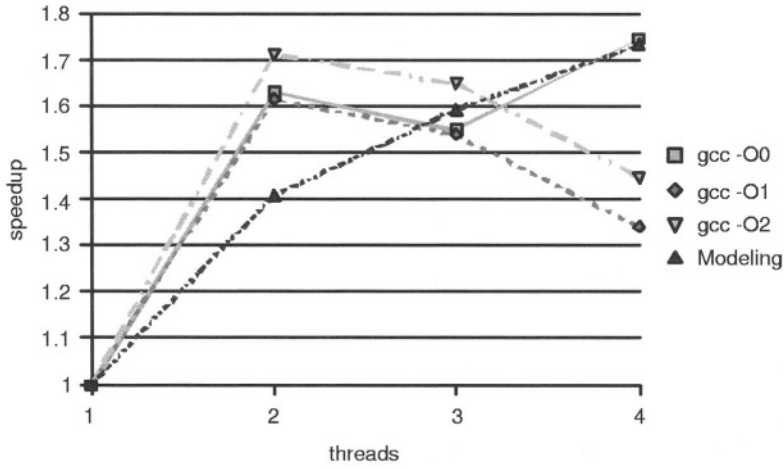


Fig. 2. Evaluation results for an example function

We have developed and implemented a simulator to allow thorough testing of the algorithm. The simulator generates random DDGs that have properties close to the graphs of real programs. Then our algorithm is applied to these graphs.

The results of simulation are introduced in the Fig. 3. We have generated graphs with 500 nodes and different degrees of dependences between them. Graphs with low degree of dependence between nodes are easier to partition. That is because our algorithm uses greedy heuristics to choose the thread for each node during partitioning. We think that allowing backtracking of possible choices during partitioning would enhance results for heavy dependent graphs, although this would slow down the partitioning process. This is a topic of our further research.

4 Related Work

One of the papers relevant to our work is [2]. This work introduces an algorithm for thread partitioning for a system based on non-preemptive multithreaded processors. Every program instruction is either “local” or “remote”. Remote operations can be overlapped with others. The goal of the work is to minimize total execution time of the threads, performing all remote operations as split-phase transactions to hide their long latencies (i.e., one thread initializes an operation, while another one uses the returned value). This problem is shown to be NP-hard [2]. Tang uses list-scheduling based heuristic to find the solution that is close to the optimal.

Our algorithm is developed for the systems with shared memory, so our execution model does not include remote operations. We consider the systems

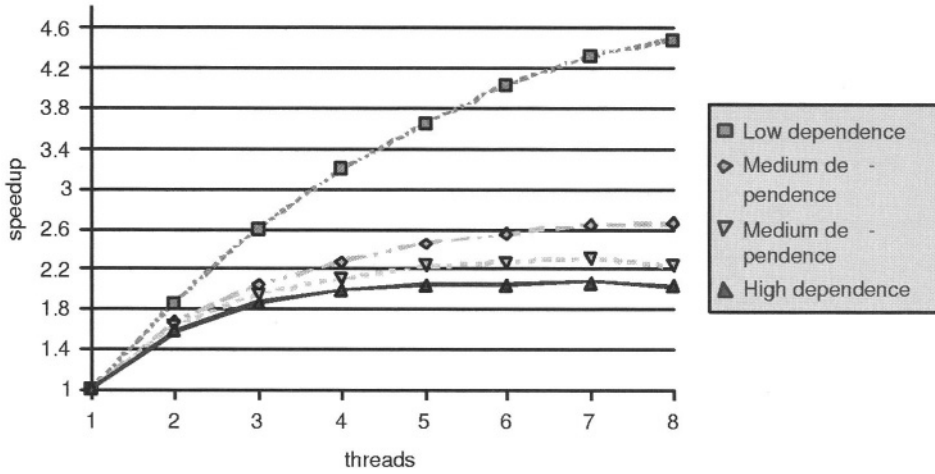


Fig. 3. Evaluation results on generated DDGs

with preemptive multithreaded architecture. Our goal is to perform such a thread partitioning that at most utilizes processor caches and reveals natural parallelism of the program. The solution of this problem would provide more benefit on existing systems with shared memory.

Table 1. Comparison of our approach and algorithm of [4]

Dependence /	Low		Medium		High	
Threads	We	Moreira	We	Moreira	We	Moreira
1	1.000	0.742	1.000	0.665	1.000	0.586
2	1.668	1.468	1.624	1.322	1.568	1.155
3	2.034	2.152	1.949	1.823	1.861	1.583
4	2.262	2.609	2.096	1.966	1.981	1.707
5	2.452	2.740	2.218	2.001	2.034	1.733
6	2.564	2.807	2.259	2.030	2.035	1.747
7	2.659	2.855	2.294	2.026	2.066	1.746
8	2.659	2.884	2.224	2.040	2.019	1.748

Another work related to our problem is [4]. This work uses hierarchical task graph (HTG), introduced in [3], as an intermediate representation. This graph contains both data and control dependencies. In our work, DDG contains only data dependencies. This doesn't lead to the loss of generality, because all kinds of dependencies can be unified as noted in [4].

When performing task scheduling for the systems with shared memory, [4] uses one global task queue with FIFO structure. A thread composed of several tasks can be scheduled on different processors during its execution. Task

scheduling algorithm does not aim at building in a manner some optimal schedule. Our algorithm tries to build a thread partition that is close to optimal in the sense of our cost model. Each thread is bound to the concrete processor (more specifically, to one cache), because it is the only way to benefit from locality events.

To compare our approaches we have implemented Moreira's algorithm. Our implementation takes DDGs as an input and models run-time autoscheduling described in [4]. We have compared two approaches on the same DDGs generated by our simulator. Results of this evaluation are shown in the Table 1. In this table the speedups of parallelized DDG with respect to sequential algorithm are shown. It can be seen that both approaches produce comparable results, though our approach is better on small number of threads (up to four). Moreira's algorithm performs better when degree of data dependence in the program is relatively low. That is because of dynamic nature of this algorithm, which has more possibilities of parallelizing non-dependent DDG nodes at run time.

5 Conclusions

Our work introduces an algorithm of program thread partitioning, which improves locality property of the program. Results of evaluation of our algorithm on example function and generated DDGs show feasibility of our approach.

Future work includes evaluation of the algorithm on real-world applications. It is also necessary to study the means for improving the algorithm by using different criterion functions for choosing the best thread and using the results of program profiling. For example, the algorithm can be recursively applied to some heavily executed piece of source code.

References

1. M. E. Wolf and M. Lam. A data locality optimizing algorithm. In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Toronto, CA, June 1991
2. X. Tang, J. Wang, K. Theobald, and G. R. Gao. Thread partitioning and scheduling based on cost model. ACAPS Tech. Memo 106, Sch. of Comp. Sci., McGill U., Montréal, Qué., Apr. 1997.
3. M. Girkar and C. Polychronopoulos. Automatic extraction of functional parallelism from ordinary programs. In IEEE Transactions on Parallel and Distributed Systems, pp. 166–178, 1992.
4. J.E. Moreira. On the implementation and effectiveness of autoscheduling for shared-memory multiprocessors. PhD. thesis, Department of Electrical and Computer Engineering, Univ. of Illinois at Urbana-Champaign, 1995.
5. C. Ding and K. Kennedy. Improving cache performance in dynamic applications through data and computation reorganization at run time. In ACM SIGPLAN'99 Conference on Programming Language Design and Implementation, pages 229–241, Atlanta, GA, 1999.

Parallel Branch-and-Bound Skeletons: Message Passing and Shared Memory Implementations

Isabel Dorta, Coromoto Leon, and Casiano Rodriguez

Departamento de Estadistíca, I.O. y Computación,
Universidad de La Laguna, E-38271 La Laguna, Tenerife, Spain
{isadorta, cleon, casiano}@ull.es
http://nereida.deioc.ull.es

Abstract. The aim of this article is to describe the design and the implementation of skeletons for the Branch-and-Bound technique. The library developed provides sequential and parallel resolution patterns to solve optimization problems through this technique on a wide range of parallel computing platforms. To use the skeletons proposed to solve a particular problem, the skeleton filler has to specify the type of the problem, the type of the solution and the specific characteristics of this paradigm using the C++ programming language, and the user does not need to have any knowledge in parallelizations techniques. This paper describes the sequential and the parallel algorithms for message passing and shared memory platforms, and shows computational results for the 0-1 knapsack problem on an Origin 3000.

1 Introduction

Branch and Bound (BnB) is a common search technique to solve problems in combinatorial optimization. BnB improves over exhaustive enumeration, because it avoids the exploration of those regions of the solution space, where it can be certified that no solution improvement can be found. BnB constructs a solution of a combinatorial optimization problem by successive partitioning of the solution space. The *branch* refers to this partitioning process. The *bound* refers to lower bounds that are used to construct a proof of optimality without exhaustive search. The exploration of the solution space can be represented by a search tree, whose nodes represent sets of solutions, which can be further partitioned in mutually exclusive sets. Each subset in the partition is represented by a child of the original node. An algorithm that computes a bound on the value of any solution in a given subset prevents further searches from a given node if the best value found so far is largest than the value of the best solution that can be obtained from the node (bound computed at the node). In this case the node is killed and no children need to be searched; otherwise it is alive.

This work presents an Object Oriented skeleton which implements the process described in the previous paragraph. The implementation of the skeleton

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 286–291, 2004.
© Springer-Verlag Berlin Heidelberg 2004

has been made in C++. Sequential code and parallel code of the invariant part of the BnB resolution pattern is provided. The users must fill a blueprint in order to solve an specific problem. The classes which compose this blueprint are used to establish the relation between the main solver and the problem. Once the user has represented the problem, he/she obtains for free a parallel solver without any additional effort. The skeleton provides modularity for the design of exact algorithms, which supposes a great advantage with respect to the direct implementation of the algorithm, not only in terms of code re-usability but also in methodology and clarity of concepts. The skeleton presented, MaLLBa::BnB is part of the MaLLBa project [1,2], whose final objective is to provide exact, heuristic and hybrid skeletons to solve combinatorial optimization problems.

The rest of the article is organized as follows. In the second section the design and implementation of the sequential and parallel resolution patterns are described. MaLLBa::BnB is used to solve the 0-1 Knapsack Problem in the third section and computational results are shown. Last section presents the conclusions and future work.

2 MaLLBa::BnB Skeletons

Formally, a combinatorial optimization problem [5] is defined as a tuple $\Pi = (I, S, f, g)$ where:

- I is the set of instances of Π. If $x \in I$ we say that x is an *instance* (or an input) of Π.
- Given an instance $x \in I$, $S(x)$ denotes the *set of feasible solutions* of x.
- For any instance $x \in I$ and any feasible solution $\sigma \in S(x)$, $f(x, \sigma)$ represents a real value, the measure (or cost or fitness) of σ with respect to Π and x. The function f is called the *objective function.*
- $g \in \{\max, \min\}$. The goal of Π is to find a feasible solution that optimizes f according to g: given an input $x \in I$, determine an *optimal solution* $\sigma' \in S(x)$ such that $f(x, \sigma') = g\{f(x, \sigma) \mid \sigma \in S(x)\}$.
- A *subproblem* Π_i is a tuple $\Pi_i = (I, S_i, f, g)$ where $S_i(x)$ is a subset of the underlying space I.

This section is devoted to explain the structure of BnB skeletons. The explanation is based on the problem definition established above.

2.1 Sequential Skeleton

Figure 1 shows the pseudo-code of the BnB algorithm for a maximization problem. During the BnB computation, *subproblems* are successively generated and tested. Given a *subproblem* Π_i, it can be decomposed into $\Pi_{i1}, \Pi_{i2}, ..., \Pi_{ik}$ by a branching operation where $S_i = \bigcup_{j=1}^{k} S_{ij}$. Thus any feasible solution $\sigma \in S_i$ belong to some S_{ij} and conversely any $\sigma \in S_{ij}$ belong to S_i. Let $\mathcal{Q}$ denote the set of *subproblems* currently generated. A *subproblem* $\Pi_i \in \mathcal{Q}$ that is neither decomposed nor tested yet is called *live.* The set of *live subproblems* are denoted

```
L := {Π0}; Q := {Π0};
bs := −∞; T := ∅;
while (L ≠ ∅) {
    Πi := s(L);
      if (upper_bound(Πi) > bs) {
        if (lower_bound(Πi) > bs) {
            bs := lower_bound(Πi);
            T := {σ};
            // σ satisfies f(x,σ) = lower_bound(Πi);
        }
        else {
            (branch) decompose Πi into Πi1, Πi2, ..., Πik
            L := L ∪ {Πi1, Πi2, ..., Πik} - {Πi} ;
        }
        Q := Q ∪ {Πi1, Πi2, ..., Πik} ;
    }
    L := L - Πi;
}
return bs // bs is the best solution value
return S // S is the best solution
```

Fig. 1. Pseudo-code of sequential BnB method (maximization case).

by $\mathcal{L}$. For each tested subproblem in $\mathcal{Q}$ its *lower bound* and *upper bound* are computed. The greatest lower bound obtained so far is called the *best solution value* and denoted by *b*s. The solution realizing *bs* is called the *best solution* and stored in $\mathcal{T}$. The algorithm proceeds by repeating the test of lives subproblems. The selection of a live subproblem for the next test is done by a *search function* s, such that, $s(\mathcal{L}) = \mathcal{L}$.

The next two sections explain in detail how the BnB skeleton has been parallelized. Two different implementations are provided, one of them based on the message passing paradigm [11] and another based on the shared memory [9] one.

2.2 Message Passing Skeleton

The message passing parallel version uses a master-slave scheme. The generation of new subproblems and the evaluation of the results of each of them are completely separated of the individual processing of each subtask. The Master is responsible of the coordination between subtasks. The Master has a data structure *busy* where registers the occupation state of each slaves; at the beginning all the slaves are idle. The initial subproblem, the best solution and the best value of the objective function are sent to an idle slave. While there are no idle slaves and the queue is not empty the Master receives information from them and decides the next action to apply depending whether the problem is solved, whether there is a slave request or whether the slave has not work to do. If the problem is solved, the solution is received and stored. When the master receives a request for a certain number of slaves, it is followed by the upper bound value.

```
L := {Π0}; Q := {Π0}; bs := −∞; T := ∅;
P := {1, ..., p};
while (L ≠ ∅) {
    while ( P ≠ ∅) {
        n := s(P); P := P − {n}; Πn := s(L); L := L − Πn;
    }
    parallel for ( n ∈ P) {
      if (upper_bound(Πn) > bs) {
        if (lower_bound(Πn) > bs) { – critical region
            bs := lower_bound(Πn);
            T := {σ}/f(x, σ) = lower_bound(Πn);
        }
        (branch) decompose Πn into Πn1, Πn2, ..., Πnk – critical region
        L := L ∪ {Πn1, Πn2, ..., Πnk} - {Πn} ;
        Q := Q ∪ {Πn1, Πn2, ..., Πnk} ;
    }
}
```

Fig. 2. Shared Memory MaLLBa::BnB skeleton

If the upper bound value is better than the actual value of the best solution, the answer to the slave includes the number of slaves that can help to solve its problem. In other case, the answer indicates that it is not necessary to work in this subtree. When the number of idle slaves is equal to the initial value, the search process finishes, then the Master notifies the slaves to finish the work. A Slave works branching and bounding the problem received. New subproblems are generated calling to the branch method. The slave asks for help. If no free slaves are provided, the slave continues working locally. In other case, it removes subproblems from its local queue and sends them directly to other slave.

The implementation of MaLLBa::BnB skeleton uses MPI_Send and MPI_Recv to send and receive messages respectively. The main loop in the Master and Slaves codes are implemented using MPI_IProbe. When a message is received its status is used to classify what kind of work should be done: finish, receive a problem for bounding and branching, receive a request of slaves, etc.

2.3 Shared Memory Skeleton

The shared memory model algorithm works with a global shared queue of tasks implemented using a linked data structure (see figure 2). First, the number of threads is calculated and established. Then subproblems are removed from the queue and assigned to each thread. Through a parallel region each assigned thread works in its own subproblem. The lower and upper bounds are calculated. The best solution value and the solution vector must be modified carefully: at any time, only one thread can change the variable. This is guaranteed using the OpenMP critical directive. The same special care must be take into account when a thread tries to insert a new subproblem in the global shared queue.

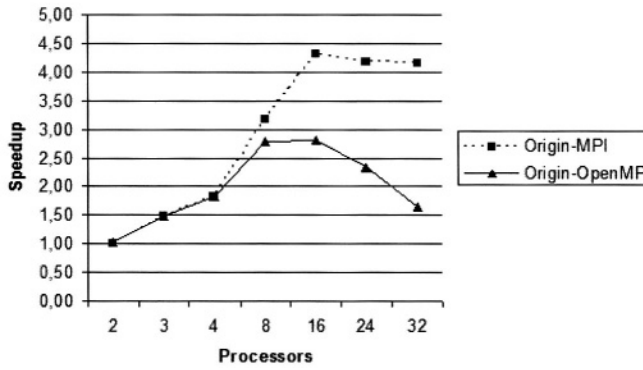


Fig. 3. Origin 3000 speedup

3 Computational Results

We consider the classical 0-1 Knapsack Problem where a subset of N given items has to be introduced in a knapsack of capacity C. The algorithm described by Martello and Toth [8] to solve this problem has been implemented using the MaLLBa::BnB skeleton. In this section we analyze the experimental behavior for this implementation on sets of randomly generated test problems. The experiments have been done on an Origin 3000, whose configuration is 160 MIPS R14000 processors at 600 MHz, 1 Gbyte of memory each and 900 Gbyte of disk. The software used in the Origin 3000 was the MPISpro CC compiler of C++ (version 7.3.1.2m) and IRIX MPI.

Figure 3 shows the speedup results of five executions of the 0-1 Knapsack Problem randomly generated for size 50,000. Only the optimal value of the objective function is calculated. The solution vector is not calculated in these experiments. Comparing the results obtained using MPI and OpenMP a similar behavior can be appreciated between both of them up eight processors. However, when the number of processors increases the speedup of the OpenMP version decrease while the MPI remains stable. The limited speedup achieved is due to the fine grain nature of the 0-1 Knapsack Problem. The branching and bounding operations take a neghsgible time.

4 Conclusion and Future Works

Several tools for the parallel implementation of general BnB algorithms using the Object Oriented paradigm have been developed [3], [7], [4], [10]. This paper presents another one: MaLLBa::BnB skeleton.

The MaLLBa::BnB skeleton offers to the user freedom to implement the data structures that represent its problem, and provides some resolution patterns that control the execution flow. The principal objective of MaLLBa::BnB is to simplify the task of researchers and users that have to implement algorithms using the Branch and Bound technique. It provides the user an added value not only in terms of the amount of code to write, but also modularity and conceptual clarity.

At present, we are working to improve the efficiency of the parallel versions. We also are looking for problems more suitable to be solve using the BnB technique. Finally, an approach that could allow a better management of the computational resources, specially in hybrid share-distributed memory architecture, is to combine data and task parallelism.

Acknowledgements. This work was partially supported by the CICYT project TIC02-04498-C05-05. We also would like to thank CIEMAT for allowing us the access to their computers.

References

1. Alba E., Almeida F., Blesa M., Cotta C., Díaz M., Dorta I., Gabarró J., González J., León C., Moreno L., Petit J., Roda J., Rojas A., Xhafa F.: MaLLBa: Towards a Combinatorial Optimization Library for Geographically Distributed Systems, Actas de las XII Jornadas de Paralelismo (2001) 105–110.
2. Alba E., Almeida F., Blesa M., Cabeza J., Cotta C., Díaz M., Dorta I., Gabarró J., León C., Luna J., Moreno L., Petit J., Rojas A., Xhafa F.: MaLLBa: A Library of skeletons for combinatorial optimisation, In Proceedings of the International Euro-Par Conference, Paderborn, Germany, LNCS 2400 (2002) 927–932.
3. Cole M.: eSkel library home page, http://www.dcs.ed.ac.uk/home/mic/sKel (2002).
4. Eckstein J., Phillips C.A., Hart W.E.: PICO: An Object-Oriented Framework for Parallel Branch and Bound, Rutcor Research Report (2000).
5. Ibaraki, T: Enumerative Approaches to Combinatorial Optimization - Part I, Annals of Operations Research,(10). Scientific, Basel, Switzerland, (1987).
6. Kuchen H.: A Skeleton Library, In Proceedings of the International Euro-Par Conference, Paderborn, Germany, LNCS 2400 (2002) 620–629.
7. Le Cun B., C. Roucairol C., The PNN Team: BOB: a Unified Platform for Implementing Branch-and-Bound like Algorithms. Rapport de Recherche n.95/16 (1999).
8. Martello S., Toth P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons Ltd (1990).
9. OpenMP Architecture Review Board, OpenMP C and C++ Application Program Interface, Version 1.0, http://www.openmp.org (1998).
10. Ralphs T.K., Ladány L.: COIN-OR: Common Optimization Interface for Operations Research, COIN/BCP User's Manual, International Business Machines Corporation Report (2001).
11. Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., Dongarra, J.J.: MPI: The Complete Reference, The MIT Press, 1996.

Selfverifying Solvers for Linear Systems of Equations in C-XSC

Carlos Amaral Hölbig[1], Paulo Sérgio Morandi Júnior[2], Bernardo Frederes Krämer Alcalde[2], and Tiarajú Asmuz Diverio[2]

[1] Universidade de Passo Fundo and PPGC-UFRGS
Campus 1 - BR 285, Passo Fundo (RS), Brazil, CEP 99001-970
holbig@upf.br
[2] Instituto de Informática and PPGC-UFRGS
Campus do Vale, Av. Bento Gonçalves, 9500,
Porto Alegre (RS), Brazil, CEP 91501-970
{sergio, bfkalcalde, diverio}@inf.ufrgs.br

Abstract. In this paper we discuss the implementation of selfverifying solvers for systems of linear equations $Ax = b$ with dense and banded matrices A and the future adaptation these solvers to high performance environments. The algorithms were implemented using C-XSC (a C++ class library for extended scientific computing). We discuss, too, the integration between C-XSC and MPI libraries on cluster computers. The main topics of our research are the development of software tools for Validated Numerics in High Performance Environments using C-XSC and MPI, the optimization of C-XSC and its use on cluster computers and the application these software tools to real life problems [5].

1 Introduction

One of the most frequent tasks in numerical analysis is the solution of linear systems of equations

$$Ax = b \tag{1}$$

with an $m \times n$ matrix A and a right hand side $b \in \mathbb{R}^n$. Many different numerical algorithms contain this task as a subproblem. Because of these aspects, this work aims the development of selfverifying solvers for linear systems of equations and the adaptation of the algorithms implemented to cluster computers using C-XSC library (see details about this library in [3] and [4]). Our solvers work with dense and sparse (in special banded matrices) linear systems of equations. Nowadays, the solver for dense matrices works with all four basic numerical C-XSC data types: *real, interval, complex,* and *complex interval* and the solver for sparse matrices works with *real* and *interval* data types. All our programs are freeware (C++ templates and the C++ exception handling are not used in the actual implementations, these characteristics will be used in future versions of our solvers).

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 292–297, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 The Algorithms

The algorithms implemented in our work were described in [6] and can be applied to any system of linear equations which can be stored in the floating point system on the computer. They will, in general, succeed in finding and enclosing a solution or, if they do not succeed, will tell the user so. In the latter case, the user will know that the problem is very ill conditioned or that the matrix A is singular. In the implementation in C-XSC, there is the chance that if the input data contains large numbers or if the inverse of A or the solution itself contain large numbers, an overflow may occur, in which case the algorithms may crash. In practical applications, this has never been observed, however. This could also be avoided by including the floating point exception handling which C-XSC offers for IEEE floating point arithmetic [2].

For this work we implemented interval algorithms for solution of linear systems of equations with dense and sparse matrices. There are numerous methods and algorithms computing approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigour if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned. We present some algorithms which answer the questions about existence and accuracy automatically once their execution is completed successfully. Even very ill conditioned problems can be solved with these algorithms. Most of the algorithms presented here can be found in [7].

3 Solvers for Dense and Sparse Linear Systems

The C-XSC programs implemented in solver for dense linear systems were written for the case of *real* input data (i.e. A is of type *rmatrix* and b is of type *rvector*) and for the case of the data types *interval, complex,* and *complex interval.* The changes made for the use of these other types are mainly changes of the data type of certain variables and functions in the program. This solver has two modules: the module `lss_aprx` contains the function `MINV` which computes an approximate inverse of the input matrix A of type *rmatrix* using the Gauss-Jordan algorithm (see i.e. [8]), when A is a square matrix. In the over- or under-determined case we use the Moore-Penrose pseudo inverse A^+ of A (if A has full rank). The second module `lss` contains the functions which solve the dense linear system. This system may be square and non square $(m \times n)$. In the over-determined case $(m > n)$ a vector $x \in \mathbb{R}^n$ is sought whose residuum $b - Ax$ has minimal Euclidian norm whereas in the under-determined case $(n < m)$ a solution $x \in \mathbb{R}^n$ is sought which has minimal norm. Example solved with this solver is showed in Sect. 4.

For the solution of a sparse linear system we present an implementation of an algorithm to compute efficiently componentwise good enclosures. Our implementation works with point as well as *interval* data (data afflicted with tolerances).

We assume linear systems whose coefficient matrix has a banded structure. In this case the well known general algorithm (using the Krawczyk operator) to solve systems with dense matrices is not efficient. Since the approximate inverse R of a banded matrix A is in general a full matrix a lot of additional storage would be required, especially if the bandwidth of A is small compared with its dimension. So a special algorithm is used to reduce the amount of storage and runtime. This method is based on the fact that matrices with banded structure are closely related to difference equations. For the banded system, we apply a LU-decomposition without pivoting (to avoid fill in) to the coefficient matrix A and derive an interval iteration similar to the well known interval iteration used in case of dense matrices. Here, however, we do not use a full approximate inverse R, but rather the interval iteration will be performed by solving two systems with banded triangular matrices L and U. The banded triangular systems are solved with the special method for difference equations described in [6]. In case of point matrices the method is designed to give almost sharp enclosures for all components (large or small in modulus) of the solution vector. A different approach to compute an enclosure for the solution vector of a large linear systems with banded or arbitrary sparse coefficient matrix (which gives enclosures with respect to the infinity norm $||\ ||_\infty$ only) is described in [7].

In addition to the implementation of the solution method in C-XSC, the program includes a small demonstration part (a driver) which can be used to solve some simple systems. First the program reads the number of lower and upper bands and then one value for each of the bands, i. e. initially a Toeplitz matrix is generated. In the next step, however, any number of elements of the matrix can be changed, such that arbitrary banded matrices can be entered. To change the element $a_{i,j}$, only i, j and the new value for this element must be entered. Changing of elements is finished by entering zeros for i and j. Next the right hand side must be entered. There are several choices of predefined solutions, such that the right hand side b will be determined from this given solution. Alternatively b can be set to a constant value in all components or all components can be entered successively. In any case, the values of the components of b may be changed again similarly as for the matrix. When no changes are done anymore, the solution algorithm starts. The banded solver is called and the solution and error statistics are printed. In this way it is quite easy to explore the our C-XSC solver. Example solved with this solver is showed in next section.

4 Tests and Results

A very well known set of ill conditioned test matrices for linear system solvers are the $n \times n$ Hilbert matrices H_n with entries $(H_n)_{i,j} := \frac{1}{i+j-1}$. As a test problem, we report the results of our program for the linear systems $H_n x = e_1$, where e_1 is the first canonical unit vector. Thus the solution x is the first column of the inverse H_n^{-1} of the Hilbert matrix H_n. Since the elements of these matrices are rational numbers which can not be stored exactly in floating point, we do not solve the given problems directly but rather we multiply the system by the

least common multiple lcm_n of all denominators in H_n. Then the matrices will have integer entries which makes the problem exactly storable in IEEE floating point arithmetic. For $n = 20$, we have $lcm_{20} = 5342931457063200$. For the system $(lcm_{20}H_{20})x = (lcm_{20}e_1)$, the program computes the enclosures (here an obvious short notation for intervals is used) showed in (2), which is an extremely accurate enclosure for the exact solution (the exact solution components are the integers within the computed intervals).

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \\ x_{19} \\ x_{20} \end{pmatrix} = \begin{pmatrix} [\ 3.999999999999999\text{E}{+}002,\ 4.000000000000001\text{E}{+}002] \\ [\text{-}7.980000000000002\text{E}{+}004,\ \text{-}7.979999999999998\text{E}{+}004] \\ [\ 5.266799999999999\text{E}{+}006,\ 5.266800000000001\text{E}{+}006] \\ [\text{-}1.716099000000001\text{E}{+}008,\ \text{-}1.716098999999999\text{E}{+}008] \\ [\ 3.294910079999999\text{E}{+}009,\ 3.294910080000001\text{E}{+}009] \\ [\text{-}4.118637600000001\text{E}{+}010,\ \text{-}4.118637599999999\text{E}{+}010] \\ [\ 3.569485919999999\text{E}{+}011,\ 3.569485920000001\text{E}{+}011] \\ [\text{-}2.237302782000001\text{E}{+}012,\ \text{-}2.237302781999999\text{E}{+}012] \\ [\ 1.044074631599999\text{E}{+}013,\ 1.044074631600001\text{E}{+}013] \\ [\text{-}3.700664527560001\text{E}{+}013,\ \text{-}3.700664527559999\text{E}{+}013] \\ [\ 1.009272143879999\text{E}{+}014,\ 1.009272143880001\text{E}{+}014] \\ [\text{-}2.133234304110001\text{E}{+}014,\ \text{-}2.133234304109999\text{E}{+}014] \\ [\ 3.500692191359999\text{E}{+}014,\ 3.500692191360001\text{E}{+}014] \\ [\text{-}4.443186242880001\text{E}{+}014,\ \text{-}4.443186242879999\text{E}{+}014] \\ [\ 4.316238064511999\text{E}{+}014,\ 4.316238064512001\text{E}{+}014] \\ [\text{-}3.147256922040001\text{E}{+}014,\ \text{-}3.147256922039999\text{E}{+}014] \\ [\ 1.666194841079999\text{E}{+}014,\ 1.666194841080001\text{E}{+}014] \\ [\text{-}6.044040109800001\text{E}{+}013,\ \text{-}6.044040109799999\text{E}{+}013] \\ [\ 1.343120024399999\text{E}{+}013,\ 1.343120024400001\text{E}{+}013] \\ [\text{-}1.378465288200001\text{E}{+}012,\ \text{-}1.378465288199999\text{E}{+}012] \end{pmatrix} \tag{2}$$

As other example, we compute an enclosure for a very large system. We take the symmetric Toeplitz matrix with five bands having the values 1, 2, 4, 2, 1 and on the right hand side we set all components of b equal to 1. Then the program produces the following output for a system of size $n = 200000$ (only the first ten and last ten solution components are printed):

```
Dimension  n = 200000
Bandwidths l,k : 2 2
A = 1 2 4 2 1
change elements ? (y/n) n
b = =1
change elements ? (y/n) n
x =
  1: [  1.860146067479180E-001,  1.860146067479181E-001 ]
  2: [  9.037859550210300E-002,  9.037859550210302E-002 ]
  3: [  7.518438200412189E-002,  7.518438200412191E-002 ]
  4: [  1.160876404875081E-001,  1.160876404875082E-001 ]
```

```
  5: [  1.003153932563721E-001,  1.003153932563722E-001 ]
  6: [  9.427129202687645E-002,  9.427129202687647E-002 ]
  7: [  1.028361799416204E-001,  1.028361799416205E-001 ]
  8: [  1.005240450090008E-001,  1.005240450090009E-001 ]
  9: [  9.874921290539136E-002,  9.874921290539138E-002 ]
 10: [  1.004617422430963E-001,  1.004617422430964E-001 ]

199990: [  1.001953939326196E-001,  1.001953939326197E-001 ]
199991: [  1.004617422430963E-001,  1.004617422430964E-001 ]
199992: [  9.874921290539136E-002,  9.874921290539138E-002 ]
199993: [  1.005240450090008E-001,  1.005240450090009E-001 ]
199994: [  1.028361799416204E-001,  1.028361799416205E-001 ]
199995: [  9.427129202687645E-002,  9.427129202687647E-002 ]
199996: [  1.003153932563721E-001,  1.003153932563722E-001 ]
199997: [  1.160876404875081E-001,  1.160876404875082E-001 ]
199998: [  7.518438200412189E-002,  7.518438200412191E-002 ]
199999: [  9.037859550210300E-002,  9.037859550210302E-002 ]
200000: [  1.860146067479180E-001,  1.860146067479181E-001 ]

max. rel. error = 1.845833860422451E-016  at i = 3
max. abs. error = 2.775557561562891E-017  at i = 1
min. abs. x[3] = [ 7.518438200412189E-002,  7.518438200412191E-002 ]
max. abs. x[1] = [ 1.860146067479180E-001,  1.860146067479181E-001 ]
```

5 Integration between C-XSC and MPI Libraries

As part of our research, we did the integration between C-XSC and MPI libraries on cluster computers. This step is necessary and essential for the future adaptation of our solvers to high performance environments. This integration was developed using, initially, algorithms for matrix multiplication in parallel environments of cluster computers. We did some comparations about the time related to the computational gain using parallelization, the parallel program performance depending on the matrix order, and the parallel program performance using a larger number of nodes. We also studied some other information like the memory requirement in each method to verify the performance relation with the execution time and memory. This integration has been developed on LabTeC Cluster at II-UFRGS (cluster with 20 Dual Pentium III 1.1 GHz (40 nodes), 1 GB memory RAM, HD SCSI 18 GB and Gigabit Ethernet; cluster server (front-end) with Dual Pentium IV Xeon 1.8 GHz, 1 GB memory RAM, HD SCSI 36 GB and Gigabit Ethernet). We want to join the high accuracy given by C-XSC with the computational gain provided by parallelization. This parallelization was developed with the tasks division among various nodes on the cluster. These nodes execute the same kind of tasks and the communication between the nodes and between the nodes and the server uses message passing protocol. Measures and tests were made to compare the routines execution time in C language, C using MPI library, C using C-XSC library and C using C-XSC and MPI libraries. In the results obtained until now, the execution time of the

algorithms using C-XSC library are much larger than the execution time of the algorithms that do not use this library. Even in this initial tests, it is possible to conclude that the use of high accuracy operations make the program slower. It shows that the C-XSC library need to be optimized to have an efficient use on clusters, and make it possible to obtain high accuracy and high performance in this kind of environment.

6 Conclusions and Future Work

In our work we provide the development of selfverifying solvers for linear systems of equations with dense and sparse matrices and the integration between C-XSC and MPI libraries on cluster computers. Actually, our software run on LabTeC Cluster at UFRGS and the integration between C-XSC and MPI was done correctly. Our tests with matrix multiplication show that the C-XSC library needs to be optimized to be efficient in a High Performance Environment (up to now the main goal of C-XSC was functionality and portability, not speed).

Acknowledgement. This work is supported by FAPERGS and LabTeC/ Dell/II-UFRGS Project (Brazil).

References

1. Albrecht, R., Alefeld, G., Stetter, H. J. (Eds.): *Validation Numerics – Theory and Applications.* Computing Supplementum **9**, Springer-Verlag (1993).
2. American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic.* ANSI/IEEE Std. 754-1985, New York, 1985.
3. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C-XSC Toolbox for Verified Computing I: basic numerical problems.* Springer-Verlag, Berlin/Heidelberg/New York, 1995.
4. Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0: A C++ Class Library for Extended Scientific Computing.* Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1 (2001).
5. Hölbig, C.A., Diverio, T.A., Claudio, D.M., Krämer, W., Bohlender, G.: Automatic Result Verification in the Environment of High Performance Computing In: IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, 2002, Paris. Extended abstracts, pg. 54-55 (2002).
6. Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II – Advanced Numerical Problems.* University of Karlsruhe (1994), see *http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz.*
7. Rump, S. M.: *Validated Solution of Large Linear Systems.* In [1], pp 191–212 (1993).
8. Stoer, J., Bulirsch, R.: *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1980.

Process Decomposition via Synchronization Events and Its Application to Counter-Process Decomposition

Susumu Kiyamura, Yoshiaki Takata, and Hiroyuki Seki

Graduate School of Information Science
Nara Institute of Science and Technology
{susumu-k,y-takata,seki}@is.aist-nara.ac.jp

Abstract. In this paper, we consider a process decomposition problem which allows one to use additional synchronization events, and propose a new decomposition method. The process decomposition problem is such that given a single labeled transition system (LTS) $L_{\rm in}$, construct LTSs $L_1, \ldots, L_n$ such that the composite process of $L_1, \ldots, L_n$ is observably bisimulation equivalent to $L_{\rm in}$. The main advantage of the proposed method is that the size of the decomposed LTSs is often much smaller than known methods. Especially, we focus on simple counters and present an algorithm which decomposes a simple counter into smaller counters by using the proposed method.

1 Introduction

Process decomposition is one of the promising approaches to distributed system design. For example, in communication protocol design, we may first write a service definition which specifies the global behavior of the system, and then derive a protocol specification by decomposing the service definition into communicating processes.

The process decomposition problem (or synthesis problem) is such that given a single process $L_{\rm in}$ defined over a set A of events (or actions), construct processes $L_1, \ldots, L_n$ such that the behavior of the composite process of $L_1, \ldots, L_n$ (denoted $L_1 \times \cdots \times L_n$) is equivalent to $L_{\rm in}$. In [6], Morin extends decomposition techniques for Petri nets[1] to labeled transition systems (LTS) and presents a decidable (necessary and sufficient) condition for an LTS L to be decomposable into LTSs $L_1, \ldots, L_n$ in the sense that the state space of the composite system $L_1 \times \cdots \times L_n$ is equal to the state space of L modulo isomorphism. Morin also shows that computational complexity needed to find the optimal decomposition is NP-complete and proposes a subclass called I/O-systems for which the optimal decomposition can be found in polynomial time. In [7], Mukund generalizes the problem by considering three types of process equivalence: state space isomorphism (as in [6]), language equivalence and bisimulation equivalence. In [7], a decidable condition for a given LTS to be decomposable is given for both state space isomorphism and language equivalence. For bisimulation equivalence, a decomposition method into deterministic systems is also presented in [7] by using the fact that bisimulation equivalence coincides with language equivalence for deterministic systems.

In communication protocol design, much effort has been devoted to the study of automatic generation of a protocol specification from a service definition [8]. The main

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 298–305, 2004.
© Springer-Verlag Berlin Heidelberg 2004

feature of these studies is to allow one to use new events to synchronize decomposed LTSs. By introducing synchronization events (and hiding them as internal events from an observer), we can always decompose a given LTS into LTSs with specified sets of events[5,3]. For example, in [5], a decomposition method (for bisimulation equivalence) of an LTS into two LTSs with disjoint sets of events is proposed. In [3], the method is extended to be applicable to the decomposition into an arbitrary number of LTSs with possibly intersecting sets of events, and a method of deleting redundant synchronizations is also proposed. These methods are based on a rather simple observation: Assume that $s \xrightarrow{a} s'$ (state transition is possible from state s to state s' by event a) in an input LTS L_{in} and L_{in} is being decomposed into L_1 and L_2 where L_1 has event a and L_2 does not. Then, L_1 and L_2 synchronize with a new event immediately after L_1 executes event a to inform L_2 of L_1's executing a. A different approach is taken in [2] where a new LTS which controls the behavior of all the decomposed LTSs by using synchronization events.

In this paper, we propose a new decomposition method (for bisimulation equivalence) which uses synchronization events. The main advantage of the proposed method is that the size of the decomposed LTSs is often much smaller than the method of [5,3]. We provide a sufficient condition for an input LTS to be decomposable by the proposed method and show the correctness of the method. Several decomposition examples are shown. Especially, we focus on an LTS $CNT_n(a,b)$ which is a counter process with counting-up event a, counting-down event b and capacity n. We show that by the proposed method, $CNT_n(a,b)$ can be decomposed into $CNT_m(a,i)$ and $CNT_{n-m}(i,b)$ where m is an arbitrary integer with $1 \leq m < n$ and i is a new event for synchronization.

2 Preliminaries

Definition 1 (events). *Let Evt be a countable set of events. Evt contains a distinguished event called the* internal event, *denoted by τ, which represents an event not visible to an observer.*

Definition 2 (LTS). *A* labeled transition system *(LTS) is a 4-tuple $L = (A, S, \rightarrow, I)$ where $A \subseteq Evt$ is a subset of events called the* alphabet *of L, S is the set of states, $\rightarrow \subseteq S \times A \times S$ is the* transition relation, *and $I \in S$ is the initial state. Expression $s \xrightarrow{\alpha} s'$ represents $(s, \alpha, s') \in \rightarrow$ and $s \xrightarrow{\alpha} \cdot$ represents $s \xrightarrow{\alpha} s'$ for some s'.*

In the following, we write A [L], S [L], $\rightarrow_L$, and I [L] to represent the alphabet of L, the set of the states of L, the transition relation of L, and the initial state of L, respectively. Below we define three basic operations on LTSs: *parallel composition, event hiding,* and *contraction.* The parallel composition is also referred to as mixed product in [6] and loosely cooperating system in [7]. Let $|A|$ denote the cardinality of the set A. Let $[m, n]$ denote the set of integers greater than or equal to m and less than or equal to n.

Definition 3. *For a tuple $\widetilde{\Sigma} = (A_1, \ldots, A_k)$ of subsets of Evt, and an event α, we define $loc_{\widetilde{\Sigma}}(\alpha) = \{i \in [1,k] \mid \alpha \in A_i\}$.*

Definition 4 (concurrent events). *[6]For a tuple $\widetilde{\Sigma} = (A_1, \ldots, A_k)$ of subsets of Evt, two events $\alpha, \beta \in A_1 \cup \cdots \cup A_k$ are* concurrent, *written $\alpha \parallel \beta$, if and only if $\{\alpha, \beta\} \not\subseteq A_i$ for any $i \in [1,k]$.*

Definition 5 (parallel composition). *The* parallel composition *of a k-tuple $(L_1,\ldots,L_k)$ of LTSs is the LTS $L_1 \times \cdots \times L_k = (\bigcup_{1\le j\le k} A[L_j], S[L_1] \times \cdots \times S[L_k], \rightarrow, (I[L_1],\ldots,I[L_k]))$ where $\rightarrow$ is defined as follows: Let $\widetilde{\Sigma} = (A[L_1],\ldots, A[L_k])$. Let $s = (s_1,\ldots,s_k)$ and $s' = (s'_1,\ldots,s'_k)$ be two states in $S[L_1 \times \cdots \times L_k]$. For an event $\alpha \neq \tau$, $s \xrightarrow{\alpha} s'$ if and only if $loc_{\widetilde{\Sigma}}(\alpha) \neq \emptyset$, $s_i \xrightarrow{\alpha}_{L_i} s'_i$ for all $i \in loc_{\widetilde{\Sigma}}(\alpha)$, and $s_j = s'_j$ for all $j \in [1,k] - loc_{\widetilde{\Sigma}}(\alpha)$. Also, $s \xrightarrow{\tau} s'$ if and only if $s_i \xrightarrow{\tau}_{L_i} s'_i$ for some $i \in [1,k]$ and $s_j = s'_j$ for all $j \in [1,k]$ not equal to i.*

Definition 6 (event hiding). *Let L be an LTS and $H \subseteq Evt - \{\tau\}$ a subset of events. We define the LTS $L\backslash H = (A[L] \cup \{\tau\} - H, S[L], \rightarrow, I[L])$ where $\rightarrow$ is defined as follows: For each event $\alpha \notin H \cup \{\tau\}$, $s \xrightarrow{\alpha} s'$ if and only if $s \xrightarrow{\alpha}_L s'$; and $s \xrightarrow{\tau} s'$ if and only if $s \xrightarrow{\tau}_L s'$ or $s \xrightarrow{\beta}_L s'$ for some $\beta \in H$.*

For an equivalence relation $\sim$ on a set A and $a \in A$, let $[a]_\sim$ denote the equivalence class containing a.

Definition 7 (contraction). *Let L be an LTS, $A \subseteq Evt$ a subset of events, and $\sim$ an equivalence relation on $S[L]$ which satisfies:*

$$\text{If } s_1 \xrightarrow{\alpha}_L s_2 \text{ for some } \alpha \notin A, \text{ then } s_1 \sim s_2. \tag{c1}$$

We define the LTS $L/_\sim = (A[L] \cap A, \{[s]_\sim \mid s \in S[L]\}, \rightarrow, [I[L]]_\sim)$ where $\rightarrow$ is defined as: $h_1 \xrightarrow{a} h_2$ if and only if there exist $s_1 \in h_1$ and $s_2 \in h_2$ such that $s_1 \xrightarrow{a}_L s_2$. We also define $cont(L,A) = L/_{\sim_A}$ where $\sim_A$ is the least equivalence relation satisfying (c1).

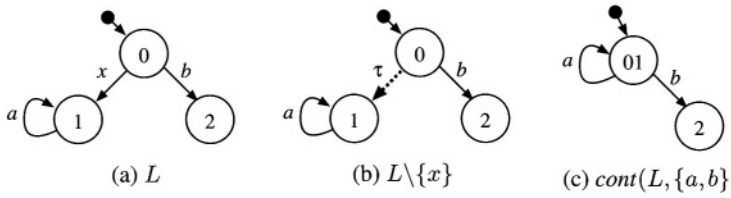


Fig. 1. Event hiding and contraction

For the LTS L in Figure 1 (a), $L\backslash\{x\}$ and $cont(L,\{a,b\})$ are shown in Figures 1 (b) and (c), respectively.In Definition 7, a relation $\sim$ is an arbitrary equivalence relation which contains at least every pair (s_1,s_2) of states such that s_1 and s_2 are connected by transitions with events not in A. In other words, $s_1 \sim s_2$ only if s_1 and s_2 are indistinguishable via events in A.

In this paper, we consider that two LTSs are *equivalent* if they are observably bisimulation equivalent[9]. Below we define the bisimulation equivalence between LTSs.

Definition 8. *Let ϵ denote the empty sequence of events. For each event $\alpha \in Evt$, $\widehat{\alpha} \in Evt \cup \{\epsilon\}$ denotes α itself if $\alpha \neq \tau$, and denotes ϵ if $\alpha = \tau$.*

Definition 9 (observable transition). *For each LTS L, we define $\Rightarrow_L \subseteq S[L] \times (Evt \cup \{\epsilon\}) \times S[L]$ as the least relation which satisfies: (a) $s \stackrel{\epsilon}{\Rightarrow}_L s$, (b) if $s \stackrel{\epsilon}{\Rightarrow}_L s'$ and $s' \stackrel{\tau}{\rightarrow}_L s''$, then $s \stackrel{\epsilon}{\Rightarrow}_L s''$, and (c) if $s \stackrel{\epsilon}{\Rightarrow}_L s'$, $s' \stackrel{\alpha}{\rightarrow}_L t'$ and $t' \stackrel{\epsilon}{\Rightarrow}_L t$, then $s \stackrel{\alpha}{\Rightarrow}_L t$.*

Definition 10 (bisimulation relation). *For a pair (L_1, L_2) of LTSs, a relation $\mathcal{R} \subseteq S[L_1] \times S[L_2]$ is an* observable bisimulation relation *(or a* bisimulation relation *for short) on (L_1, L_2) if $\mathcal{R}$ satisfies the following properties for each $(s_1, s_2) \in \mathcal{R}$ and $\alpha \in Evt$.*

(a) If $s_1 \stackrel{\alpha}{\rightarrow}_{L_1} s_1'$, then there exists $s_2' \in S[L_2]$ such that $(s_1', s_2') \in \mathcal{R}$ and $s_2 \stackrel{\widehat{\alpha}}{\Rightarrow}_{L_2} s_2'$.

(b) If $s_2 \stackrel{\alpha}{\rightarrow}_{L_2} s_2'$, then there exists $s_1' \in S[L_1]$ such that $(s_1', s_2') \in \mathcal{R}$ and $s_1 \stackrel{\widehat{\alpha}}{\Rightarrow}_{L_1} s_1'$.

Definition 11 (bisimulation equivalence). *Two LTSs L_1 and L_2* are observably bisimulation equivalent *(or* bisimulation equivalent *for short), written $L_1 \approx L_2$, if there exists an observable bisimulation relation $\mathcal{R}$ on (L_1, L_2) such that $(I[L_1], I[L_2]) \in \mathcal{R}$.*

Definition 12 (isomorphism). *Two LTSs L_1 and L_2 are* isomorphic *if there exists a bijection $\mathcal{R} : S[L_1] \rightarrow S[L_2]$ such that for each $s_1, s_1' \in S[L_1]$ and $\alpha \in Evt$, $s_1 \stackrel{\alpha}{\rightarrow}_{L_1} s_1'$ if and only if $\mathcal{R}(s_1) \stackrel{\alpha}{\rightarrow}_{L_2} \mathcal{R}(s_1')$.*

3 Decomposition Method

We define the decomposition problem by the following input and output.

Definition 13 (decomposition problem).

Input: L_{in} *a finite LTS.*
$\widetilde{\Sigma} = (B_1, \ldots, B_n)$ *a tuple of subsets of events satisfying $B_1 \cup \cdots \cup B_n = A[L_{\mathrm{in}}] - \{\tau\}$.*

Output: $L_1, \ldots, L_n$ *a tuple of finite LTSs such that $A[L_i] - A_{\mathrm{sync}} = B_i$ for each $i \in [1, n]$ and $(L_1 \times \cdots \times L_n) \backslash A_{\mathrm{sync}} \approx L_{\mathrm{in}}$, where $A_{\mathrm{sync}} = Evt - (A[L_{\mathrm{in}}] \cup \{\tau\})$.*

Thus, the decomposition problem is such that for a given LTS L_{in} and subsets of events $B_1, \ldots, B_n$ whose union is $A[L_{\mathrm{in}}] - \{\tau\}$, construct LTSs $L_1, \ldots, L_n$ such that:

- L_i $(1 \leq i \leq n)$ can use new events not in $A[L_{\mathrm{in}}] \cup \{\tau\}$ to synchronize with other L_j $(i \neq j)$;
- if we make the parallel composition $L_1 \times \cdots \times L_n$ and hide the synchronization events, then the resultant LTS $(L_1 \times \cdots \times L_n) \backslash A_{\mathrm{sync}}$ is bisimulation equivalent to L_{in}.

If we are not allowed to use events in A_{sync} for synchronization and $\tau \notin A[L_{\mathrm{in}}]$, then the decomposition problem is equivalent to the synthesis problem modulo bisimulation for loosely cooperating systems in [7].

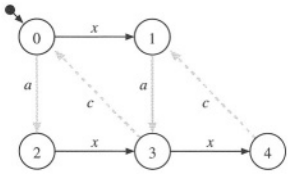


Fig. 2. Example of L_{in}

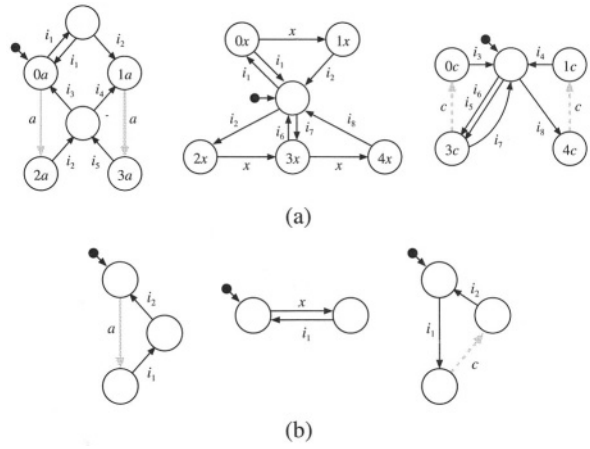


Fig. 3. Outputs for the L_{in} in Figure 2

Example 1. Suppose that L_{in} in Figure 2 and $\widetilde{\Sigma} = (\{a\},\{x\},\{c\})$ are given. Figures 3 (a) and (b) are examples of the valid outputs for this input, where $\{i_1,\ldots,i_8\} \subseteq A_{\mathrm{sync}}$, $(A[L_1], A[L_2], A[L_3]) = (\{a,i_1,i_2,i_3,i_4,i_5\},\{x,i_1,i_2,i_6,i_7,i_8\}, \{c,i_3,i_4,i_5,i_6,i_7,i_8\})$ in Figure 3 (a), and $(\{a,i_1,i_2\},\{x,i_1\},\{c,i_1,i_2\})$ in Figure 3 (b).

In the following, we fix L_{in}, n, and $\widetilde{\Sigma} = (B_1,\ldots,B_n)$ mentioned in Definition 13, and let $A_{\mathrm{sync}} = \mathit{Evt} - (A[L_{\mathrm{in}}] \cup \{\tau\})$. We propose the following method for solving the decomposition problem.

Definition 14 (the decomposition method).

Step 1: *Find a finite LTS L_{inx}, a tuple $\widetilde{\Sigma}_{\mathrm{x}} = (A_1,\ldots,A_n)$ of subsets of events, and a tuple $\mathit{Eq} = (\sim_{A_1},\ldots,\sim_{A_n})$ of equivalence relations on $S[L_{\mathrm{inx}}]$ which satisfy the following properties (p1) to (p4).*

(p1) $B_i \subseteq A_i \subseteq B_i \cup A_{\mathrm{sync}}$ *for each* $i \in [1,n]$.

(p2) $A_1 \cup \cdots \cup A_n = A[L_{\mathrm{inx}}]$.

Properties (p3) and (p4) are shown below. Note that (p1) and (p2) imply $\tau \notin A[L_{\mathrm{inx}}]$.

Step 2: *Define* $L_i = L_{\mathrm{inx}}/_{\sim_{A_i}}$ *for each* $i \in [1,n]$. *(This implies* $A[L_i] = A_i$.*)*

Properties (p3) and (p4):

(p3) There exists a total surjection $R : S[L_{\mathrm{inx}}] \to S[L_{\mathrm{in}}]$ *(i.e., each state of* L_{inx} *corresponds to exactly one state of* L_{in} *and each state of* L_{in} *corresponds to at least one state of* L_{inx}*) which satisfies properties (p3-1) to (p3-4).*

(p3-1) $I[L_{\mathrm{in}}] = R(I[L_{\mathrm{inx}}])$.

(p3-2) If $s_1 \xrightarrow{\alpha}_{L_{\mathrm{inx}}} s_2$ *and* $\alpha \notin A_{\mathrm{sync}}$, *then* $R(s_1) \xrightarrow{\alpha}_{L_{\mathrm{in}}} R(s_2)$.

(p3-3) If $s_1 \xrightarrow{\alpha}_{L_{\mathrm{inx}}} s_2$ *and* $\alpha \in A_{\mathrm{sync}}$, *then* $R(s_1) = R(s_2)$ *or* $R(s_1) \xrightarrow{\tau}_{L_{\mathrm{in}}} R(s_2)$.

(p3-4) If $s_1 \xrightarrow{\alpha}_{L_{\mathrm{in}}} s_2$, *then for each* $s_1' \in S[L_{\mathrm{inx}}]$ *such that* $R(s_1') = s_1$, *there exist* s_1'' *and* s_2' *in* $S[L_{\mathrm{inx}}]$ *which satisfy the following three properties.*

- $R(s_2') = s_2$.

– *Extending the transition relation* $\rightarrow_{L_{\mathrm{inx}}}$ *to sequences of events in the usual way,* $s_1' \stackrel{\gamma}{\rightarrow}_{L_{\mathrm{inx}}} s_1''$ *for some sequence* γ *of events in* A_{sync}.
– *If* $\alpha \neq \tau$, *then* $s_1'' \stackrel{\alpha}{\rightarrow}_{L_{\mathrm{inx}}} s_2'$. *Otherwise,* $s_1'' = s_2'$.

(p4) For each $i \in [1,n]$, *the relation* $\sim_{A_i}$ *satisfies (c1) in Definition 7 by substituting* A_i *for A. For each* $s \in S[L_{\mathrm{inx}}]$ *and* $\alpha \in A[L_{\mathrm{inx}}]$, *if there exists* s_i *and* s_i' *for each* $i \in loc_{\widetilde{\Sigma}_{\mathrm{x}}}(\alpha)$ *such that* $s \sim_{A_i} s_i$ *and* $s_i \stackrel{\alpha}{\rightarrow}_{L_{\mathrm{inx}}} s_i'$, *then for each choice of such* s_i*'s and* s_i'*'s, there exists* s' *such that* $s \stackrel{\alpha}{\rightarrow}_{L_{\mathrm{inx}}} s'$ *and* $s' \sim_{A_i} s_i'$ *for each* $i \in loc_{\widetilde{\Sigma}_{\mathrm{x}}}(\alpha)$.

Property (p3) guarantees $L_{\mathrm{inx}} \backslash A_{\mathrm{sync}} \approx L_{\mathrm{in}}$ and property (p4) guarantees $L_{\mathrm{inx}} \approx L_1 \times \cdots \times L_n$ (see Lemmas 1 and 2 shown below). Property (p3) says that L_{inx} is obtained by splitting each state in L_{in} into one or more states connected with one another by synchronization events. Property (p4) corresponds to the conditions (*i*) and *(iii)* in Theorem 4.1 of [7], which states a condition for an LTS to be decomposable modulo isomorphism without using synchronization events.

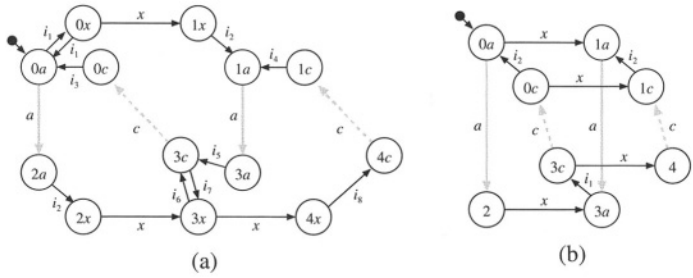


Fig. 4. L_{inx} for Figure 2

When we find a tuple $(L_{\mathrm{inx}}, \widetilde{\Sigma}_{\mathrm{x}}, Eq)$ which satisfies properties (p1) to (p4), a valid output $L_1, \ldots, L_n$ is automatically obtained by Step 2. In general, there are infinitely many L_{inx}'s which satisfy properties (p1) to (p4) for the same input, and they may differ from one another in the total size of the derived output. For example, L_{inx}'s in Figures 4 (a) and (b) satisfy properties (p1) to (p4) for the input shown in Example 1, where $\sim_{A_i}$ is the least equivalence relation satisfying (c1), and they result in Figure 3 (a) and (b), respectively. In our previous work[3], we proposed an algorithm which implements Step 1. Although the algorithm finds a tuple $(L_{\mathrm{inx}}, \widetilde{\Sigma}_{\mathrm{x}}, Eq)$ for any input $(L_{\mathrm{in}}, \widetilde{\Sigma})$, it outputs the L_{inx} in Figure 4 (a) for the L_{in} in Figure 2; that is, the algorithm outputs L_{inx} in which for any events α, β such that $\alpha \parallel \beta$ for $\widetilde{\Sigma}$, there is no state s such that $s \stackrel{\alpha}{\rightarrow}_{L_{\mathrm{inx}}} \cdot$ and $s \stackrel{\beta}{\rightarrow}_{L_{\mathrm{inx}}} \cdot$ and there are no states s_1, s_2 such that $s_1 \stackrel{\alpha}{\rightarrow}_{L_{\mathrm{inx}}} s_2 \stackrel{\beta}{\rightarrow}_{L_{\mathrm{inx}}} \cdot$.

Correctness of the Decomposition Method

Lemma 1. *If property (p3) holds, then* $L_{\mathrm{inx}} \backslash A_{\mathrm{sync}} \approx L_{\mathrm{in}}$.
(Proof Sketch) *R in property (p3) is a bisimulation relation on* $(L_{\mathrm{inx}} \backslash A_{\mathrm{sync}}, L_{\mathrm{in}})$.

Lemma 2. *If properties (p1), (p2), and (p4) hold and $L_1, \ldots, L_n$ are obtained by Step 2 in Definition 14, then $L_{\mathrm{inx}} \approx L_1 \times \cdots \times L_n$.*
(Proof Sketch) *Let $L_{\mathrm{o}} = L_1 \times \cdots \times L_n$ and $Q \subseteq S[L_{\mathrm{in}}] \times S[L_{\mathrm{o}}]$ be the relation defined as $Q = \{(s, (s_1, \ldots, s_n)) \mid s \in s_i \text{ for each } i \in [1,n]\}$. We can easily show that Q is a bisimulation relation on $(L_{\mathrm{inx}}, L_{\mathrm{o}})$.*

Theorem 1. *LTSs $L_1, \ldots, L_n$ obtained by the method in Definition 14 is a valid output of the decomposition problem.*

Definition 14 does not describe a concrete algorithm of Step 1. We are interested in finding an algorithm of Step 1 constructing L_{inx} which derives a smaller output. We discuss such an algorithm for some subclasses of L_{in}'s in the following section.

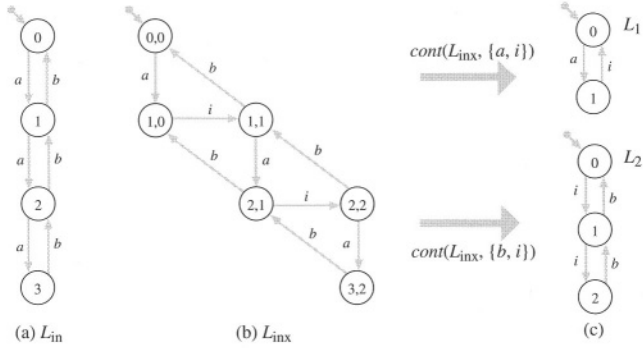


Fig. 5. Decomposition of a simple counter

4 Counter Process

In the following, for a subset $A \subseteq Evt - \{\tau\}$, let $\sim_A$ denote the least equivalence relation satisfying (c1) in Definition 7.

Definition 15 (simple counter). *Let $a, b \in Evt - \{\tau\}$ be arbitrary events and m a non-negative integer. An LTS L is a* simple counter *with the counting-up event a, the counting-down event b, and the capacity m, if $A[L] = \{a, b\}$ and L is isomorphic to the LTS $CNT_m(a,b) = (\{a,b\}, \{0,1,2,\ldots,m\}, \rightarrow, 0)$ where $\rightarrow = \{(j,a,j+1) \mid j \in [0,m-1]\} \cup \{(j,b,j-1) \mid j \in [1,m]\}$.*

Figure 5 (a) shows $CNT_3(a,b)$. In the following, we do not distinguish LTSs which are isomorphic. Thus, $CNT_m(a,b)$ is the only one simple counter with a, b and m.

Consider the decomposition problem for a simple counter $L_{\mathrm{in}} = CNT_m(a,b)$ and $\widetilde{\Sigma} = (\{a\}, \{b\})$. For this L_{in} and an arbitrary integer $l \in [1, m-1]$, we can construct the $L_{\mathrm{inx}} = (\{a,b,i\}, \{(y,x) \mid x \in [0,m-l], y \in [x,x+l]\}, \rightarrow, 0)$ where

$$\begin{aligned}\rightarrow = {} & \{((y,x), a, (y+1,x)) \mid x \in [0,m-l],\ y \in [x,x+l-1]\} \\ & \cup \{((y,x), b, (y-1,x-1)) \mid x \in [1,m-l],\ y \in [x,x+l]\} \\ & \cup \{((y,x), i, (y,x+1)) \mid y \in [1,m-1],\ x \in [\max(0,y-l), \min(y-1, m-l-1)]\}\end{aligned}$$

and $i \in A_{\mathrm{sync}}$ is a synchronization event. Figure 5 (b) shows the L_{inx} for the L_{in} in Figure 5 (a) and $l = 1$. Let $\widetilde{\Sigma}_{\mathrm{x}} = (A_1, A_2)$ and $Eq = (\sim_{A_1}, \sim_{A_2})$ where $A_1 = \{a, i\}$ and $A_2 = \{b, i\}$. Then, the tuple $(L_{\mathrm{inx}}, \widetilde{\Sigma}_{\mathrm{x}}, Eq)$ satisfies the properties (p1) to (p4) in Definition 14, and $L_1 = CNT_l(a, i)$ and $L_2 = CNT_{m-l}(i, b)$ are obtained from the tuple.

5 Conclusion

In this paper, we considered the process decomposition problem modulo bisimulation equivalence which allows using additional synchronization events, and proposed a decomposition method for an LTS such that the size of the decomposed LTSs is often much smaller than known methods. Based on the proposed method, we showed a decomposition algorithm for a subclass of LTSs called simple counters. In [4], we dissussed an extension of the algorithm to a generalized counter, which behaves as a simple counter and may preform other tasks, i.e., transitions with events other than the count-up and -down events.

Evaluation of our algorithms in the size of the output is a future study. Extending our algorithm to broader subclasses of LTSs is another interesting question.

References

1. A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures; Part II: State spaces of concurrent systems. *Acta Informatica,* 27, pp. 343–368, 1990.
2. K. Go and N. Shiratori. A decomposition of a formal specification: An improved constraint-oriented method. *IEEE Trans. Software Engineering,* 25(2), pp. 258–273, 1999.
3. S. Kiyamura, Y. Takata and H. Seki. A method of decomposing a labeled transition system into parallel processes. *Trans. of Inform. Processing Society Japan,* 42(12), pp. 2992–3003, 2001. in Japanese.
4. S. Kiyamura, Y. Takata and H. Seki. Process decmposition via synchronization events and its application to counter-process decomposition, *Technical Report of IEICE,* SS2003-4, pp. 19–24, 2003.
5. R. Langerak. Decomposition of functionality: A correctness preserving lotos transformation. In *Protocol Specification, Testing and Verification (PSTV) X,* pp. 229–242, 1990.
6. R. Morin. Decompositions of asynchronous systems. In *CONCUR '98, LNCS 1466,* pp. 549–564. Springer, 1998.
7. M. Mukund. From global specifications to distributed implementations. In *Synthesis and Control of Discrete Event Systems,* pp. 19–35. Kluwer Academic Publishers, 2002.
8. K. Saleh. Synthesis of communications protocols: An annotated bibliography. *ACM SIGCOMM Computer Communication Review,* 26(5), pp. 40–59, 1996.
9. C. Stirling. Modal and temporal logics for processes. In *Logics for Concurrency: Structure versus Automata, LNCS 1043,* pp. 149–237. Springer, 1996.

Exception Handling Model with Influence Factors for Distributed Systems

Paweł L. Kaczmarek and Henryk Krawczyk

Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology, Poland
{pkacz, hkrawk}@eti.pg.gda.pl

Abstract. Program specification is clearly identified in sequential programing where it has its standard and exceptional transitions. The paper extends the model of program specification to distributed environments where a set of control drivers is taken into account. The model includes the analysis of program specification for distributed data and multiprocess communication. Further, the impact of distributed hardware is discussed for both deterministic and indeterministic systems. The general model was implemented in CORBA to test various system designs behavior. The implementation is a simulator of distributed applications that measures reliability characteristics for different exception handling strategies.

1 Sequential Exception Handling Model – SEH

The exception handling (eh) mechanism is a well known structure and is commonly used in object oriented programming. The main idea of exception handling is the separation of normal and exceptional control flow. A *guarded region* is defined as a part of program code where exceptions are handled in a specified way by *handling functions* [Jal94]. If an exception occurs, the execution of the normal program code is suspended and a handling function is called.

F. Cristian [Cri95] proposed an eh model that divides the program specification into standard and exceptional. A program is modeled as a transition from an initial state s to a final state s'. The states s, s' are in S that is the set of all possible data values. A standard specification (goal) is a relation between initial and final states, we note:

$$G_\sigma \subseteq S \times S \tag{1}$$

A pair $(s, s') \in S \times S$ in G_σ is the intended outcome of a program P. Standard domain (SD) is the set of all possible s, such that $(s, s') \in G_\sigma$, i.e.:

$$SD \equiv \{s \mid \exists s' \ : (s, s') \in [P]_\sigma \ \& \ (s, s') \in G_\sigma\} \tag{2}$$

Apart from the standard specification the program defines also exceptional specification (*ED*) when it can not support its defined functionality (for a detailed description please refer to [Cri95]). In this situation an exception is signaled, we note: $ED \equiv S - SD$ what means exceptional specification is a specification for data in the exceptional domain.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 306–313, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Moreover, there might exist a set of data for which the specification of a program is not defined - the set is described as Unanticipated Input (*UI*). The designer of the program did not expect data in this domain. In contrast to UI there exists Anticipated Domain ($AD \equiv S - UI$) - it could be either standard or exceptional (*AED*). Finally if an error happens in the program, the behavior is different than expected according to the specification. In this situation the input data is in Fault Domain (*FD*). In general:

$$Dom \equiv SD \cup AED \cup FD \cup UD \tag{3}$$

According to F.Cristian [Cri95] the whole set S could be divided in four different areas as it is shown in Fig. 1.

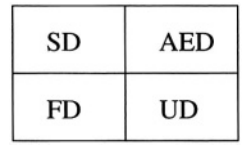


Fig. 1. The partition of input states for sequential programming

Cristian's model is very general and assumes the existence of the set of all possible input and output data, which could be difficult to identify in practice [All00]. However it covers all possible situations in sequential programming.

2 Extending SEH Model to Distributed Applications

A distributed environment differs significantly from a sequential one, however the main idea remains the same: input data is transformed according to a given specification to produce output data. A distributed system can be modeled as shown in Fig. 2.

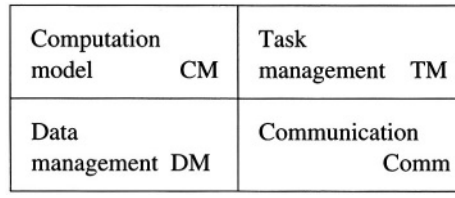


Fig. 2. Distributed system drivers

In relation to models shown in Fig. 1 and Fig. 2, we have designed a distributed exception handling (DEH) concept. We can distinguish the following representative cases:

- DEH for applications running on a single work station (*DEH/S*),
- DEH for applications running in a deterministic multiprocessor environment (D^2EH/M),
- DEH for applications running in a nondeterministic multiprocessor environment (*NDEH/M*).

Each model is an extension of the preceding one and incorporates new features that were shown in Fig. 2. The details of the models are described further in the paper. The last case (*NDEH/M*) describes the most general situation and could be adopted to any system behavior by tuning suitable parameters.

2.1 Distributed Application Model Running on One Processor (*DEH/S*)

First we focus on the Computation model and Data flow of an application for the *DEH/S* model. In this case we make a simplification and ignore the impact of the environment, i.e. Task management and Communication. We denote by:

$$P_{distr} \equiv \{P_1, P_2, \ldots, P_k\} \equiv P_{inout} \cup P_{inter} \tag{4}$$

processes that run in a distributed environment, where

$$P_{inout} \equiv \{P_{inout\ 1}, P_{inout\ 2}, \ldots, P_{inout\ l}\} \subseteq P_{distr} \tag{5}$$

$$P_{inter} \equiv \{P_{inter\ 1}, P_{inter\ 2}, \ldots, P_{inter\ m}\} \subseteq P_{distr} \tag{6}$$

are input/output and internal processes respectively. The P_{inout} processes could be further divided into input P_{in} and output P_{out} ones (the input and output processes sets are usually joint). Internal processes are responsible for the business logic of an application. The whole structure of P_{distr} defines the Computation model of an application. Each individual process has a specification that is similar to a sequential program's one. However, the specification of the whole system is more complex and consists of specifications of individual units including input and output domains. Note, that a process may receive an exception on input, so it also needs a specification for exceptions.

Data management and Computation model create the specification of a distributed application that is the combination of data domains and transition functions of individual processes. The input and output data is a k-dimensional structure consisting of data of individual processes as shown in equation 7:

$$S_{distr} \equiv \{S_{proc\ 1} \times S_{proc\ 2} \times \ldots \times S_{proc\ k}\}^2 \equiv S_{proc}^{2l} \times S_{proc}^{2m} \equiv Dom^{2k} \tag{7}$$

Relating equations 1 and 7 to Fig. 2 we have:

$$G_{distr} \subseteq S_{proc}^{2l} \times S_{proc}^{2m} \equiv PM \times DM \tag{8}$$

The data model for a distributed application is presented in Fig. 3. The model is the cartesian product of the model for sequential programming with the distinction of input, output and internal processes.

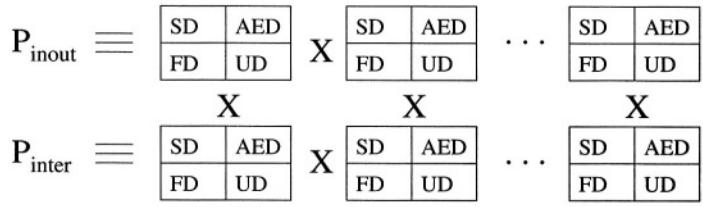


Fig. 3. The structure of a distributed system in the context of input, output and internal processes

3 Exception Handling Model for Deterministic Multiprocessor Environments D^2EH/M

In the previous section we described the Computation model and the Data management of a distributed application, but we omitted the impact of a distributed environment. In this section we analyze how a distributed, deterministic environment influences the execution of an application. An application running in a distributed environment could be analyzed in the four aspects as presented in Sect. 2. The analysis presented in this section focuses mainly on Task management and Communication.

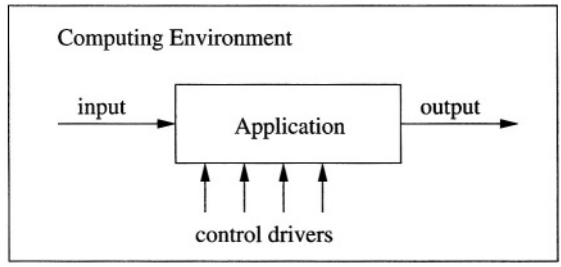


Fig. 4. The influence of environment on application

A distributed environment is a lower layer for an application and it needs to support adequate functionalities. The impact of a distributed environment on an application is shown in Fig. 4. An application running in a distributed environment is modeled as:

$$G(Appl_{distr}, Env_{distr}) \subseteq PM \times DM \times TM \times Comm \tag{9}$$

that represents the complete set of factors that influence the execution as shown in Fig. 2. Usually we assume that the execution environment is correct (*CE*), i.e. it behaves conformably to its specification. We denote by *Env* the set of all possible environments and by *CE* the set of all correct environments. If an

environment does not meet its specification, it is described as incorrect (*IcE*) as shown in the equation:

$$Env \equiv IcE \cup CE \tag{10}$$

However, apart from the correctness of the environment we also define the suitability for a given application. The result of the execution of an application in a suitable environment (*SE*) is the same as in a monoprocessor system. SE may be described as compatible for an application. The environment is further described concerning the suitability for the whole application (*ASE*) and for individual processes (*PSE*). We note:

$$SE \equiv \{e \in CE : G(Appl_{distr}, e) \equiv G(Appl_{distr}, seq)\} \tag{11}$$

$$UsE \equiv CE - SE \tag{12}$$

An unsuitable environment (*UsE*) for an application is the one that causes the application to give different results than for a monoprocessor system, however the environment behaves conformably to its specification.

As a concrete example, take an application that can run on P_1 processors, however it fails to run on $P_2 > P_1$ processors. The first environment is suitable for the application and the second one is not.

Finally the environment domain is divided into:

$$Env_{distr} \equiv IcE \cup SE \cup UsE \equiv TM \times Comm \tag{13}$$

We may define a set of environments adequate for a given application. An application that fails for some data in any environment is considered erronous.

Then we get the relation of input data, output data and the environment.

$$G_{distrenv} \subseteq S^k \times S^k \times Env_{distr} \tag{14}$$

4 Exception Handling Model for Nondeterministic Multiprocessor Environments *NDEH/M*

In the previous section we described the EH model for deterministic multiprocessor systems. Notice that now most distributed systems are based on the TCP/IP protocol that is a non-deterministic one. In this section we analyze the influence of indeterminism on the transition from input to output.

The model is corrected by the indeterminism factor ϵ (epsilon) that is the result of applying the indeterminism function (IF) to a running application in a given environment. The IF can change the result of processes or application according to the control flow (CF) that we define as:

- favorable CF (*FCF*) - the execution of a program P with input data s is the same as in a suitable, deterministic environment,
- unfavorable CF (*UCF*) - the execution of a program P with input data s is different than in a suitable, deterministic environment.

Similarly to the data model the CF of a distributed application consists of CFs of individual processes (noted as CF_{proc}). If the execution of an application is unfavorable the application gives a result different than expected, so in fact it results in a fault. An application is considered as fully correct if any control flow is favorable for it.

$$CF_{distr} \equiv CF^{k}_{proc} \in \{FCF, UCF\} \tag{15}$$

Finally the model of input and output data and a distributed application running in a nondeterministic distributed environment is described as:

$$G_{distr} \subseteq S^{k} \times S^{k} \times Env_{distr} \times CF_{distr} \tag{16}$$

The equation 16 represents a theoretical model that can be easily tuned for practical situations. The cartesian product emphasizes multi-factor problems of distributed applications running in real environments.

5 Implementation of CORBA-Based Application Simulator

We implemented a tool that allows to simulate and test various application models and architecture solutions. The tool is implemented in Java/CORBA [Obj02] as a simulator that allows to test the exception handling fault-tolerant mechanism [AST02]. The architecture of the simulator and its main building blocks are shown in Fig. 5.

A user defines a simulation for a given application, then the tool executes the simulation and gives detailed statistics about program behavior. An application is described by the following parameters:

- class and method names together with executed calls and potential loops
- the amount of computation in methods, the size of input and output data,
- the definition of data repositories of any kind (both input and output),
- the number of: exceptions in methods and errors in data sources,
- the specification of local and remote objects,
- the specification of parallel executions.

The simulation is augmented with fault-tolerant mechanisms that are defined for each method, so each call is executed in a context that specifies what kind of eh is used. The eh can use the following strategies: try-catch block for the whole method or try-catch block for particular operations in a method (i.e. data access, other method call, etc.). Further, the simulator may execute methods with no exception handling blocks.

The result of a simulation is a profile that gives information about errors, exceptions, performance and the result of corrections. The amount of caught and uncaught exceptions is measured to compare exception handling for different program architectures. The statistics show the number of correct and incorrect executions of functions with the percentage of operations finished successfully.

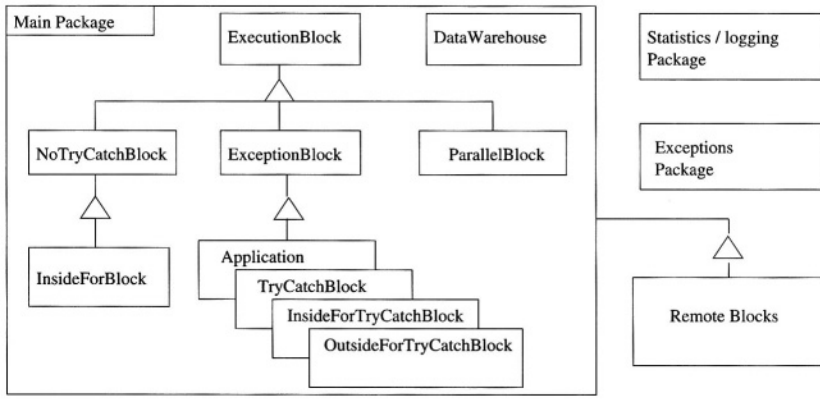


Fig. 5. The architecture of the simulator

We executed simulations to analyze the effect of various exception handling strategies in two different programming models. In each model, an exception was thrown during a function execution with a given probability ranging from 0 to 8 percent.

The first simulated application consists of cooperating objects located in four layers: data base, entity beans, session beans and interface, which could represent a typical J2EE application [SB02]. Two different eh strategies were tested for this application model. We modified the following factors: the number of function bodies with guarded blocks defined and the location of guarded blocks. The first strategy implemented basic fault-tolerance and it set one guarded block for a function body in half of functions. In contrast, the second strategy implemented advanced eh with guarded blocks in nearly all function bodies.

Fig. 6 shows experimental results of the total number of lost function calls in relation to the number of exceptions. We can see that the number of lost function calls differs significantly for the two eh models.

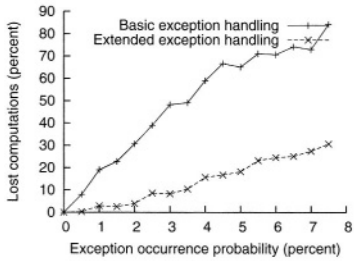


Fig. 6. The number of lost function calls

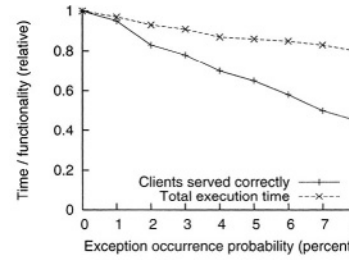


Fig. 7. Execution time and functionality

The second simulated application consists of clients invoking server methods. The server is located on a different machine than the client to present performance issues. The server method calls other methods that access data base and perform some calculations. In this test, no guarded blocks are defined in server methods, so any exception causes a failure of client request.

With this model, we analyzed the dependence between: the probability of exception occurrence, the execution time and the number of correctly served clients. Fig. 7 shows experimental results expressed in relative values from 0 to 1. Note that if an exception occurs during a server method, the method is terminated immediately, so the remaining part is not executed. As a result, the total execution time is shorter, but some clients are not served correctly. The number of correctly served clients decreases faster than the execution time. This is because exceptions are thrown after doing some work in server methods, which occupies system resources, even though the functionality is not supported finally.

6 Conclusions and Future Work

The size and complexity of distributed systems grow steadily and a complete system specification becomes a rare product. We have presented a model that allows to identify different areas of problems in distributed systems and applications. The model extends sequential exception handling with a number of factors that covers both application model and execution environment. The model is the basis for the implementation of a simulator that allows to test various fault-tolerant solutions in distributed systems.

Our future work in the field of exception modeling concentrates on the precise analysis of process state changes during the execution. Further we will conduct simulations and compare them with working programs to proof the accuracy or identify potential discrepancies of experimental results.

Acknowledgments. We would like to thank MsC student Robert Krasiński for the help to implement the software package. This work was supported in part by KBN under grants number 4T11C 00422 and 4T11C 00525. The recent stable version of the simulator is available at author's web page www.eti.pg.gda.pl/~pkacz.

References

[All00] G. B. Alleman. Exception handling in corba environments. Technical report, Niwot Ridge Consulting, 2000.

[AST02] M. van Steen A. S. Tannenbaum. *Distributed Systems Principles and Paradigms.* Prentice Hall, 2002.

[Cri95] F. Cristian. Exception handling and tolerance of software faults. In *Software Fault Tolerance.* John Wiley & Sons Ltd, 1995.

[Jal94] P. Jalote. *Fault Tolerance in Distributed Systems.* Prentice Hall PTR, 1994.

[Obj02] Object Management Group. *CORBA: Core Specification,* 2002.

[SB02] K. Hase S. Bodoff, D. Green. *The J2EE Tutorial.* Addison-Wesley, 2002.

Program Structuring Heuristics for Parallel Systems Based on Multiple Crossbar Switches

Eryk Laskowski

Institute of Computer Science
Polish Academy of Sciences
01-237 Warsaw, Ordona 21, Poland
laskowsk@ipipan.waw.pl

Abstract. A new graph structuring algorithm for look-ahead reconfigurable multi-processor systems based on multiple crossbar switches is presented. It is based on list scheduling and a new iterative clustering heuristics for graph partitioning. Look-ahead dynamic interprocessor connection reconfiguration is a multi-processor architectural model, which has been proposed to eliminate connection reconfiguration time overheads. It consists in preparing link connections in advance in parallel with program execution. An application program is partitioned into sections, which are executed using redundant communication resources, i.e. crossbar switches. Parallel program scheduling in such a kind of environment incorporates graph partitioning problem. The experimental results are presented, which compare the performance of several graph partitioning algorithms for such a kind of environment.

1 Introduction

The main disadvantage of reconfigurable multi-processor systems consists in introducing link connection reconfiguration overheads into the communication execution time. To tackle with this problem, a new program execution model called look-ahead dynamic link reconfiguration [5] has been proposed. Its overall idea relies on preparing link connections in advance in parallel with program execution. A special architectural solution is necessary to provide link connection reconfiguration in advance. In this paper, we present the architecture based on multiple link connection switching devices used alternately for communication execution and look-ahead connection setting.

Our previous works [2,3] show that structuring of parallel program plays a meaningful role in its efficient execution in look-ahead environment. Application program has to be partitioned into sections in such a kind of systems, thus programs structuring consists in task scheduling and graph partitioning. In this article we focus on the partitioning phase and we present a new algorithm for system with many crossbar switches. In [2,3] several kinds of structuring algorithms have been presented for systems with two crossbar switches. The new algorithm, intended for application in multi-crossbar systems, incorporates refinements in heuristics used, which allow to adapt its functioning to system and program parameters.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 314–322, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The paper consists of three parts. The first part describes the look-ahead dynamic link reconfigurable system and parallel program model. In the second part program graph scheduling algorithms are discussed. The last part consists of discussion of experimental results and summary.

2 The Look-Ahead Reconfigurable Multi-processor System

The look-ahead dynamically reconfigurable system, with redundancy of link connection switches, is a system with distributed memory and message passing, Fig. 1. Worker processors $(\mathbf{P_i})$ have sets of communication links connected to the crossbar switches $\mathbf{S_1} \ldots \mathbf{S_X}$ by the Processor Link Set Switch. This switch is controlled by the Global Control Processor (**PS**). The switches are interchangeably used as the active and configured communication resources. An application program is partitioned into sections, which assume fixed direct inter-processor connections. Connections for next sections are prepared while current sections are executed. PS collects messages on the section execution states in worker processors sent via the Control Communication Path and prepares connections for next program sections. In parallel with reconfiguration, synchronization of processors in clusters for next sections is performed using the hardware Synchronization Path. When connections for a section are ready and the synchronization has been reached, PS switches links of processors, which will execute the section, to the proper switch and enables execution of the section. Thus, this method can provide inter-processor connections setting with almost no delay in program execution time.

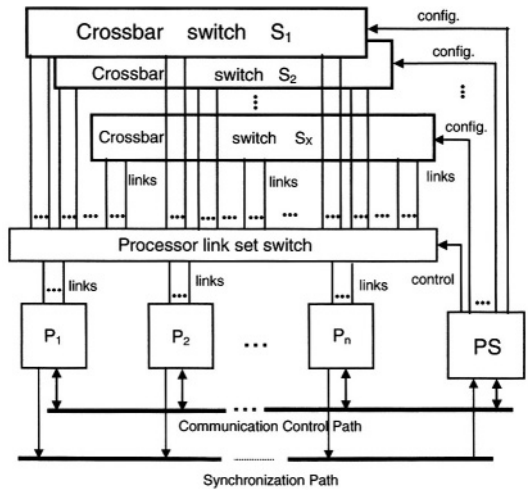


Fig. 1. Look-ahead reconfigurable system with multiple connection switches.

3 Program Structuring Algorithms in the Look-Ahead Configurable Environment

The representation of a program is a weighted Directed Acyclic Graph (DAG) where nodes represent computation tasks and directed edges represent communication (data dependencies) among nodes. Program is executed according to the *macro-dataflow* [4] model. The graph is static and deterministic.

In the paper, a two-phase approach is utilized to tackle the problem of scheduling and graph partitioning in assumed environment [5]. In the first phase, a list scheduling algorithm is applied to obtain a program schedule with reduced number of communications and minimized program execution time. In the second phase, scheduled program graph is partitioning into sections for the look-ahead execution.

In the look-ahead reconfigurable environment the schedule, which determines task execution order, also consists of program partitioning into sections. In presented algorithm, a program with specified schedule is expressed in terms of the Assigned Program Graph (APG, see [3]). APG assumes the synchronous communication model (CSP-like). Modeling in APG of asynchronous, non-blocking communications as in the look-ahead reconfigurable environment, is shown in Fig. 2(a). Processor links are modeled as subgraphs (marked as L_{i1}), parallel to the computation path. Each communication is modeled as an activation path on the sender processor, which is used for sending a message to the link subgraph and as an activation path on the receiver processor, which transmits a message from link to processor.

To enable an easier partitioning analysis we introduce another program graph representation, which is called communication activation graph (CAG). Nodes of this graph correspond to external communication edges of the APG, edges correspond to activation paths between communications in the APG, Fig. 2(b).

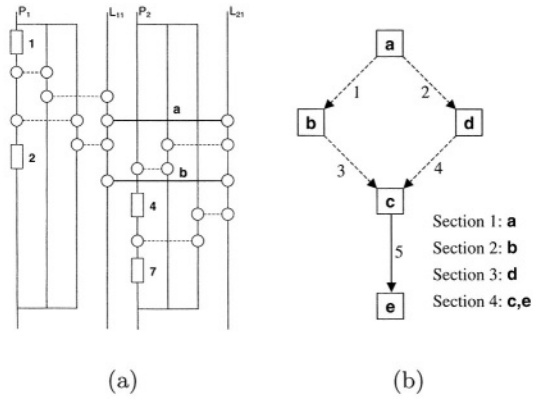


Fig. 2. (a) Modeling of scheduled *macro-dataflow* graph by the APG. (b) Communication Activation Graph partitioned into sections.

Program sections are defined by identification of such subgraphs in the APG or in CAG that the validity conditions hold (see [3] for details). A partition into sections of CAG of an exemplary program graph is shown in Fig. 2(b) (communication activation edges, which do not belong to any section are denoted by dashed lines).

The scheduling algorithm is modified version of ETF (Earliest Task First) strategy, proposed by Hwang et al. [1]. The main difference from original version of ETF is that instead of fixed inter-processor network topology, we investigate system with look-ahead dynamically created connections. We take into account a limited number of links and links contention. Modification of ETF consists in new formulae used for evaluation the earliest starting time (*Ready* procedure [1]). In this procedure link reconfiguration time overheads are minimized by reduction of the number of link reconfigurations.

A second phase of the scheduling procedure is the graph partitioning algorithm. This heuristics finds program graph partitioning into sections and assigns crossbar switch to each section, Fig. 3. The heuristics also finds the minimal number of switches, which allow program execution without reconfiguration time overheads. The algorithm starts with an initial partition into sections, which are assigned to the same crossbar switch and each of it is built of single communication. In each step, a vertex of CAG is selected and then the algorithm tries to include this vertex to a union of existing sections determined by edges of the current vertex. The heuristics tries to find such a union of sections, which doesn't break rules of graph partitioning. The union, which gives the shortest program execution time is selected.

When section clustering doesn't give any execution time improvement, the section of the current vertex is left untouched and crossbar switch is assigned to it. As with section clustering, the choice of the switch depends on program execution time. When algorithm cannot find any crossbar switch for section that allows to create connections with no reconfiguration time overhead, then current

```
B := initial set of section, each section is composed of single communication assigned to x-bar 1
curr_x := 1    {current number of switches used}
finished := false
While not finished
    Repeat until each vertex of CAG is visited and there is no execution time improvements {1}
      v := vertex of CAG which maximizes the selection function and which is not in tabu list
      S := set of sections that contain communications of all predecessors of v
      M := Find_sections_for_clustering(v, S)
      If M ≠ ∅ Then
        B := B - M
        Include to B a new section built of v and communications from sections in M
      Else
        s := section that consists of communication v
        Assign crossbar switch (from 1..curr_x) to section s
        If reconfiguration introduces time overheads Then
          curr_x := curr_x + 1
          Break Repeat
        EndIf
      EndIf
    EndRepeat
    finished := true
EndWhile
```

Fig. 3. The general scheme of the graph partitioning algorithm.

number of switches used (*curr_x* in Fig. 3) is increased by 1 and algorithms is restarted.

The vertices can be visited many times. The algorithm stops when all vertices have been visited and there hasn't been any program execution time improvement in a number of steps. Heuristics manages a list of last visited vertices (*tabu list* in Fig. 3), which is used to prevent algorithm from frequent visiting small subset of all vertices.

The most important part of the heuristics is a vertex selection procedure. Heuristics selects vertex, which maximizes the value of selection function. The following APG and CAG graph parameters are taken into account:
a) the *critical path* CP of APG,
b) the *delay* D of vertex of CAG,
c) the value of *critical point of reconfiguration* Q for the vertex,
d) the dependency on links use between communications.

Critical path of APG is established in the graph partitioned into sections according to the best recent partitioning found. The delay D_v of vertex v is defined as follows:

$$D_v = I_v/(\mathrm{su}_v - \max(eu_{P(v)}))$$

where (as shown in Fig. 4):
$I_v = (s_v - \max(e_{P(v)}))$ – length of reconfiguration interval of vertex v,
$P(v)$ – parents of vertex v,
e_v – finishing time of vertex v,
s_v – starting time of vertex v,
su, *eu* are starting and finishing times, respectively, in APG before graph partitioning (i.e. with reconfiguration time overheads neglected).

The choice of the vertex for visiting depends also on the reconfiguration time overheads, which are measured by applying the *critical point of reconfiguration* heuristics. For every communication v the value of reconfiguration burden C_v is computed:

$$C_v = I_v/R$$

The value Q of *critical point of reconfiguration* function for vertex v is equal to sum of reconfiguration burden of all vertices whose reconfiguration interval is overlapping with reconfiguration interval of v (see Fig. 4):

$$B(t) = \sum C_i : i = 1 \ldots n, s_i > t > \max(e_{P(i)})$$

$$Q_v = \max(B(t) : s_v > t > \max(e_{P(v)}))$$

The communications (vertices of CAG) are classified into three disjoint sets depending on their relationship in processor link use. The first set G_1 contains communications which use the same links as one of their parent vertex (two edges between vertices v_5, v_6; Fig. 4). The second set G_2 contains vertices, which cannot be clustered into single section with their parents because of conflicting link connection requirements. As the third set G_3 heuristics takes vertices not belonging to any of previous sets. During visiting of a vertex, the set G_1 is

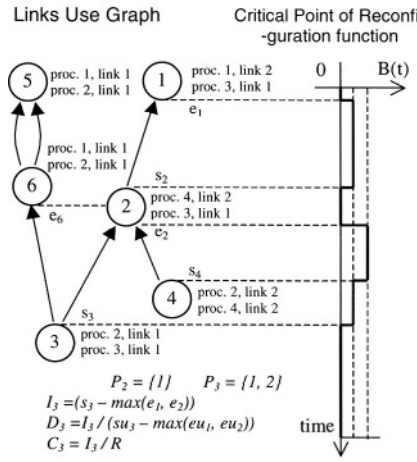


Fig. 4. Evaluation of the delay and critical point of reconfiguration based on the link use graph.

considered as the first, because it is advised to join such communications into single section. The set G_3 is considered as second, the set G_2 as the last.

Using the described APG graph parameters and the dependency on links use between communications, several vertex selection heuristics have been identified (iter0÷iter4). They select a vertex with the biggest value of selection function Z, which is computed as follows:

iter0 – $Z = Q$

iter1 – $Z = c_1Q + c_2D + c_3CP$ /c_1, c_2, c_3: arbitrary constants/,

iter2 – sort vertices according to value CP, D, Q; get the first vertex,

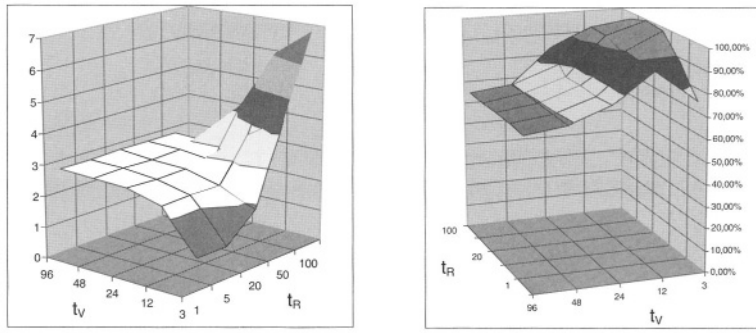
iter3 – $Z = D$

iter4 – $Z = CP$.

The program execution time is estimated by simulated execution of the partitioned graph in a modeled look-ahead reconfigurable system. A APG graph with a valid partition is extended by subgraphs which model the look-ahead reconfiguration control [3,2]. The functioning of the Communication Control Path, Synchronization Path and the Global Control Subsystem PS are modeled as subgraphs executed on virtual additional processors. Weights in the graph nodes correspond to latencies of respective control actions, such as crossbar switch reconfiguration, bus latency, and similar.

4 Experimental Results

The results were obtained for three exemplary programs: **strassen, sparse** (matrix multiplication), **robot** (robot control), executed in the *look-ahead* and in *on-request* system, with the following system parameters: **nb. of processors:** 4, 8, 12, **nb. of proc. links:** 2, 4, **synchronization:** via bus (“bus” in Fig. 6),



(a) (b)

Fig. 5. Speedup against the *on-request* system **(a)** and reduction of the reconfiguration time overhead **(b)** for Strassen algorithm executed in the *look-ahead* environment (12 processors, 4 links, 6 crossbars).

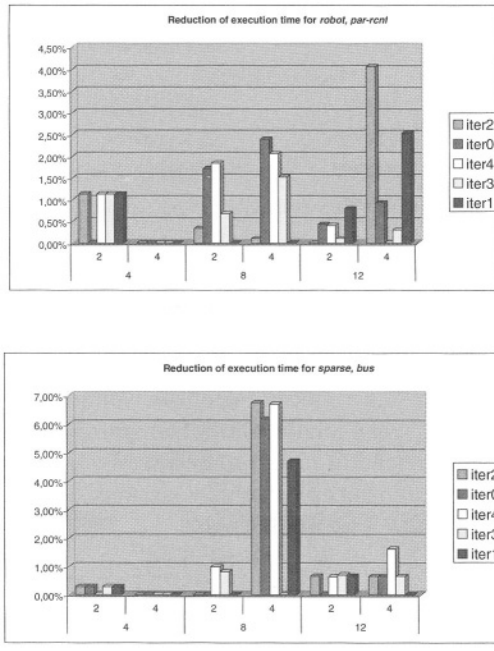


Fig. 6. Comparison of performance of graph partitioning algorithm for different vertex selection heuristics used and for different program graphs.

hardware barrier ("direct"), or with parallel reconfiguration of switches ("par-rcnf"), reconfiguration time of a single connection $\mathbf{t_R}$ in range 2... 200, section activation time $\mathbf{t_V}$ in range 1... 100.

The program execution speedup over the *on-request* environment in function of parameters of reconfiguration control ($\mathbf{t_R}$ and $\mathbf{t_V}$), on the example of the **strassen** graph are shown in Fig. 5(a). Figure 5(b) shows the reduction of the reconfiguration control time overhead when *look-ahead* control is used instead of *on-request.* Multiple crossbar switches used with the look-ahead control strongly reduce reconfiguration time overheads. When reduction is close to 100%, the system behaves for any given program as a system with fully-connected inter-processor network. The larger is the number of links in a processor, the look-ahead method is successfully applicable for wider range of reconfiguration and activation time parameters than with the *on-request* reconfiguration.

The results of experiments with different heuristics of vertex selection in graph partitioning algorithm are shown in Fig. 6. Although the difference between iter0÷iter4 heuristics is small (only the order of selection of vertices is changed), the difference in total execution time of partitioned program was up to 6.5%. It should be noted that the performance of different versions of partitioning algorithm depends strongly on parallel program graph used. The biggest difference was for **sparse** graph, while for **strassen** graph the performance of all methods was similar. For a given graph, the performance depends mainly on the number of processor and processor links.

The analysis of experimental results has shown that methods iter0, iter1, iter2 behaves slightly better than others. These heuristics use *critical point of reconfiguration* function, thus we could deduce that this APG parameter is the most important. The choice of vertex selection method allows to tune partitioning algorithm to program graphs parameters.

5 Conclusions

Several variations of the iterative graph structuring algorithm for the look-ahead reconfigurable multi-processor system has been presented in the paper. They allow to adjust heuristics to program and system parameters, thus a better performance of graph partitioning could be achieved. The presented algorithm gives better results than the one with greedy partitioning heuristics and has the same time complexity, which has been proved by our experiments. The future works will focus on partitioning algorithm improvements, which could lead to further optimizations in section clustering and mapping of resources to communications.

References

1. J-J. Hwang, Y-C. Chow, F. Angers, C-Y. Lee *Scheduling Precedence Graphs in Systems with Interprocessor Communication Times,* SIAM J. Comput., Vol. 18, No. 2, 1989.

2. E. Laskowski *New Program Structuring Heuristics for Multi-Processor Systems with Redundant Communication Resources,* Proc. of PARELEC 2002 Intl. Conference, Warsaw, Poland.
3. E. Laskowski, M. Tudruj, *A Testbed for Parallel Program Execution with Dynamic Look-Ahead Inter-Processor Connections,* Proc. of PPAM '99, Sept. 1999, Kazimierz Dolny, Poland.
4. El-Rewini H., Lewis T. G., Ali H. H. *Task Scheduling in Parallel and Distributed Systems.* Prentice Hall 1994.
5. M. Tudruj, *Look-Ahead Dynamic Reconfiguration of Link Connections in Multi-Processor Architectures,* Parallel Computing '95, Gent, Sept. 1995, pp. 539-546.

Automatic Generation of Optimized Parallel Codes for N-body Simulations*

David E. Singh[1], María J. Martín[2], and Francisco F. Rivera[1]

[1] Univ. Santiago de Compostela, Dept. of Electronics and Computer Science, SPAIN
{david,fran}@dec.usc.es
[2] Univ. A. Coruña, Dept. of Electronics and Systems, SPAIN
mariam@udc.es

Abstract. In this work a novel strategy for solving the n-body problem in parallel is presented. Our method, named slice classification, not only extracts the maximum available parallelism, but also it increases data locality. Results on a SGI O2000 validate the efficiency of our proposal as compared with other well-know methods.

1 Introduction

Nowadays, n-body codes are spreadly used for the simulation of a large number of physical processes. These codes can be structured using different approaches, although the most popular and efficient strategies are based on the use of irregular structures. An typical example of this kind of codes is shown in Figure 1, where arrays $y[j]$ and $x[j]$ are used to point the bodies. In this code, the spatial coordinates of the bodies are stored in the *position* array. For each loop iteration the interaction force is computed and accumulated in f. The i^{th} entry of f collects the net interaction force for the i^{th} body. When every interaction force is computed, speeds and positions of the bodies are updated, and a new time step of the simulation starts.

There are different strategies for the automatic parallelization of this kind of codes. The array expansion [1] is one of the most popular approaches. Its main drawback is the memory and communication overheads it introduces. With the aim of reducing these overheads two new proposals were developed [2]: selective privatization and sparse reductions with privatization in hash tables. Recently new approaches has arisen in order to increase data locality, being the main proposals LOCAL-WRITE [3] and DWA-LIP [4]. Both maximize locality in the write accesses by means of an inspector that specifies the iterations assigned to each processor. LOCAL-WRITE does not need synchronizations, but in contrast, it introduces overheads due to the replication of computations. DWA-LIP eliminates replication but it requires synchronizations.

In this work a new parallel approach for CC-NUMA shared memory machines is proposed. Our strategy not only maximizes locality in the accesses to array f, but also in the read accesses to arrays x and y.

* This work was supported by the MCYT under project TIC2001-3694 and European Community Access to Research Infrastructure contract No HPRI-CT-1999-00026.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 323–330, 2004.
© Springer-Verlag Berlin Heidelberg 2004

```
DO  j = 1, N_interact
    force = compute_force(position[x[j]], position[y[j]]);
    f[x[j]] += force
    f[y[j]] -= force
END DO
position = update_position(f)
```

Fig. 1. N-body kernel

2 Automatic Parallelization Scheme

Our proposal is based on the inspector-executor paradigm. We deal with three main topics: first, we present a new parallel executor that allows the concurrent execution of irregular loops without conflicts in the accesses to data. Second, we present a new data distribution scheme in which data locality is highly improved by means of a reordering of the indirection arrays. Finally, an adaptive scheduler is introduced to achieve near-optimal load balance. The last two points comprised our proposed inspector.

2.1 Executor

The owner-computes rule in the accesses to f is applied using a uniform block distribution. For a given iteration, j, the couple of values $owner_x$ and $owner_y$ denote the indexes of the processors that owns $f[x[j]]$ and $f[y[j]]$, respectively. We use iteration-level parallelism. Therefore, $f[x[j]]$ and $f[y[j]]$ are always computed by the same processor. The iterations assigned to a given processor p can be classified into three categories:

Exclusive. Both accesses are performed in the same block of f. That is to say $owner_x = owner_y = p$.

Shared_x. These iterations perform a local access to $f[x[j]]$, and a non-local access to $f[y[j]]$. In other words, $owner_x = p$ and $owner_y \neq p$.

Shared_y. These iterations fulfill $owner_x \neq p$ and $owner_y = p$.

The indirection arrays x and y are reordered and new indirection arrays (x_{out} and y_{out}) are generated. In these new indirection arrays the iterations executed by processor 1 ordered by execution order are stored first, then the iterations executed by processor 2, and so on. Using this reordering data locality in the accesses to arrays x and y is improved. To illustrate their use, consider the example of Figure 2(a). It represents the values of the indirection arrays for a problem with $N_{interact} = 10$. The resulting reordered array for a distribution with 2 processors is shown in Figure 2(b). As f has six entries, processor 1 and 2 are assigned, respectively, to the intervals of entries [1,3] and [4,6] of f. The complete set of iterations is divided into subsets associated to each processor. For instance, iterations 1, 2 and 7 of the reordered arrays are exclusive and the rest of them are classified as shared.

Figure 3 shows the structure of the proposed executor. The parallel execution of the irregular code is divided into 4 stages. First, the exclusive stage computes

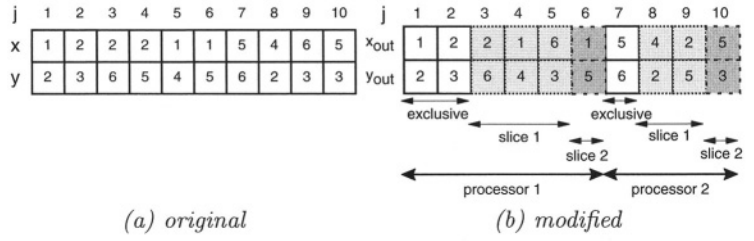


Fig. 2. Examples of indirection access patterns

the exclusive iterations. Next, the iterations associated with the *shared* stages are executed. According to their definitions, for each iteration in this stage, one of the accesses to f is local and other is non-local. We name them, respectively, *local* and *remote* access. Note that remote accesses produce risk of memory conflicts. To deal with this problem, we introduce an intermediate buffer, called g, that is used to store partial results. The size of this buffer is the same as f. When the iterations of a *shared* stage are executed, both f and g arrays are updated according to the following rule:

Rule 1: f is used to accumulate the result of local accesses, and g is used to store partial contributions associated to remote accesses. For both arrays, the contributions are stored according to the value of the local index.

In the example of Figure 2(b), iterations 3 and 4 are executed by processor in the $shared_x$. For iteration 3, the results are stored in $f[2]$ and $g[2]$, and for iteration 4, the computed force is stored in $f[1]$ and $g[1]$. In the case of the $shared_y$ stage the same procedure is applied. In the example, processors 1 executes iteration 5 updating $f[3]$ and writing $g[3]$. Note that each entry of g can store only one value. Therefore, this procedure can not continue when an iteration tries to write an occupied entry of g. To solve this problem we introduce the concept of *slice.* A *slice* is defined as a set of *shared* iterations that do not cause memory conflicts in the access to g. According to this definition, the following property is verified:

Property 1: The local accesses of two iterations belonging to the same slice are different.

For the example of Figure 2(b), iterations 3,4,5,8 and 9 are in the same slice, whereas iterations 6 and 10 belong to other slice. Note that iteration 6 performs the same local access as iteration 4.

In order to organize the accesses to data three auxiliary arrays are generated. Array ρ_{excl} stores the number of iterations that each processor executes in the *exclusive* stage. Arrays ρ_c^x and ρ_c^y are three-dimensional arrays where the first dimension specifies the owner processors of the local accesses, the second dimension the owner of the remote accesses, and the third one the index of the associated slice. For instance, array $\rho_c^x[4,3,5]$ points to the first element owned by processor 4 with a remote access to an entry of f owned by processor 3, and it is executed in the 5^{th} slice of the $shared_x$ stage. In the same way, $\rho_c^y[4,3,5]$ points to the first element owned by processor 4 with a remote access to an en-

```
    DOALL p = 1, N_p                                          %Exclusive stage
L1      DO j = ρ_c^x[p,1,1] − ρ_excl[p], ρ_c^x[p,1,1] − 1
            force = compute_force(position[x_out[j]], position[y_out[j]])
            f[x_out[j]] += force
            f[y_out[j]] −= force
        END DO
    END DOALL
    DO slice = 1, N_slices
        DOALL p = 1, N_p
L2          DO j = ρ_c^x[p,1,slice], ρ_c^y[p,1,slice] − 1        %Shared_x stage
                force = compute_force(position[x_out[j]], position[y_out[j]])
                f[x_out[j]] += force
                g[x_out[j]] = force
            END DO
L3          DO j = ρ_c^y[p,1,slice], ρ_c^x[p,1,slice+1] − 1n     %Shared_y stage
                force = compute_force(position[x_out[j]], position[y_out[j]])
                g[y_out[j]] = force
                f[y_out[j]] −= force
            END DO
        END DOALL ; BARRIER                                       %Synchronization
        DOALL p = 1, N_p                                          %Gathering stage
            DO k = 1, Np
L4              DO j = ρ_c^x[k,p,slice], ρ_c^x[k,p+1,slice] − 1
                    f[y_out[j]] −= g[x_out[j]]
                END DO
L5              DO j = ρ_c^y[k,p,slice], ρ_c^y[k,p+1,slice] − 1
                    f[x_out[j]] += g[y_out[j]]
                END DO
            END DO
        END DOALL ; BARRIER                                       %Synchronization
          END DO
```

Fig. 3. Executor algorithm

try of f owned by processor 3, and it is executed in the 5^{th} slice of the $shared_y$ stage. Figure 4(a) shows these pointer arrays for the example of Figure 2(b). In this figure, the second dimension is ignored because only two processors are considered.

Loop `L1` of Figure 3 illustrates the interval of entries of the *exclusive* stages for a given processor p. Since there are no memory conflicts, the parallel loop does not need synchronizations. Loops `L2` and `L3` of Figure 3 show the intervals of shared entries for a given processor p and slice s. According to Rule 1, all the iterations of stages $shared_x$ and $shared_y$ that belong to the same slice s can be executed without synchronizations.

Finally, f is updated in the *gathering* stage. Each processor checks the other processors taking the required values from g for updating its local entries of f. For a given processor k, these values are given by intervals shown in Figure 3 in loops `L4` and `L5` . Once f is updated, g can be reused for the next slice. Figure 4(b)

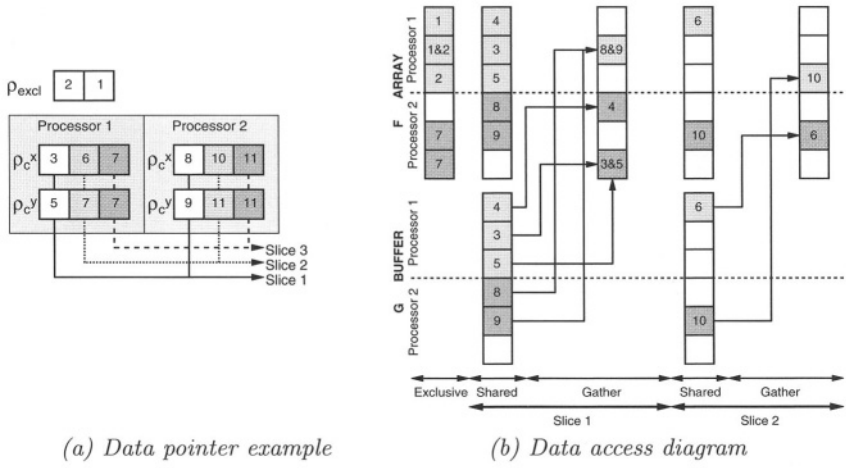


Fig. 4. Pointer and execution examples

shows the access diagram for the example of Figure 2. In each stage, the acceded entries are colored. Inside each one, the iteration that performs the access is written. Note that write operations are always local accesses, both in f and g, and f is always private.

2.2 Inspector

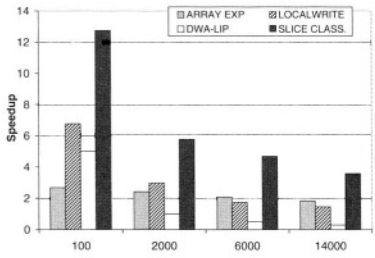
The inspector is divided into two independent modules. The first one reorders the indirections arrays x and y (arrays x_{out} and y_{out}) and generates the data layout described in the previous section (arrays ρ_{excl}, ρ_c^x, ρ_c^y). The second one, the scheduler, assigns iterations to each processor.

Due to the lack of space, we do not depict their structure. A full description of both modules can be found in [5].

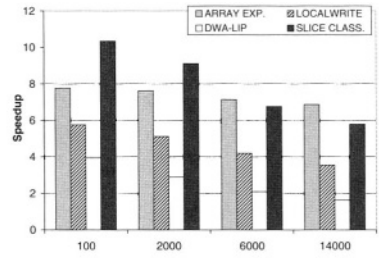
3 Results

The Lennard-Jones Molecular Dynamics code [6] is used as benchmark. This code simulates the interactions of a set of particles under a Lennard-Jones potential and allows different kinds of decompositions. Specifically, as source code, the sequential version of the code named LJA was selected. Parallel versions of this kernel were generated using our proposal, named slice classification (SC), and three representative and efficient strategies: the array expansion, the LOCAL-WRITE and the DWA-LIP. For the array expansion technique, a block distribution of the iteration space was used. For all the other strategies, an uniform block distribution of the array f was considered.

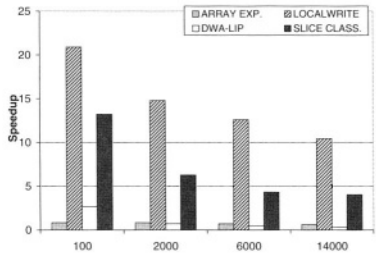
The LJA benchmark allows us to specify the number of bodies that are simulated and their average connectivity, that is, the average number of neighbors each body interacts with. In this work we have considered three different



test1



test2



test3

Problem	N_{body}	N_x	Connectivity
*test*1	$97K$	$1008K$	10.4
*test*2	$32K$	$1202K$	37.6
*test*3	$97K$	$297K$	3.1

Access patterns specifications

Fig. 5. Memory access patterns specifications and speedups for 32 processors

problems, trying to cover the main real situations. Their features are summarized in Figure 5. The first one, named *test1,* corresponds to an usual problem with an average connectivity around 10, and a number of bodies of $97K$. The other two cases correspond to more extreme conditions. The problem *test2* is a high-connectivity problem with a small number of bodies and *test3* represents a low-connectivity context.

An important feature of each example is the locality of the access pattern. This parameter measures how close in memory are the acceded entries in each iteration. LJA presents high locality at the beginning of the execution, however, it decreases as the number of iterations increases. For each test problem, the patterns were stored for iterations 100 (high locality), 2000, 6000 and 14000 (low locality).

Figure 5 shows the speedups obtained with 32 processors on an SGI Origin 2000. Our proposal obtains the best results for all the considered iterations of *test1* and also for *test2* when data locality is high.

DWA-LIP presents good performance for sparse problems with high data locality. The main drawback of this technique is that the shared iterations are processed with a small degree of parallelism.

The array expansion strategy presents, as main drawback, the communications overhead of the reduction operation. There are two main factors that affects the efficiency of this proposal. On one hand, the bigger size of f array, the smaller performance of this approach. On the other hand, when the number of iterations, N_x, decreases, the communication overhead approaches to the cost

Table 1. Iteration threshold for 32 processors

Proc.	*test1*				*test2*				*test3*			
	100	2K	6K	14K	100	2K	6K	14K	100	2K	6K	14K
8	5	8	9	10	4	8	7	7	2	4	6	6
16	8	13	9	10	4	8	8	13	3	4	5	6
32	6	13	10	11	5	6	8	9	3	4	5	7

of the computations, decreasing the performance of the parallel code. Therefore, this proposal obtains the best results for *test2* when data locality is poor. One important advantage of this technique is the excellent load balance achieved due to the block distribution of iterations. However, this strategy does not exploit data locality in the write accesses to f.

With the LOCAL-WRITE strategy, data locality in the write accesses is exploited. However, it has the penalty of the replication of the computations. This replication is more important when data locality decreases, and has a great impact in the parallel performance. For *test3,* LOCAL-WRITE obtains the best results because the connectivity is low.

An important factor of a parallel proposal is its scalability. A source of overhead of our proposal is the number of slices since they introduce synchronizations. The number of slices depends on the structure of the access pattern, and it is not strongly influenced by the number of processors. For instance, for *test1,* the number of slices for 8, 32 and 128 processors are 8, 10 and 12 respectively. This is not the case of the array expansion technique, in this technique each processor always communicates the whole array f. With respect to DWA-LIP and LOCAL-WRITE, the performance decreases as the number of processors increases. For the first one, the execution of the code losses parallelism. For instance, in *test2* for iteration 6000, the percentage of iterations that are executed with less than half of available parallelism increases from 62% with 8 processors to 86% for 32 processors. For the LOCAL-WRITE strategy the computation replication depends strongly on the number of processors. As an example, for the same test, with 8 processors each processor computes 170% more iterations that the computed by the array expansion technique, whereas with 32 processors this percentage increases to 190%.

In order to complete the evaluation of our proposal, the overhead of the inspector have been considered. Table 1 shows the *iteration threshold* for the SC strategy for 32 processors. The *iteration threshold* is defined as the number of iterations that the executor needs to calculate to compensate the overhead of the inspector. That is to say, if the inspector is reused at least as many times as the iteration threshold, then our inspector-executor approach is faster than the sequential version of the code. The overhead of the inspector increases with the number of processors, but due to the high scalability of our proposal, the iteration threshold remains constant and even decreases as the number of processors increases.

4 Conclusions

In this work we present a new parallel strategy for solving irregular n-body codes on CC-NUMA shared memory architectures. Our proposal focus on obtaining the maximum available parallelism, a good exploitation of the memory hierarchy and a correct load balance.

We introduce the concept of slices to compute in parallel, without replication nor remote accesses, iterations that are shared between two different processors. We use an intermediate buffer to store shared accesses in a local way. Good scheduling politics are introduced to minimize synchronizations. With the proposed scheduler, the memory and synchronization overheads do not strongly depend on the number of processors. As this scheduling is performed in the inspector stage, it can be computed in the first iteration of the temporal loop and it can be reused if the access pattern does not change significantly.

Our proposal also exploits data locality in the access to data. As the owner-computes rule is applied, the access locality of f is maximized. Additionally, the indirections arrays are reordered, obtaining high locality in the read accesses. We introduce a pointer structure to reduce the memory overhead of the accesses to the reordered indirections.

Finally, the scheduler provides a good load balance, that can be easily adapted to particular situations. For instance, it can take into account the computational power of each processor or the conditions of the communication network.

References

1. William Blume et al. Parallel programming with polaris. *Computer,* 29(12):78, 1996.
2. Hao Yu and Lawrence Rauchwerger. Adaptive reduction parallelization techniques. In *International Conference on Supercomputing,* 2000.
3. H. Han and C-W. Tseng. Efficient Compiler and Run-Time Support for Parallel Irregular Reductions. *Parallel Computing,* 26(13-14):1861–1887, 2000.
4. Eladio Gutiérrez, Oscar Plata, and Emilio L. Zapata. Improving parallel irregular reductions using partial array expansion. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing,* pages 38–38, 2001.
5. David E. Singh, María J. Martín, and Francisco F. Rivera. Automatic generation of optimized parallel code for n-body simulations. Technical report, Univ. Santiago de Compostela, http://www.ac.usc.es/ david/reports/TR43-0301.ps.gz, 2003.
6. S. J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J Computational Physics,* 117:1–19, March 1995.

Monitoring Threaded Application with Thread-Enabled OMIS Monitor*

Bartosz Baliś [1], Marian Bubak[1,2], Włodzimierz Funika[1], Roland Wismüller[3,4], and Grzegorz Kaplita[1]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
{balis, bubak, funika}@agh.edu.pl, orkid@student.uci.agh.edu.pl
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM – Technische Universität München, D-80290 München, Germany
wismuell@in.tum.de
[4] Institute for Software Sciences, University of Vienna, A-1090, Wien Austria
Phone: (+48 12) 617 39 64, Fax: (+48 12) 633 80 54, Phone: (+49 89) 289-28243

Abstract. This paper describes an infrastructure for threaded applications monitoring – TOM (Thread-enabled OMIS Monitor). TOM provides tools supporting application development, such as debuggers or performance analyzers, with needed monitoring services. We describe problems specific to monitoring multithreaded applications. Then we overview the architecture of TOM and provide some design details. We focus on Application Monitors which are the key concept in TOM, being additional monitoring threads for each process of the monitored applications. The Application Monitors enable efficient and portable solutions to the most important problems related to monitoring multithreaded applications. Finally, we describe the evaluation of the monitoring overhead on a real application example.

Keywords: Multithreading, monitoring, debugging, performance analysis, parallel tools

1 Introduction

Application monitoring is needed for various tools supporting development of applications. For example, debuggers need such a functionality as breakpoint insertion, process suspension, reading and writing process' memory, etc. Another example are performance analyzers which need information related to events inside an executing application, such as calls to communication or synchronization subroutines, to obtain delays due to synchronization, volume of data transferred, etc.

Multithreading is an efficient solution for parallel processing. However, these applications are subject to specific problems such as race conditions due to accesses to shared memory. Also, monitoring threaded applications is more challenging than in case of multiprocess ones.

* This work has been carried out within the Polish-German collaboration and is supported, in part, by KBN under grant 4 T11C 026 22, and, in part concerning SGI, under grant 6 T11 0052

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 331–337, 2004.
© Springer-Verlag Berlin Heidelberg 2004

One of the most popular thread standard today is POSIX threads, called pthreads [10]. Pthreads define a standardized interface and behaviour for thread creation and management subroutines, as well as related data types. Almost every operating system (OS) provides a pthreads library, although the underlying implementation is strongly system-dependent. This makes it hard to develop tools that support the debugging of pthreads applications – there is no standard interface for accessing pthreads internals, and standard system mechanisms such as ptrace or procfs [12] are often not pthreads-enabled.

In this paper, we describe the TOM monitoring infrastructure for multithreaded applications, which supports the development of portable tools, such as debuggers or performance analyzers. The key concept in TOM are Application Monitors which are additional monitoring threads in the monitored application. The Application Monitors concept allows for efficient and portable solutions to the most important problems of thread monitoring. We describe the current implementation of TOM with special focus on Application Monitors and provide an evaluation of effcienyc of Application Monitors on an example of a real application.

2 Problems with Monitoring Threads

Below we shortly summarize the most important problems related to the monitoring of multithreaded programs.

- **Asynchronous control.** We wish to deal with threads asynchronously like with processes, for example, execute step-by-step only one thread of a process while others are still running. Unfortunately, OS interfaces used for this purpose, such as ptrace or procfs, usually do not support threads. This means that manipulating one of the threads with ptrace would require stopping the whole process.
- **Transparency of instrumentation.** Threads share not only data but also code address space. This introduces transparency problems when the monitoring system instruments a thread's code. A breakpoint set in one thread is immediately visible in all other ones while we are going to apply the breakpoint to one thread only.
- **Portability.** Though the common pthreads interface is widely agreed, the underlying implementations of libraries are very different from system to system or even between two versions of the same system (such as in IRIX 6.4 and IRIX 6.5). In addition, there is no standard debug-interface for pthreads. Some vendors do provide a pthreads debug library, but it is not portable and sometimes not even available to the public. As a result, in some cases only the vendor tools are thread-aware, as it is the case for IRIX.

3 Architecture of TOM – Thread-Enabled OMIS Monitor

The architecture of TOM is shown in Fig. 1. TOM is an autonomous infrastructure composed of three types of components: Service Managers (SM), Local

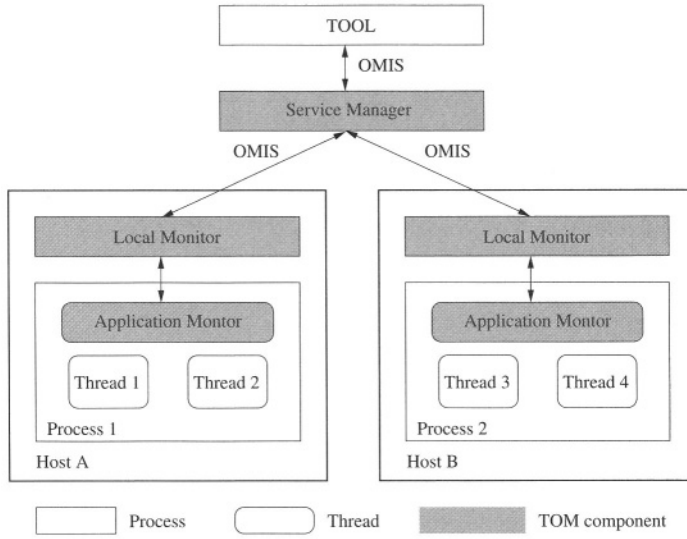


Fig. 1. Thread enabled OMIS Monitor

Monitors (LM), and Application Monitors (AM). Tools connect to TOM via SMs, send monitoring requests and receive replies. The communication interface is based on the OMIS specification (On-line Monitoring Interface Specification) [11,2]. On each local unit of the system, which tightly shares a common memory, one LM is supposed to reside. For example, on a cluster there is a single SM and one LM per host, while on a multiprocessor shared-memory machine, such as SGI Origin, there may be one SM, and one LM per each SMP node. Finally, there is one AM thread per each process of the application. The monitoring requests are executed by either LMs or AMs, while the SM's task is to distribute the requests to appropriate LMs and collect the corresponding replies.

In the rest of the paper, we focus on the Application Monitors, which are the key concept in TOM enabling efficient and portable monitoring of threads.

4 Application Monitors

4.1 General Concept

The Application Monitors (AMs) are designed for two main objectives: portability and efficiency of monitoring. An AM is an additional thread in each process of the application. The main benefit is that, as an AM shares address space with the application, it can perform some actions directly, without OS mechanisms involved.

An LM sends asynchronous requests to an AM and waits for responses. In this way, we can read the address space (e.g., variable values), set breakpoints or

even insert a general instrumentation code dynamically. The asynchronous control of threads is addressed owing to the communication pattern: asynchronous queries/replies as opposed to synchronous mechanisms such as ptrace. At the same time we also benefit from a higher portability. With AMs we can also easily solve the single code image problem. This can be done in such a way that when an instrumentation point is hit, the AM checks if it was hit in the appropriate thread, and only in this case the instrumentation code will be executed.

4.2 Monitoring Solutions Using Application Monitors

Below we describe some example monitoring scenarios which pose problems in traditional approaches. For each problem, we describe how it can be resolved using Application Monitors.

- **Problem: suspend a single thread.** Standard system mechanisms for suspending processes such as ptrace or the proc fs usually are not thread-aware, therefore they allow only for suspending an entire process. In TOM, this scenario is realized with an asynchronous request "suspend thread t" sent to an AM. The AM next sends a signal to the target thread to make it sleep on a condition wait in the signal handler. This problem shows how *asynchronous control* is achieved with AMs.
- **Problem: read process' memory.** To read a process' memory, normally ptrace or procfs are used. These, however, usually require to stop the whole process before any operation can be performed. Moreover, interaction with procfs is not portable across OS's. With AMs, again an asynchronous request "get n-byte valu at address X" is sent to the AM which has a direct access to the process' address space. This problem shows how AMs enable *portability.*
- **Problem: breakpoint is set in a thread which shares code with other threads.** This is a transparency problem. Though the breakpoint is set only in one thread, other threads which execute the same code will hit it. Normally, this is reported to the (external) debugger which, after having checked that the breakpoint was hit by an incorrect thread, continues the execution again. This, however, is a high-overhead operation. With AMs, the filtering of improper hits can be done directly in the local breakpoint handler. Since this does not involve a process switch but only an additional control transfer, the overhead is minimal. This problem illustrates how AMs help achieve *transparency of instrumentation* and *efficient monitoring.*
- **Problem: dynamically instrument code at specified address.** Normally this is very hard to achieve. Only very few systems exist that enable this, such as DynInst [3]. DynInst is a very powerful approach, which allows to insert (nearly) arbitrarily complex code at run-time. AMs also enable dynamic instrumentation of code. Though they are more restricted, since only the jump to the handler function is inserted dynamically while all the other required code is included into the application by linking it with the AM library, they are much more lightweight and portable.

5 Evaluation

We have performed a test to test the overhead of monitoring using tradidional approaches and TOM-based approach. The test application was the SPLASH2 [13] FFT bechmark based on the PARMACS macros [9] implemented on top of pthreads. We have instrumented one of the loops in the algorithm, so that for the data size of 65536 complex data points we have obtained 128 events generated by the instrumentation. The test platform was SGI Origin 2800 running IRIX 6.5. We have measured total time needed to run the program without the instrumentation and for three different instrumentation techniques:

1. TRAP instruction inserted at instrumentation point, TRAP signal handled by an external process. This is the usual method used, e.g., by debuggers to insert breakpoints. Obviously this method involves a process switch, and a signal delivery.
2. TRAP instruction inserted at instrumentation point, TRAP signal handled in the same process. Unlike the previous case, this one does not need an additional process switch.
3. JUMP instruction inserted at instrumentation point. This is the most efficient method, since no process switch or signal delivery, only control transfer is involved.

Addidionally, we have performed the test for two latter cases with an extra system call in the instrumentation handler. This is usual in a real situation, since a system call is needed, e.g., to obtain a time stamp.

Table 1. Evaluation of monitoring overhead

	Normal	Ext. TRAP	Int. TRAP	JUMP	Int. TRAP + syscall	JUMP + syscall
Mean	6111	1013396	8884	6362	9504	6695
Stdev	244	547624	362	214	270	375

Tab. 1 shows the results (time in microseconds). The table shows clearly that with application monitors and JUMP instruction, the instrumentation overhead is within the standard deviation of the measurement, even when there is an additional system call. TRAP instruction with signal handled within the same process still provides good results. The overhead for external TRAP, however, is huge and unnaceptable.

6 Related Work

Until now, thread-enabled tools for parallel programming are still not as well supported as those for multiprocess applications. Most of existing tools are

debuggers, for example, Wildebeest, TotalView, kdb, NodePrism and LPdbx. Wildebeest [1] is an example of a debugger based on gdb, which supports both kernel and user threads. However, it is strictly limited to HP-UX platforms and implements only a synchronous thread control. TotalView [6] is a commercial debugger which supports a variety of platforms and offers a rich set of debugging capabilities. It is well suited for multithreaded applications and even provides support for applications developed in OpenMP. However, it does not allow for asynchronous thread control unless this feature is supported by the operating system. Kdb [4] was designed to overcome the limitations of other debuggers, specifically for handling user-level threads and for controlling each target thread independently. It does not support pthreads, though.

There are some efforts to address performance analysis of multithreaded applications. One example is the `Tmon` tool [7], which is a monitoring system combined with a visualization module used to present *waiting graphs* for multithreaded applications. An interesting approach is the thread-enabled Paradyn tool [14]. Paradyn is able to associate performance data with individual threads and relies on dynamic instrumentation to lower the instrumentation cost and overall monitoring intrusiveness.

7 Conclusion and Future Work

We have described our experience in design and implementation of a portable and efficient monitoring system which supports multithreaded applications. The key concept of our solution are Application Monitors which are additional threads for each process of the application. The presented concept is generic – it is designed to support a variety of tools, for example, debuggers and performance analyzers. We have shown a case study of fast breakpoints based on the concept of Application Monitors.

Currently, we have implemented the first prototype of TOM which contains the infrastructure for Application Monitors with basic monitoring services, and a simple external monitoring infrastructure. The tasks for the future are as follows. First, an existing OMIS-compliant monitoring system for clusters – the OCM [2] will be extended with threads support so that we can benefit from the full monitoring functionality offered by OMIS. Second, the Application Monitors will be extended by additional monitoring functionality. Finally, the thread-enabled OCM will be integrated with Application Monitors to obtain a fully functional OMIS-based thread-enabled monitoring system – TOM.

References

1. S. S. Adayapalam. In Search of Yeti: Footprint Analysis with Wildebeest. In Mireille Ducassé, editor, *Proceedings of AADEBUG 2000, Fourth International Workshop on Automated Debugging,* Munich, Germany, August 2000.
2. M. Bubak, W. Funika, B. Baliś, R. Wismüller. On-line OCM-based Tool Support for Parallel Applications. In *Annual Review of Scalable Computing,* vol. 3, pp. 32–62. World Scientific Publishing and Singapore University Press, Singapore, 2001.

3. B. Buck and J. Hollingsworth. An API for Runtime Code Patching. *The International Journal of High Performance Computing Applications,* 14(4):317–329, Winter 2000.
4. P. A. Buhr, M. Karsten, and J. Shih. KDB: A Multi-threaded Debugger for Multithreaded Applications. In *Proc. SPDT'96: SIGMETRICS Symposium on Par. and Distrib. Tools,* pp. 80–89, Philadelphia, Pennsylvania, USA, May 1996. ACM Press.
5. J. Cargille and B. P. Miller. Binary Wrapping: A Technique for Instrumenting Object Code. *ACM SIGPLAN Notices,* 27(6): 17–18, June 1992.
6. TotalView Multiprocess Debugger. WWW-Site of Etnus Inc., Framingham, MA, USA, 1999. `http://www.etnus.com/products/totalview/index.html`.
7. M. Ji, E. W. Felten, and K. Li. Performance Measurements for Multithreaded Programs. In *Measurement and Modeling of Computer Systems,* pp. 161–170, 1998.
8. P. B. Kessler. Fast Breakpoints. Design and Implementation. *ACM SIGPLAN Notices,* 25(6):78–84, June 1990.
9. The parmacs macros homepage. http://www.netlib.org/parmacs/
10. Portable Operating System Interface: The Pthreads standard (POSIX 1003. 1c).
11. Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A. OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series, 1997. `http://wwwbode.in.tum.de/~omis`
12. J. B. Rosenberg. *How Debuggers Work: Algorithms, Data Structures, and Architecture.* John Wiley & Sons, 1996.
13. The SPLASH benchmarks homepage. `http://www-flash.stanford.edu/apps/SPLASH`
14. Z. Xu, B. Miller, O. Naim. Dynamic Instrumentation of Threaded Applications. In: Proc. *7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.* Atlanta, Georgia, May 4-6, 1999.

Parallel Program Design Tool with Application Control Methods Based on Global States

Janusz Borkowski[1], Marek Tudruj[1,2], and Damian Kopanski[1]

[1] Polish-Japanese Institute of Information Technology
86 Koszykowa Str., 02-008 Warsaw, Poland
[2] Institute of Computer Science, Polish Academy of Sciences
21 Ordona Str. 01-237 Warsaw, Poland
{janb, damian, tudruj}@pjwstk.edu.pl

Abstract. In this paper we develop a high-level parallel program synchronization and control mechanism. A specialized controller process (or a hierarchy of such processes) collects information about process states and construct Strongly Consistent Global States using time interval timestamps. Global predicates evaluated by controllers prompt sending of control signals to processes and the signals trigger computation activation or cancellation. The proposed framework is integrated with a message passing system – it is added to the GRADE visual parallel programming environment to enhance its message-passing based features. The GRADE enhancements introduced here are discussed.

1 Introduction

Message passing has become one of the most successful parallel programming paradigms. Its success can be attributed mainly to the standardization enabled by PVM and MPI libraries. However, writing programs using these libraries remains difficult. To overcome this problem high-level parallel programming tools have been developed. The GRADE system, developed in Hungary in the SZTAKI Institute, is one of the most advanced complete parallel programming environments [8,9,13]. A GRADE user does not need to know anything about underlying message passing (MP) libraries, but he is still confined within the strict message-passing paradigm. In [1] an advanced synchronization environment for multi process parallel applications has been proposed. The environment provides control and synchronization facilities in parallel applications. They are based on asynchronous evaluation of high-level conditions (predicates) defined on application global states. Adding this environment to GRADE overcomes many GRADE limitations and introduces a number of novel positive features. So far global states were used for verification and debugging [5,12,7,16]. Control based directly on them can lead to applications correct by constructions and also immediately verifiable. Putting synchronization/control code into one place - into predicate definitions - leads to a better structured and easy to maintain program. In the proposed environment predicate satisfaction triggers sending of a control signal to a process. The process upon receiving the signal immediately

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 338–343, 2004.
© Springer-Verlag Berlin Heidelberg 2004

activates a predefined procedure, which is an integral part of the parallel application. In that way higher priority or more relevant actions are activated by predicates, without any form of process waiting.

2 Application Control Based on Global Predicates

It is not easy to capture global states in a parallel system without common clock and without shared memory. Logical vector clocks [10,4] or rough local clock synchronization must be employed [14]. Processes should send messages about their local states with attached timestamps to a controller, which combines the received information to identify application Consistent Global States (CGS). In general it is not possible to know the actual CGS sequence and to answer whether an application run has passed through a CGS satisfying a given predicate [5,4,7]. However, we need to know on-line, as early as possible, what is the actual application state. These requirements are met best by predicate modality *Instantly* [14]. To be able to apply it, we need to synchronize process local clocks with an assumed tolerance ε and use real time interval timestamps. CGS obtained in that manner are called Strongly CGS (SCGS). Global state $s = <s_1, ..s_N>$ is a SCGS iff there was a period in real time during application run, when for $i = 1..N$ simultaneously process P_i was in state s_i. $Instantly(\varphi)$ tells whether an application has passed through a SCGS, in which predicate φ was satisfied. In our environment whenever a predicate is satisfied, the monitor undertakes an action that influences application execution by sending control signals. Processes should react as quickly as possible to incoming signals. These objectives are met by activation and cancellation [1,2]. In the code of a process, designated regions are made sensitive to incoming signals. If the process control is inside a region sensitive to a signal of type *T,* and such a signal arrives, then a reaction is triggered. The reaction can be either activation or cancellation. Activation makes the current computation to be suspended and a reaction code associated with the region to be executed. After completion of the reaction code the suspended computing resumes. Cancellation makes the current computation to be stopped and a cancellation handling procedure associated with the region to be performed. The program execution resumes just after the abandoned region. Fig. 1 illustrates this concept.

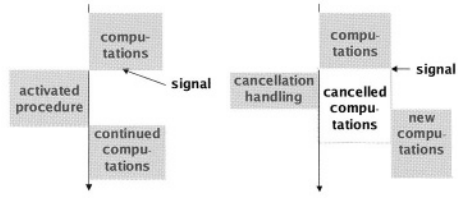


Fig. 1. Principle of activation (left part) and cancellation (right part)

The benefits of such program execution control have been discussed in [3] with a branch-and-bound (B&B) algorithm taken as an application example. Processes report to the controller newly found solutions, their current load and bounds of the currently considered problem. The controller propagates the global best solution as a value attached to a signal, while an activated procedure passes immediately the new value to the ongoing computations. The controller also monitors load imbalance and activates load balancing actions (give load, receive load). The most advanced feature is the ability to preempt problems being currently solved if their bound is lower than the newly found global best solution. This is done by using cancellation.

3 Extended GRADE

GRADE is a visual parallel programming environment based strictly on message passing. The programmer has only to draw rectangles representing processes, connect them by channels, draw control flow diagrams to define behavior of each process, supply sequential C code representing sequential parts of processes and assign which variables should be sent/received in which points of the program [8,9,13]. Our target is to enhance GRADE by adding control and synchronization methods based on application global state analysis. A controller, as described above, is represented as a special kind of a process. Using dedicated channels, application processes send to it messages about their local states; these messages are properly timestamped. The controller constructs SCGSs using obtained information, evaluates defined predicates on them and sends back control signals. E.g. to monitor the bound of the currently considered branch&bound subproblem, the controller exposes a dedicated input port. Processes send there the current value of the bound, the values are stored in an array associated with the port, and each process has one corresponding element in the array. So, global states are abstracted as array values. Upon reaching a SCGS defined predicates are evaluated, see fig. 2. Predicates are expressed using the same control flow diagrams as used for process definitions. They can examine current application state by reading arrays that contain state information.

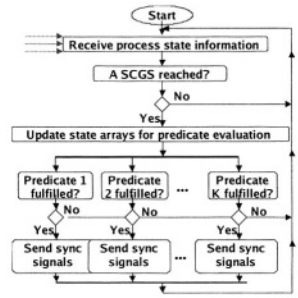


Fig. 2. Control flow diagram of a synchronizer

There can be many synchronizers in the system, each taking care of particular aspect of the application state, e.g. one monitors system imbalance, another keeps track of what each process is computing currently. This is a simple way to decentralize the control and communication avoiding bottlenecks. However, for a large number of processes and for complex predicates the amount of computations and communication performed by the controller can be problematic. We introduce controller hierarchies to overcome that threat. Application processes can be split into groups. Each group cooperates with its own synchronizer. The synchronizers are connected to a higher-level synchronizer, there can be many levels in the hierarchy, see an example on Fig. 3. The interaction between levels works as follows: a lower level controller sends a message about the current group state to its parent controller as a result of predicate evaluation. The parent controller combines messages from subordinate controllers to form a global state. Details of hierarchical SCGS construction are currently under consideration. Practically, the parent controller can know when and which predicates at the subordinate controllers are satisfied. Higher-level predicates can be constructed based on that knowledge and their evaluation can produce control signals. The signals are propagated to lower levels until they reach processes. Every predicate in the form $\varphi = \varphi_1 \circ \varphi_2 \circ .. \circ \varphi_N$, where φ_i is a local predicate for process P_i, and $\circ$ is an associative operation, fits very well in the described framework. $\varphi g_{i-k} = \varphi_i \circ .. \circ \varphi_k$ are evaluated at group level, $\varphi = \varphi g_{1-s} \circ .. \circ \varphi g_{r-N}$ is evaluated at the top level. In that way the evaluation is performed in a parallel and distributed manner. Barrier is a simple example: $\varphi = P_1$ *has reached barrier* $\wedge .. \wedge P_N$ *has reached barrier.* A low level synchronizer checks whether processes $P_i, .., P_k$ have reached the barrier while the top level synchronizer verifies that all subordinate process groups have hit the barrier.

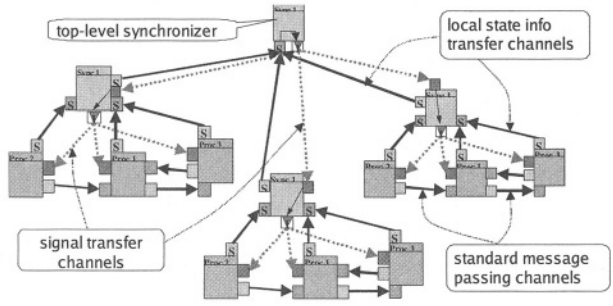


Fig. 3. Connections between processes and synchronizers

The process control flow diagram has been also extended to express the new functionality. A simplified example of a process control flow diagram accepting control signals is presented in Fig. 4. The normal execution flow goes along path marked by a dotted line. If a signal arrives on ports 1 or 3 when the process execution is within a dashed rectangle then the control is transferred to the right-hand side branch.

Clock synchronization necessary for SCGS detection is achieved with the use of NTP [11], but more precise mechanisms will be employed in the future e.g. RBS [6]. A special version of SCGS detection algorithm has been developed to work with unterminated states. Such a version allows for quicker reactions, as it does not wait for state termination events to detect SCGS and to evaluate predicates. The time between reporting of a new state to the synchronizer and receiving a control signal is an important parameter of our system. It determines parallel task granularity, which can be managed effectively by the controller. This time can be estimated as twice the message transfer time + watch-dog timer delay used by our SCGS detection algorithm + maximum skew between local clocks + SCGS detection algorithm runtime. For Myrinet 2000 network this time is less than 100μs, and can be improved 3-6 times by using even faster networks (Quadrics, DIMNET [15]).

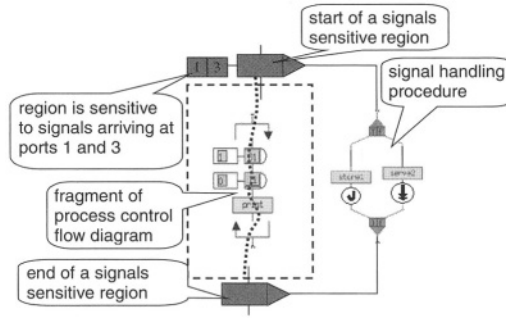


Fig. 4. Flow diagram sensitive to synchronization signals

4 Conclusions

In this paper we have presented a parallel programming environment, which combines the standard message-passing paradigm with an advance synchronization and control model based on application global states and global predicates. The idea of using global predicates to control application on-line is combined with special process reactivity features - asynchronous activation and cancellation mechanism. We propose to separate data transmissions from synchronization and control aspects. The former is very well realized by message passing functions, the latter is performed using of the proposed model. In this way the control code is well separated from the proper application code and it is easy to be understood and maintained. Also, with the help of global predicates one can implement application control and synchronization, that are correct by construction, and one can verify them without any separate tool. Our target is to have a user-friendly system with the described capabilities. We build such a system by enhancing the GRADE visual parallel programming environment. Our proposals integrate smoothly with the GRADE user interface philosophy, making it fairly easy for a programmer to understand and use them.

Acknowledgements. The work has been sponsored by KBN grant 4T11C 007 22 and an internal PJIIT research grant.

References

1. J. Borkowski, "Interrupt and Cancellation as Synchronization Methods", Proc of 4th Int. Conf. Parallel Processing and Applied Mathematics PPAM 2001, Naleczow, Poland, Springer 2001, LNCS 2328
2. J. Borkowski, D. Kopanski, M. Tudruj, "Implementing Control in Parallel Programs by Synchronization-Driven Activation and Cancelation", Proc. of the Eleventh Euromicro, Italy, IEEE 2003
3. J. Borkowski, D. Kopanski, M. Tudruj, "Adding Advanced Synchronization to Processes in GRADE", in Proceedings of the PARELEC 2002, Warsaw, Poland, IEEE 2002
4. O. Babaoglu and K. Marzullo, "Consistent global states of distributed systems: fundamental concepts and mechanisms", in: Distributed Systems, Addison-Wesley, 1995
5. R. Cooper and K. Marzullo, "Consistent detection of global predicates," Proceedings ACM/ONR Workshop on Parallel Distributed Debugging, pages 163-173, 1991.
6. J. Elson, L. Girod and D. Estrin. "Fine-grained network time synchronization using reference broadcasts." In Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachussetts, USA, December 2002. To Appear.
7. V. K. Garg, B. Waldecker, Detection of Strong Unstable Predicates in Distributed Programs, IEEE Trans. on Parallel and Distrib. Systems, Vol. 7, No. 12, December 1996 (pp. 1323-1333)
8. Kacsuk, P., Dozsa, G. and Fadgyas, T., "GRADE: A Graphical Programming Environment for PVM Applications", Proc. of the 5th Euromicro Workshop on Parallel and Distributed Processing, London, 1997, pp. 358-365
9. P. Kacsuk, G. Dozsa, T. Fadgyas, R. Lovas, "The GRED Graphical Editor for the GRADE Parallel Program Development Environment", Future Generation Computer Systems, No. 15 ('99), pp. 443-452.
10. F. Mattern. "Virtual Time and Global States in Distributed Systems". Proc. Workshop on Parallel and Distributed, Algorithms, Chateau de Bonas, Oct. 1988, M. Cosnard et al. (eds.), Elsevier / North Holland, pp. 215-226, 1989.
11. D. Mills, Network Time Protocol (Version 3) Specification, Implementation and Analysis, RFC1305
12. M. Minas, "Detecting Quantified Global Predicates in Parallel Programs", Europar 95 , Stockholm, Sweden. Proceedings. Lecture Notes in Computer Science, Vol. 966, Springer, pp 403-414
13. The P-GRADE Visual Parallel Programming Environment, http://www.lpds.sztaki.hu/index.php?menu=pgrade&&load=pgrade.php
14. S. D. Stoller: "Detecting Global Predicates in Distributed Systems with Clocks". Distributed Computing, Volume 13 Issue 2 (2000) pp 85-98
15. N. Tanabe et al., "Low Latency Communication on DIMMnet-1 Network Interface Plugged into a DIMM Slot", Proceedings of the PARELEC 2002., Warsaw, Sept. 2002, pp. 9-14.
16. A. Tarafdar and V.K. Garg. "Predicate Control for Active Debugging of Distributed Programs", Symposium on Distributed and Parallel Debugging, 1998.

Event Handling in the J-OCM Monitoring System

Marian Bubak[1,2], Włodzimierz Funika[1], Marcin Smętek[1], Zbigniew Kiliański[1], and Roland Wismüller[3]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
{bubak,funika}@uci.agh.edu.pl, {smentos,zkilian}@icslab.agh.edu.pl
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM – Technische Universität München, D-80290 München, Germany
wismuell@in.tum.de
Phone: (+48 12) 617 39 64, Fax: (+48 12) 633 80 54, Phone: (+49 89) 289 28243

Abstract. The demand for tool support for distributed Java programming is increasing. A universal, open interface between tools and a monitoring system, On-line Monitoring Interface Specification (OMIS), allows to extend the functionality of the OMIS Compliant Monitoring system (OCM) for building a Java oriented monitoring infrastructure which provides all indispensable functions for diverse kinds of tools. The paper presents an approach to dealing with the target Java system in terms of the event model established in the J-OCM.

Keywords: Java, monitoring system, monitoring interface, distributed object system, OMIS.

1 Introduction

Java technology has grown in popularity and usage because of its portability. This simple, object oriented, secure language supports multi-threading and distributed programming including remote method invocation, garbage collection and dynamic class loading. There are many performance problems for Java programmers. The *Garbage Collection mechanism* may influence the performance of application due to a possibly large heap and the asynchronous mode of operation. The *Object Oriented* nature of Java programming causes the use of a very large number of classes and therefore a lot of jumps in control flow between pieces of software, which may need to be optimized. The *Dynamic Class Loading* can have a significant impact on the amount of memory used, moreover the JVM sometimes loads classes before they are needed. *Memory leaks* occur when an instance of a longer life cycle has a reference to an instance of a shorter life cycle, which prevents the instance with a shorter life cycle from being garbage collected. Remote Method Invocation (RMI) combines the problems of pure Java with those stemming from the distributed Java programming: *downloading of stubs* needed by a client may cause downloading of indispensable classes through a web server, thus generating large network traffic; the use

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 344–351, 2004.
© Springer-Verlag Berlin Heidelberg 2004

of *Distributed Garbage Collection* protocol and *Object Serialization* introduce a substantial overhead on the performance of RMI calls.

Our goal is to build a comprehensive tool support for building distributed Java applications by providing uniformed, extendible monitoring facilities for communication between components, for analyzing an application's execution, and for detecting bugs.

The primary goal of the J-OCM is to allow programmers to develop monitoring tools for Java distributed applications. The new concept considers the J-OCM a distributed monitoring platform for building portable tools. As in the original OMIS, the J-OMIS [1] uses the notation of objects which is an important part of the interface. Although the J-OCM is not exactly an object oriented system in the usual sense, the underlying concept of J-OMIS is a set of object types provided with support of object specific services.

The tool is provided with transparent access to software objects such as: *node* objects, *JVM* objects, *threads, class* objects etc. For each object type a set of appropriate services is defined. Condition requests represents services used to detect events, being emitted by software objects.

The architecture of the J-OCM comprises *Node Distribution Unit* (NDU), responsible for distributing tool requests [1] and assembling replies, *Local Monitors* (LM), providing monitoring activities on local objects, *JVM Local Monitors* (JVMLM) which are agents embedded into JVM processes. The *Shared Memory based Local Agents Environment* (SHMLAE) is a communication layer to support cooperation between agents and the LM. It provides an application programming interface (API) for the parallel programming model based on signals and shared memory regions. It offers *non-blocking send* and *interrupt/poll driven receive* operations to support monitoring techniques used in the OCM, based on the event-action paradigm. The J-OCM uses asynchronous communication to avoid blocking of the monitoring processes and Java agents in order to provide fast reactions on occurring events.

In an event-based monitoring system, basic events, characteristic atomic program state transitions are captured by sensors which are inserted in the target system and sent to the monitoring system. The monitoring system takes some action(s) - a sequence of instructions associated with the event. These actions can either carry out data collection, or also manipulate the running program. In order to follow the idea of event-based monitoring both the LM and the JVMLM must support the event notification.

The paper is organized as follows. A short presentation of the J-OCM monitoring techniques based on event detection in Section 2 is followed by a consideration of the event model in OMIS (Section 3). Then follows an overview of the extension to the general event model by Java-oriented events (Section 4). Section 5 gives a characteristics of the interaction scheme between the low-level components of the J-OCM, based on the event detection. Conclusions and future work are summarized in Section 6.

[1] A detailed discussion on the request processing mechanism in the J-OCM can be found in [2]

2 Java-Oriented Techniques for Event-Based Monitoring

JVM notifies several internal events to the the JVMLM, using JVMPI [6] and JVMDI [7]. To support the interactive observation of the target system all events must be processed by the JVM agent and sent to LM selectively. This approach, however, induces too much overhead on the agent. In JVMPI, event notification can be selectively enabled and disabled at runtime using `EnableEvent` and `DisableEvent` function. But the agent cannot disable all events, because it will be receiving incomplete monitoring information. For example, if the agent enables the event indicating *class loading* after a number of classes have been loaded, the agent will not be able to reconstruct the list of previously loaded classes. In this situation the JVMLM must record all events, these are necessary to avoid this kind of problems. To control the forwarding of events, the agent uses a filter in form of a table, where it stores information about what events the LM is interested in. The LM manipulating this event filter table can *stop* or *resume* the notification of specific events sent from its agents. The JVM calls `NotifyEvent` function to inform the JVMLM of run-time events. Events are represented by the data structures consisting of an integer indicating the type of event and the identifier of the thread whose execution caused the event. All events supported by JVMPI are fired by changes in the state of Java threads *(started, ended, blocked on a locked monitor),* the beginning/ending of an invoked method, class loading operations, object allocation/deallocation, and the beginning/ending of JVM garbage collection.

3 The OMIS Event Model

The OMIS Event Model, along with the *event* term, additionally defines the *event class* predicate, specifying a set of event occurrences, a pattern defining the events of interest. An event belonging to an event class is said to match the class as a given *pattern.* In the OCM, event classes are represented by an event service together with its parameters. E.g. `jvm_loaded_class([j_1,j_2])` defines the event class comprising all events where `j_1` and `j_2` loaded any class. The event processing in the OCM is based on the idea that event classes form a tree hierarchy, where the root is the universal class containing all detectable events. New event classes are derived from the existing ones via the use of filters that allow only certain events to pass. E.g. we could have an event service `jvm_thread_control(token* thread_list)` detecting when an event generated by the JVM process is a thread-related event and compares event context parameters - parameters that are set on occurrence of any event containing the thread token - against the tokens specified in a `thread_list`. Then the event class `jvm_thread_control([t_1])` represents the event fired only by `t_1`. The event hierarchy can be expanded by adding the `jvm_thread_ended([t_1])` class event that can be derived from `jvm_thread_control([t_1])` using the filter that checks whether the JVM (executing `t_1`) fired a `THREAD_END` event.

Each class in the event class tree may be associated with a set of filters, each of which is used to derive a more specific event class from a more general one. In

addition, some event classes may have action lists associated with them. When an event is detected by the OMIS compliant monitoring system, it is matched against all the event classes in the tree, by doing a tree traversal starting at the tree's root, which matches all events. At each traversed node, the filters associated with that node are evaluated. If a filter evaluates to true, the event matches the derived event class, which is therefore traversed, too. During this tree traversal, the action lists associated with the event classes matched by the event are scheduled for execution.

The event class in the OCM is represented by an abstract data type. For historical reasons, this data type is called `Ocm_ip` or simply `IP`.

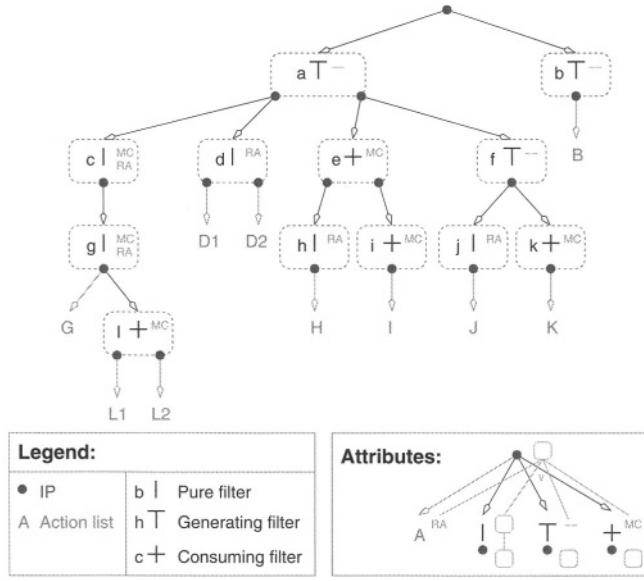


Fig. 1. An example of IP tree in the OCM

Figure 1 illustrates an example event class hierarchy, where we can distinguish between principally tree different kinds of filters:

1. *Pure filters* that get an event as input, analyze that event and probably let it pass.
2. *Generating filters* that detect new events. These filters ignore their input events. They just use it as a means to trigger their execution. The filter will check for the occurrence of a new event. An example for this kind of filter is one that detects the arrival of a message. It will be invoked each time the monitor gets a signal, but the signal does not really tell us that there is a message, it merely is used to invoke the filter which uses a non-blocking receive call to determine whether there really is a message.
3. *Consuming filters* that (may) consume their input event when they trigger.

The event hierarchy of the OCM consists of some event classes, characteristic for the core of the LM such as `mon_received_signal`, `monitor_executes_hook` and for specific monitor extensions. To install new event services, four functions must be provided:

1. The *event define function:* This function constructs an IP representing event class defined by the event service and its actual parameters.
2. The *instrumentation function:* Some events can only be detected after certain manipulations have been done to the target system. These manipulations are usually called *instrumentation.* An example is to install a signal handler in a target process in order to detect that this process receives signals. The instrumentation function is responsible for inserting and removing this kind of instrumentation.
3. The *filter function:* As explained above, filters are used to derive more specific event classes from more general ones. The filter function is called whenever an event matching the more general event class is detected. The filter then has to determine whether the event matches one or more of the more specific event classes. This decision is usually based on the values of event context parameters, but can also be based on the target system that is acquired by the filter itself. The filter then executes the set of child IPs that represent the event classes matched by the processed event.
4. The *event delete function:* This callback function is invoked to perform additional cleanup, before an IP is deleted from the IP tree.

4 Event Processing in the J-OCM

The J-OCM as adaptation of the OCM for Java application extends the event tree of the OCM by its own IP subtree. The new event hierarchy, shown in Fig. 2, consists of three types of event classes.

The `jvm_any` is the 'root' of Java related event classes and is triggered whenever any of JVMs registered in the monitoring system generates an event. The `jvm_any` is a *Singleton* [2], the event class that has only one instance to provide the global point of access to it in the Local Monitor.

All other event classes, which relate to other object types specified in J-OMIS e.g *jvms, threads, classes, etc.,* are derived from this one using filters. The `jvm_control`, `jvm_{thread, class, object, method}_control`, which represent abstract object control event classes, comprise all event occurrences, characteristic for the suitable type of object. The term 'abstract' is used in the sense that these event classes cannot be involved in a condition request via the J-OMIS tool/monitor interface. These event services hook up to its parent event service, installing its filter to this IP and generate a new one. The new generated IP is triggered, when an object control event for one of the objects, passed as parameters to the event service, has been detected. In order to allow its filters to determine which of generated IP must be executed, the object event services

[2] Singleton Design Pattern(see [3])

Fig. 2. Hierarchy of J-OCM events

maintain a list of objects, where each object is associated with the list of IPs generated for this object.

Leaves, the elements of the tree which do not have children, represent the events which are defined by the interface specification, i.e. J-OMIS. The J-OMIS classifies event services based on the categories of elements of Java program architecture the services operate on. The J-OCM event tree represents a similar classification, where the control services groups the event services operating on the same type of token. But some event services were moved over in the event hierarchy in order to recognize the situation (context) where the event took place. There are two kinds of information which are needed to determine whether an event matches a given event class: the time of occurrence of an event and location (context) of the event. The shape of the J-OCM events tree indicates to the second one. As mentioned above, the event class that is a pattern of an event occurrence is specified in the OCM by four functions. The *event define function* specifies the conditions which enable to detect the occurrence of a specific event. For example, an event service:

```
jvm_method_entered(token* method_list, token* interface_list,
                   token* object_list, token* thread_list)
```

is realized when `thread` in `thread_list` enters the `method` in `method_list` defined by `interface` in `interface_list`. The `object` in `object_list` determines the instance of the class for which the method is invoked. All these parameters will be needed to detect the proper event, but some of them are statical, so they do not change during the target system's run-time, and others determine the context of the event occurrence. E.g. each interface in the list of interfaces for `jvm_method_entered` event class can be converted by *interface to method expansion* operator, defined in the J-OMIS, to a set of methods that are declared in the given interface and defined in the classes which implement this interface.

Thus, there is no need to take into consideration information about interfaces in order to detect events, which match `jvm_method_entered` event class, by filters. This is the reason, why events classes, relating to the method objects do not have parents from the event classes relating to the interfaces in the J-OCM event hierarchy. In the above example, the most important for detecting method-bound events are the tokens referring to two last parameters: objects and threads.

5 Event-Based Interaction in Low-Level Components

The most critical part of J-OCM event system are interactions between the Local Monitor and its Java agents (JVMLMs). The Java agents use the SHMLAE to notify the LM of the event occurrence and to transfer event-specific parameters. A specific event, before being handled by the LM, must be enabled in the specific JVMLM. This instrumentation is performed, when a tool enables the previously defined event service request, by issuing `csr_enable` service and the `instrumentation function` for this event service, is called. E.g. the instrumentation function for `jvm_thread_ended([t_1])` event service enables THREAD_ENDED event notification in the JVM agent associated with the JVM where the thread represented by `t_1` token is running. Once the JVM agent has received the request from the LM, it starts passing events to the LM which must take care of handling of them.

The Local Monitor process operates in an event driven fashion - it is normally blocked in a system call, which returns when an event in a monitored process has been detected. In addition, the call is interrupted when a message arrives. The OCM uses signals as asynchronous mechanism to notify the message arrival and defines a special event class, called `mon_received_signal(integer sig_no)` that is triggered when the monitor process receives the specified signal. The use of the event class is the solution to handle events from the JVM agents, and next to process them, according to the OMIS Event Model. The universal event class `jvm_any` relating to Java specific events derives from `mon_receive_signal(SHMLAE_signal)` event class, where `SHMLAE.signal` is the signal number used by the SHMLAE. The filter that is located between these two event classes (Fig. 2) is a generating filter (Section 3) and is invoked each time the monitor gets a `SHMLAE.signal`. It uses a non-blocking receive call provided by the SHMLAE to get messages containing information about an event that has occurred in a monitored JVM. Based on this information, the filter updates the event context parameters specific for the detected event and stores the event tag in the proxy object of the JVM where the event took place. Finally, the filter executes `jvm_any`, the 'root' of the Java event classes hierarchy, where further event processing takes place.

6 Concluding Remarks

The idea of separating the layer of tools from a monitoring system's functionality underlies the concept of building a portable extensible monitoring infrastructure.

We extended the On-line Monitoring Interface Specification by a Java specific hierarchy of objects and a set of relevant services. The work on a Java-oriented monitoring system, the J-OCM, concentrated on extending the functionality of Local Monitors which are the distributed part of the system and control the activities in the low-level Java-bound monitoring agents via a hierarchy of events. We have extended the original event model provided in the OCM by a Java specific event sub-model which cover the functioning of the basic application and execution entities of the Java distributed application. This enables to provide a comprehensive monitoring functionality for Java tools.

Our on-going work focuses on completing the implementation of the J-OCM and extending it by monitoring RMI calls.

Acknowledgement. This research was carried out within the Polish-German collaboration and it was partially supported by the KBN grant 4 T11C 032 23.

References

1. M. Bubak, W. Funika, P. Mętel, R. Orłowski, and R. Wismüller: Towards a Monitoring Interface Specification for Distributed Java Applications. In Proc. 4th Int. Conf. PPAM 2001, Nałęczów, Poland, September 2001, LNCS 2328, pp. 315-322, Springer, 2002.
2. M. Bubak, W. Funika, M. Smętek, Z. Kiliański, and R. Wismüller: Request processing in the Java-oriented OMIS Compliant Monitoring System. Accepted to the 5th Int. Conf. PPAM 2003, Częstochowa, Poland, September 2003.
3. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Addison-Wesley, 1995.
4. T. Lindholm and F. Yellin, *The Java Virtual Machine Specification,* Addison-Wesley Publishing Co., Reading, MA (1996).
5. T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series, (1997) `http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz`
6. Sun Microsystems: Java Virtual Machine Profiler Interface (JVMPI) `http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html`
7. Sun Microsystems: Java Virtual Machine Profiler Interface (JVMDI) `http://java.sun.com/products/jdk/1.2/docs/guide/jvmdi/jvmdi.html`
8. Sun Microsystems: Java Platform Debug Architecture (JPDA) `http://java.sun.com/j2se/1.4.1/docs/guide/jpda/index.html`
9. Sun Microsystems: Java Native Interface (JNI) `http://java.sun.com/products/jdk/1.2/docs/guide/jni/`
10. R. Wismüller, J. Trinitis and T. Ludwig: A Universal Infrastructure for the Runtime Monitoring of Parallel and Distributed Applications. In Euro-Par'98, Parallel Processing, volume 1470 of Lecture Notes in Computer Science, pages 173-180, Southampton, UK, September 1998. Springer-Verlag.

Request Processing in the Java-Oriented OMIS Compliant Monitoring System

Marian Bubak[1,2], Włodzimierz Funika[1], Marcin Smętek[1], Zbigniew Kiliański[1], and Roland Wismüller[3]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
{bubak,funika}@uci.agh.edu.pl, {smentos,zkilian}@icslab.agh.edu.pl
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM – Technische Universität München, D-80290 München, Germany
wismuell@in.tum.de
Phone: (+48 12) 617 39 64, Fax: (+48 12) 633 80 54, Phone: (+49 89) 289 28243

Abstract. Performance problems with Java distributed applications due to their complexity, design problems, and communication mechanism motivate building a uniform Java oriented monitoring environment that provides on-line access to the target distributed application and particular elements. A prototype monitoring system, the J-OCM, compliant with the On-line Monitoring Interface Specification (OMIS) concept, provides the ability to observe and manipulate a whole distributed Java application's execution. The paper presents an approach to dealing with the target Java system in terms of components of the distributed system architecture: interface definition, proxy, object manager, naming service, and communication protocol. To follow these approach we consider the stages of processing the requests issued by monitoring based tools like performance analyzers, debuggers, visualizers.

Keywords: Java, monitoring system, monitoring interface, distributed object system, OMIS.

1 Introduction

The Java language is a popular basis for building distributed services and application because it hides the problems that arise due to the heterogeneity of server and client HW and SW platforms. The Remote Method Invocation mechanism, intended for supporting distributed object based computing, allows client/server based distributed applications to be developed easily because a client application running in Java virtual machine at one node can invoke objects implemented by a remote Java virtual machine in the same way as local objects. However, after the development and deployment process of an RMI-based application is finished, performance problems may occur. To understand the nature of problems and to find out which elements of distributed application are responsible for causing undesirable behavior, appropriate monitoring tools (performance analyzers, debuggers etc.) are needed. But distributed applications pose one major

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 352–359, 2004.
© Springer-Verlag Berlin Heidelberg 2004

problem when it comes to testing - they are distributed over several machines and therefore are difficult to monitor.

The primary goal of the Java oriented OMIS Compliant Monitoring system (J-OCM) [1] is to allow programmers to develop monitoring tools for Java distributed applications. The new approach makes the J-OCM a basis for a distributed monitoring environment.

The major concept of the Java oriented On-line Monitoring Interface Specification (J-OMIS), which underlies the J-OCM, is a set of object types with support of object specific services. The tool (performance analyzer, debugger, visualizer etc.) is provided with access to such objects as *node* objects, *JVM* objects, *threads, class* objects etc. For each object type a set of appropriate services is defined. Besides *software objects,* some *internal objects* of the monitored system are defined.

Based on J-OMIS, we have extended the functionality of the OCM[2], adding new software components and adapting existing ones. This approach allowed to combine the existing functionality of the OCM with Java platform to support Java homogeneous and heterogeneous in the future computing. The *Node Distribution Unit* (NDU) is responsible for distributing requests and assembling replies. E.g. the tool may issue a request in order to run Garbage Collector on specified JVMs, therefore the NDU must determine the nodes executing the JVMs and, if needed, to split the request into separate sub-requests to be sent to the proper nodes.

The *Local Monitor* is a monitor process, independent from the whole global monitoring infrastructure. Each monitor process provides an interface similar to that of the NDU, but it accepts requests to operate on local objects. LM stores information about the target Java application's objects. The *Java Virtual Machine Local Monitor* is an agent embedded into a JVM process, which is responsible for execution of the requests received from the LM. It uses Virtual Machine native interfaces such as JVMPI, JVMDI, JNI that provide low level mechanisms for interactive monitoring, independent of a JVM implementation. The *Shared Memory based Local Agents Environment* (SHMLAE) is a communication layer to support cooperation between agents and the LM.

All these features allow to see the whole monitored system as a set of distributed objects and the monitoring system as a higher-level software layer (middleware) that provides a standardized interface for the tool to access monitored objects, regardless of implementation details, like hardware platform or software language.

Generally speaking, every distributed client-server system is based on the important principle that the definition of behavior is separated from the implementation of the behavior. The *client* is concerned about the definition of services whereas the *server* focuses on providing an implementation of services. Beside that, in order to meet the requirements of distributed computing, the distributed system has to comprise additional architectural elements: an object interface specification, object stub and skeleton, object manager, registration/naming service and communication protocol. To deal with the distributed

target system, we consider the functioning of the J-OCM as a distributed client-server system with focusing on the functionality of its components.

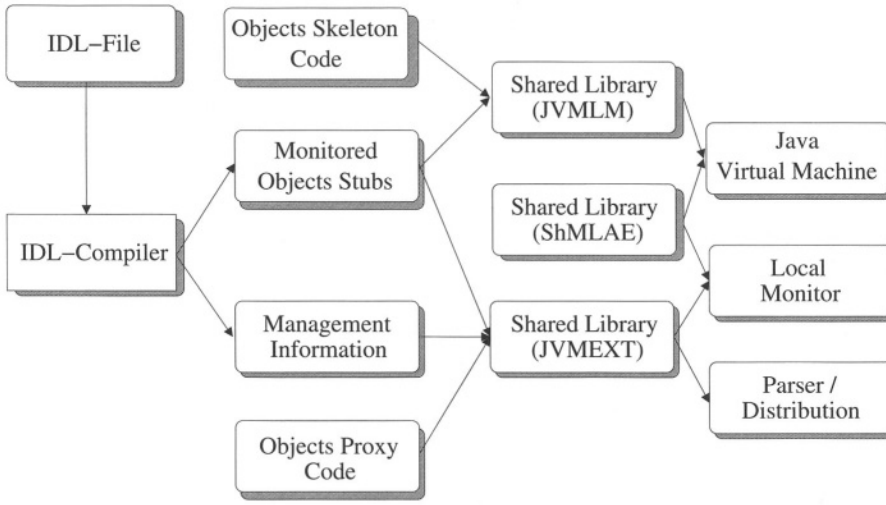


Fig. 1. Development process of the J-OCM

The paper is organized as follows. A short presentation of the J-OCM basics in Section 1 is followed by a general consideration of defining the tool interface for monitoring (Section 2). Then follows an overview of the J-OCM's components: stub/skeleton (Section 3) and naming service (Section 4). Section 5 gives a characteristics of the transport layer in the system. Conclusions and future work are summarized in Section 6.

2 Interface Definition

The main goal of each distributed environment such as CORBA, COM/DCOM, Java RMI is to simplify the process of developing a distributed application. The first stage of this process is to define the interface of a remote object (e.g. methods, data types), written in an Interface Definition Language (OMG IDL, Java interface), which enables to describe the interface in some independent manner. Next, the IDL file is compiled with a utility program to a file in a specific language, to generate the *client stub* and *the server* skeleton. Similarly, the leading idea of OMIS is to provide support for building monitoring tools and systems for new parallel or distributed environments by extending its functionality. The development of an extension resembles that of an application in the case of the distributed middleware.

1. For each extension, an IDL file, called *registry,* has to be provided that specifies new objects, new services and their relevant attributes.

2. The OCM comprises of a suitable utility, called *mkregistry* to translate the registry file into a *extension stub,* files containing service templates and some management information, needed to register new objects and services. Except for the services defined in the interface specification, the translator generates templates for three additional functions. These functions are called by the monitor core:
   - to localize an object,
   - when the object is about to be deleted (object's destructor),
   - when the object is attached/detached by the tool.
3. For each service in the extension, one or more C functions have to be written. The functions for all services of the extension are collected into a single extension library.
4. The library is then linked to the OMIS compliant (J-OCM) monitoring system using a special tool.

3 Stub and Skeleton

Stub and *Skeleton* provide transparent communication between the client and remote object. They are based on `Proxy Design Pattern`[1], where the object is represented by another object (the proxy), in order to control access to the object. The proxy object implements the same interface as the real object, so from the client's point of view, the proxy appears to be the actual object. In distributed computing the stub plays the role of the proxy and allows to make a local call on the remote object. It intercepts a client's call, performs some administrative operations, and forwards it to the remote proxy, called *skeleton.* The skeleton, residing on the server side, receives an incoming call, and invokes it on the real object.

The Proxy pattern is used by the J-OCM to provide the tool with transparent access to monitored objects[2]. The monitored objects are identified by tokens which refer to the proxy object. The proxy is a representation of the real object in the monitoring system. However, access to the proxy object is limited to the node where the object resides, except for node tokens, where the corresponding *node proxy* objects exist both in local monitor processes and NDU. The object proxy contains all information that is needed to deliver tool's requests to the JVM agent (JVMLM) that directly accesses the JVM. The proxy approach allows to perform some operations requested by the tool only at the monitor representation of a target object without engaging the agent's functionality. E.g. the proxy object can cache statical information about the object that it represents, retrieved by information services to avoid repeated communication with agents. Fig. 2 illustrates the general concept of the proxy access in the monitoring infrastructure. The JVM agent acts as a skeleton, while the remote proxy which is embedded into JVM as a platform dependent native library. JVM loads the agent

[1] Proxy design pattern as described in "Design Patterns" by Gamma, Helm, Johnson and Vlissides [11]

[2] The word *object* is used in terms of the J-OMIS object hierarchy

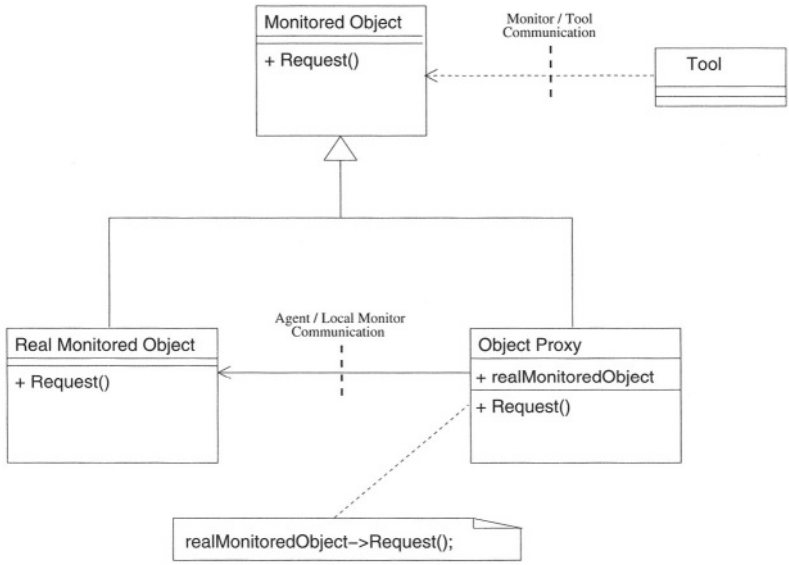


Fig. 2. Proxy Design Pattern

at startup and looks for a specified entry point. The agent transforms a call and parameters received from the LM into the format required by one of interfaces used to interact with JVM. The interaction mechanism between the agent and JVM is a critical part of the agent's implementation. The JVMLM uses three available interfaces depending on the purposes of performed operations: the `Java Virtual Profiling Interface` (JVMPI) [7] for obtaining profiling data, `Java Virtual Machine Debug Interface` (JVMDI) [8] to query the current status of an application and to control its execution, and `Java Native Interface` (JNI) [10] to patch the lacks of the previous ones. While avoiding the details of use of each interface, we are focusing on support for manipulation and information services provided by JVMPI (events support will be explained later). JVM implements a set of call-back functions to enable the monitoring agent to set control parameters and obtain information about the JVM's state and the application being executed. The following are key call-back functions operates on two kinds of JOMIS' objects: *thread* and *JVM.*

- `SuspendThread`, `ResumeThread` - used to suspend and resume threads.
- `ThreadHasRun` - used to determine whether a thread has run since the last time it was suspended.
- `GetThreadStatus` - called to obtain the status of a thread, whether it is *runnable, blocked,* or *waiting on a monitor,* and whether it has been suspended or interrupted.
- `EnableGC`, `DisableGC`, `RunGC` - called to *enable, disable,* or *run the garbage collector.*

JVMDI provides more powerful functions exploited in direction from the agent to the JVM, which allow for: memory management, thread-oriented operations,

access to stack frame and local variables, setting breakpoints and watched fields, getting information on classes, objects, fields, and methods, as well for support of Java monitors.

Once the agent has been notified that the JVM is initialized, it starts a *background communication thread.* The thread initializes communication by registering the agent in the SHMLAE and waits for incoming requests from the LM. When it intercepts a call, it unmarshals parameters and invokes the proper local method implementing directly access to the monitored elements in the JVM.

4 Object Manager and Registration/Naming Service

The *object manager* is at the heart of distributed system, since it manages the object skeletons and object references. The object manager plays a role similar to that of an Object Request Broker (ORB) in the CORBA architecture, or the *registry service* in RMI. When a client asks for object, the object manager locates the skeleton for the object, to obtain a reference to the object and send it to the client. Remote method calls done by the client are routed through the object manager to the proper object on the server. The object manager also routes the result back to the client. The *registration/naming service* acts as an intermediary layer between the object client and the object manager. Once an interface to the object has been defined, an implementation of the interface needs to be registered with the naming service so that it can be accessed by clients. In order to use an object from a remote host, the client needs the naming service so it can identify the objects existing on the server side by their name. The main components of the J-OCM - the NDU and LMs - can be classified as an object manager and provide operations similar to the naming service that is present in distributed systems. Any object that can be observed or manipulated by tools is represented by a *token,* which is used in OMIS to provide a platform independent way of the object addressing. A token as a globally unique identifier refers to the proxy object, which allows to access to the real object in the target system. Before the tool can access an object defined in J-OMIS, the token and proxy object representing this object in the J-OCM must be created. When the JVMLM is started up, it notifies the node's LM of its existence in the monitoring infrastructure what causes the LM to create the token and proxy object, representing Java virtual machine, with which the JVMLM is associated. The other tokens representing *threads, classes, objects, interfaces, etc.* are created when:

- events that are related to those entities and the tool is interested in occur.
- the tool issues information requests with the following syntax: {`jvm`, `thread`, `class`, `etc.`}`_get_tokens()` to obtain a list of tokens of all monitored objects of a given class. These information requests extract information on the monitored objects known to JVMs which reside on the same node w.r.t. the Local Monitor.

The *token* data type solves the addressing problem between the tool and the monitoring system and provides access to the proxy object that stores information needed to correctly identify JVM entities. JVMPI, one of native interfaces

used by the JVMLM, uses unique identifiers to refer to such JVM entities as threads, classes, methods and objects. An identifier is assigned to an entity by the JVM during its *defining event* when all information associated with that identifier is sent. The identifier is valid until its *undefining event* arrives. For example, the defining event for a class identifier is `CLASS_LOAD`, sent when a class is loaded in the JVM while the corresponding `CLASS_UNLOAD` event is the class identifier's undefining event. A thread is identified by its JNIEnv interface pointer. Other entities are uniquely identified by their addresses in memory. Since the object may be relocated during garbage collection, JVMPI specifies that `MOVE_OBJECT` events invalidate the object identifier and assigns a new object identifier that is sent within event-specific data. Because all entity IDs are changed by the JVM during an event notification, the JVMLM maintains a mapping between JVMPI's IDs and the JVM agent's internal representation of monitored object identities, and update the mapping in response to the defining and undefining events for JVMPI IDs. The JVMLM's internal identifier of the monitored objects are also stored by the LM in the appropriate proxy object. Within the whole monitoring infrastructure, the monitored object reference is translated from a token through the JVMLM's identifier to JVMPI, JNI, JVMDI IDs.

5 Transport Layer

The distributed system consists of components that need to communicate with each other and transfer any data required for the remote method. The OCM incorporate the whole communication functionality into a separate module named INTERCOMM, for the inter-process/inter-node communication. Whereas the INTERCOMM for the OCM is based on PVM `send` and `receive` functions as well as UNIX-style sockets, the J-OCM introduces an additional communication component, called SHMLAE that was described in details in [1]. The SHMLAE as the communication layer, allows the components involved in communication to find each other during start-up and notify about its existence. When the agent is attaching to the environment, it sends a special message to the process that it is the manager of the SHMLAE. In the monitoring infrastructure the responsibility for management relies on the Local Monitor process. In order to fulfil the transportation function, the SHMLAE supports sending any data structures: simple data types and arbitrarily complex ones (lists, trees, ...), these can be packed and sent or received and unpacked. The user can register `pack` and `unpack` function for a particular message type and the runtime system automatically calls these functions before sending or after receiving the respective message denoted by the corresponding message tag.

6 Concluding Remarks

Separating the layer of tools from a monitoring system's functionality enables to build a versatile monitoring infrastructure supporting various kinds of tools. The

work on building a Java-oriented tools followed the idea of extending the On-line Monitoring Interface Specification by a specific hierarchy of objects and a set of relevant services. To deal with the target Java system we considered it in terms of the distributed system architecture which allows to separate the work on the definition of services on the tool side from their implementation on the server side provided by the monitoring system. Our on-going work focuses on completing the implementation of the J-OCM and designing a set of Java-oriented tools.

Acknowledgement. This research was carried out within the Polish-German collaboration and it was partially supported by the KBN grant 4 T11C 032 23.

References

1. M. Bubak, W. Funika, M.Smętek, Z. Kiliański, and R. Wismüller: Architecture of Monitoring System for Distributed Java Applications. In: Dongarra, J., Laforenza, D., Orlando, S. (Eds.), Proceedings of 10th European PVM/MPI Users' Group Meeting, Venice, Italy, September 29 - October 2, 2003, LNCS 2840, Springer, 2003
2. M. Bubak, W. Funika, B. Balis, and R. Wismüller. On-Line OCM-Based Tool Support for Parallel Applications. In: Yuen Chung Kwong, editor, Annual Review of Scalable Computing, volume 3, chapter 2, pages 32-62. World Scientific Publishing Co. and Singapore University Press, 2001.
3. M. Bubak, W. Funika, P. Mętel, R. Orłowski, and R. Wismüller: Towards a Monitoring Interface Specification for Distributed Java Applications. In Proc. 4th Int. Conf. PPAM 2001, Nałęczów, Poland, September 2001, LNCS 2328, pp. 315-322, Springer, 2002.
4. T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series, (1997)
   http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz
5. R. Wismüller, J. Trinitis and T. Ludwig: A Universal Infrastructure for the Runtime Monitoring of Parallel and Distributed Applications. In Euro-Par'98, Parallel Processing, volume 1470 of Lecture Notes in Computer Science, pages 173-180, Southampton, UK, September 1998. Springer-Verlag.
6. T. Lindholm and F. Yellin, *The Java Virtual Machine Specification,* Addison-Wesley Publishing Co., Reading, MA (1996).
7. Sun Microsystems: Java Virtual Machine Profiler Interface (JVMPI)
   http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html
8. Sun Microsystems: Java Virtual Machine Profiler Interface (JVMDI)
   http://java.sun.com/products/jdk/1.2/docs/guide/jvmdi/jvmdi.html
9. Sun Microsystems: Java Platform Debug Architecture (JPDA)
   http://java.sun.com/j2se/1.4.1/docs/guide/jpda/index.html
10. Sun Microsystems: Java Native Interface (JNI)
    http://java.sun.com/products/jdk/1.2/docs/guide/jni/
11. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Addison-Wesley, 1995.

Architecture and Implementation of Distributed Data Storage Using Web Services, CORBA and PVM*

Pawel Czarnul

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, Poland,
pczarnul@eti.pg.gda.pl, http://fox.eti.pg.gda.pl/~pczarnul

Abstract. We propose an architecture and its implementation called PVMWebCluster I/O targeted for distributed data storage and retrieval. Data is submitted via Web services from geographically distant clients or via CORBA from within clusters which offers better performance. The system consists of many, possibly geographically distributed clusters which run DAMPVM/PVM. Each cluster has a manager waiting for CORBA read/write calls which are passed to particular nodes in the cluster via PVM messages. Each cluster has a corresponding Web service that passes read/write calls to the CORBA manager. Web services form a top layer of PVMWebCluster I/O and call each other to obtain the best cluster/node to store a particular chunk of data. This results in a very flexible architecture which integrates distributed clusters for data storage of small and large files. The architecture proposes caching at both Web service and cluster layers. We have tested an initial implementation by submission of files of varying sizes in four configurations: via the Web service and CORBA layers on a local machine, via Web services in a LAN and via Web services through the Internet.

1 Introduction

In recent years, the need for parallel and distributed computing has increased significantly. This refers to both high performance computing within tightly coupled clusters using MPI and PVM ([1]), grid architectures ([2]) and Internet based multi-tier technologies ([3]) like J2EE, servlets, JSP etc. As high performance computing processes huge amounts of data, efficient means of handling it is necessary. We present the evolution from high performance parallel file server based architectures to highly distributed Web based data storage in XML. The proposed PVMWebCluster I/O uses Web services ([4]) and CORBA ([5]) as interfaces and PVM ([1]) within clusters.

2 Related Work

There are many systems in the literature targeted for parallel and distributed data storage/retrieval. However, they seem to be solutions either too focused on tightly coupled parallel computations like implementations of MPI-I/O ([6]) or are very general like:

* Work partially sponsored by the Polish National Grant KBN No. 4 T11C 005 25

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 360–367, 2004.
© Springer-Verlag Berlin Heidelberg 2004

WWW-based systems like Metacat ([7]) or OceanStore ([8]) or grid systems like EU-DataGrid ([9]) based on Globus ([10]) or GridLab ([11]). The latter ones are general solutions which integrate remote job control, data management and several other services. The proposed PVMWebCluster I/O is a dedicated solution for a collection of distributed PVM-based clusters so specific solutions like submission from within a cluster can be optimized considerably using the loads by other user processes, dynamic requirements reported by DAMPVM processes etc.

Network file systems like NFS or AFS ([12]) are easy to use within clusters as they support one uniform file tree for all the applications. However, integration of such systems in various clusters is not easy and requires additional tools.

MPI-I/O ([6]) is a set of parallel access operations to files defined within MPI-2 and thus limited to MPI. There are systems which support interoperability and file controlled sharing for applications running on different Massively Parallel Processors (MPPs). This enables different MPI vendor implementations to interoperate like in MPI_Connect ([13]). MPI_Conn_IO API allows file access from parallel applications running on different parallel computers. MPI_Conn_IO API is used to open a file globally and split it across the parallel sites, then possibly open it using the MPI-2 MPI_File_open() and update it. [14] proposes Stampi-I/O – a distributed parallel I/O library, an implementation of MPI-I/O that supports parallel file read/write operations. Another similar system is PACX-MPI PIO ([13]) in which many clients can access many parallel file servers. PIOUS ([15]) is a similar system for PVM which implements a virtual file system within a PVM environment and thus is a parallel not a distributed system by our definition.

The following systems support distributed file storage but their architectures do not seem to offer any specific optimization possibilities for PVM clusters with changeable loads and shared by many users at the same time as in the case of PVMWebCluster I/O. [16] presents the architecture of WebFS – a cache coherent distributed file system for unmodified applications which uses global HTTP naming to write and retrieve files. UFO ([17]) is another user level implementation of a distributed file system in which remote files can be treated as if they were local and are accessed using FTP or HTTP.

The following two systems are again too general and high-level in their architectures to be used efficiently in HPC applications on a collection of PVM-based clusters. As an example, OceanStore ([8]) has been developed with thousands of users and terabytes or more of data in mind. It is highly dispersed across the Internet and consists of distributed pools of storage systems each of which consists of particular servers. [7] presents Metacat – a framework for distributed data storage that is physically distributed across the Internet, possibly heterogeneous with respect to data format. Data is stored in XML in SQL-compliant relational databases.

Regarding the latest developments, there are grid-based systems like EU-DataGrid ([9]) and GridLab ([11]). However, since the specifications focus rather on the requirements, APIs, portability and ease of integration of various systems, there is no specific mention of PVM/MPI optimizations as the systems focus on high-level distributed job control of large applications instead. The Global Access to Secondary Storage system (GASS, [18]), a part of the Globus toolkit ([10]) used in grid-based systems, is a data access and movement service that uses URLs to implement a global file space. GASS provides cache techniques for read and write operations.

3 PVMWebCluster I/O Architecture

PVMWebCluster I/O is based on the three-tier PVMWebCluster architecture which corresponds to the following three layers in PVMWebCluster I/O:

1. Web Service Data Submission Layer (WSDSL) – a geographically distributed PVMWebCluster system is composed of particular clusters each of which has a corresponding Web Service interface. Web Services representing separate clusters call each other to determine the best cluster to store a particular chunk of data.
2. Cluster Data Submission Layer (CDSL) – each cluster has a CORBA representative that intermediates incoming data submission calls to the cluster as well as returns information such as: available storage space, processor speeds, internode latency and bandwidth, number of active processes on particular nodes etc. This information can be used by the WSDSL to make a decision in which cluster to store a particular chunk of data. The cluster manager listens to CORBA calls and uses PVM communication to store the data on one of the available nodes in the cluster. The CDSL has a database of files submitted to it. Such submission is faster but data can be partitioned and spread using only the cluster nodes. On the other hand, using the WSDSL, data can be replicated and stored on physically distant clusters in case one cluster is damaged.
3. Cluster Layer (CL) – finally a chunk of data is stored on a certain node in the cluster. The data is submitted from the CDSL, currently implemented as PVM messages.

The proposed system architecture of PVMWebCluster with distributed file storage PVMWebCluster I/O is presented in Figure 1. We distinguish the following data submission modes for both distributed and sequential processing:

1. Large data submission through Web Services – used to store large amounts of data which does not require frequent and fast access:
   - this solution involves large latency both when data is submitted and retrieved,
   - allows really large data capacity as the system architecture includes a set of clusters each of which consists of many nodes equipped with disks.
2. Storage for high performance computing within clusters – reasonably small amounts of data which is to be accessed reasonably frequently. In the initial implementation, we assumed that in this submission mode files would not need partitioning. However, it may be useful to replicate files across the nodes in the cluster so that processes can access them in parallel.

4 Data Access Patterns, Submission and Partitioning in PVMWebCluster I/O

WebFS implements three cache coherence policies:

1. Last writer wins – a server keeps a listing of all sites caching the given file. If it has been updated by one of them, invalidation notifications are sent to the others. This makes this policy reasonable for occasional updates rather than frequent updates of many files.

2. Append only – writes append information to files and can be simply forwarded to other servers and receivers.
3. Multicast – all updates are sent to all clients. One dedicated channel is used to distribute invalidation (update) notifications while another one for sending updates.

In GASS ([18]), common access patterns have been distinguished and implemented to achieve low latency access to files as well as high bandwidth. They include: read-only, last writer wins, append-only and unrestricted access to remove the need for parallel call synchronization.

PVMWebCluster I/O is more about distributed file storage i.e. file partitioning and distribution rather than multi-user access. In PVMWebCluster I/O, we assume that a file is generally used by the user who has written it. However, extensions are possible in the future thanks to the flexible architecture. As in OceanStore ([8]), we assume that any update of a file is a new version of it and is stored as a new file. It essentially eliminates the need for concurrent write control for files. The URL global space is naturally used as the implementation is based on Web services. There is a dedicated directory for data storage on each node.

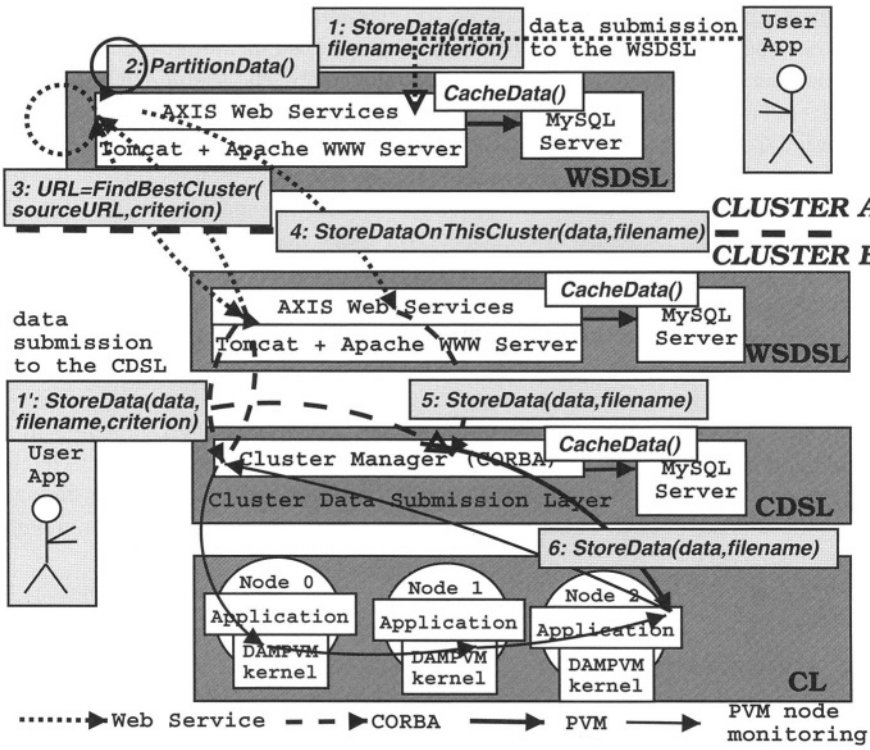


Fig. 1. PVMWebCluster I/O Architecture

The write submission requests to both the WSDSL and the CDSL are shown in Figure 1. For the WSDSL the steps are as follows:

1. A write call is invoked in the client code. Assuming the storage of data in a string, the client code invokes Web service `public static String StoreData (String sData, String sFileName, int nCriterion)`. The data is distributed among the available clusters. The information about all the blocks, their numbers and locations is stored in a MySQL database. In the experiments, single data submissions were not partitioned across nodes. In this case, the Web service returns the URL of the final Web service, the node within the cluster where the data has been saved and the available free space (in KBs) on this node (`<BestService URL>>><BestNodeName>>><result>e.g. wolf>>.eti.pg.gda.pl>>wolf 10773.836`). The criterion given as an argument determines the algorithm used. The advantage of the Web Service technology is the simplicity of the client code.
2. The Web service decides whether the data size is too large and thus should be partitioned into chunks. If this is the case, the data is partitioned by method `Partition Data()`. Then the following operations are executed on the chunks in parallel.
3. For a data chunk, invoke method `public static string FindBestcluster (String sSourceServiceURL, int nCriterion)` which finds the best (with respect to the criterion set, in this case `FILE_STORAGE_AVAILABLE_MAXIMIZATTON_CRITERION` since we are looking for maximum available disk space) cluster in the subgraph of the system. The clusters being called memorize the URL and the parameters of the best Web service/node found so far. The source URL is given in order not to call it back recursively. The cluster which returns the maximum available storage space is assigned the data. On every cluster method `FindBestCluster()` invokes method `GetClusterMaxFreeFileStorageSpace()` which returns the file storage (corresponding to the node with the maximum free space in the cluster). This method uses the available storage space for every node within the cluster that is cached at the WSDSL layer. It is gathered by another thread working concurrently which makes CORBA calls to the CDSL layer cluster manager in parallel.
4. Invoke Web service `public static String StoreDataOnThisCluster (String sData, String sFileName)` which passes the data to the optimal cluster selected in the previous step.
5. On the final cluster, a call is made to the `StoreData()` CORBA method in the CDSL layer cluster manager. Its Interoperable Object Reference (IOR) is fetched from a file written to the disk during the initialization of the server. The file name and the data are passed. This layer can also be called by an application from within the cluster via CORBA (1'. in Figure 1).
6. Finally the CORBA manager stores the data on the best node in the cluster i.e. the node with maximum available free storage space for the aforementioned criterion and sends the data via PVM messages.

Figure 1 shows the cache procedures (currently being implemented) at the WSDSL and CDSL levels. Subsequent read operations can use the cached data. The cache at the CDSL level can contain more information than the WSDSL cache as some files may have been submitted to the cluster manager via CORBA calls from within the cluster.

5 Implementation and Experimental Results

Entry points to clusters have been implemented as Web services in Java with the AXIS server (published as `.jws` files, [4]) running in the Tomcat application server ([19]). AXIS is a SOAP engine and Tomcat runs on the Apache WWW server. The architecture allows PVM clusters running on different user accounts.

We have implemented read and write operations through Web services (implemented in Java) and then via CORBA calls to particular clusters. Cluster managers (implemented in C++) monitor performance parameters including available storage space from all the nodes in the clusters they are in charge of. This is done by DAMPVM kernels ([20], [21], [22]), previously developed by the author. They use PVM communication and OS system calls. The following four configurations were tested:

CORBA – local machine – data is written to a local machine via a CORBA call. A Pentium 4-M 1.4GHz workstation running Redhat Linux 8.0, kernel 2.4.18-18.

Web service – local machine – data is written to a local machine through a Web service (the WSDSL level). The WSDSL layer contacts the cluster manager at the CDSL level which writes the data in the cluster. Same configuration as above.

Web service – through LAN – data is written to a machine in a local network through a Web service (the WSDSL level) which invokes Web services on other nodes to determine the cluster with the largest available storage space. Then a Web service is invoked on the chosen cluster which contacts the cluster manager at the CDSL level which writes the data in the cluster. Pentium 4-M 1.4GHz and Athlon XPs 1800+ workstations running Redhat Linux 8.0, kernel 2.4.18-18, 10Mbps Ethernet.

Web service – through Internet – data is written to a distant machine through the Internet via a Web service (the WSDSL level). The communication is done as in the previous configuration with the exception of much larger latency and lower bandwidth through the Internet. Same configuration as above through the Internet (one node connected to the Internet via a shared 115kbps SDI connection).

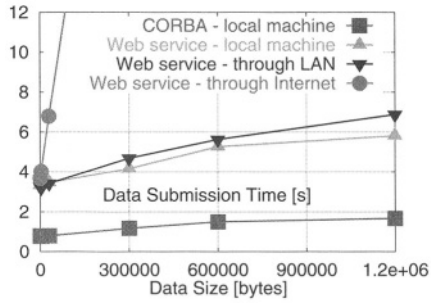


Fig. 2. Write Times with Client Overhead

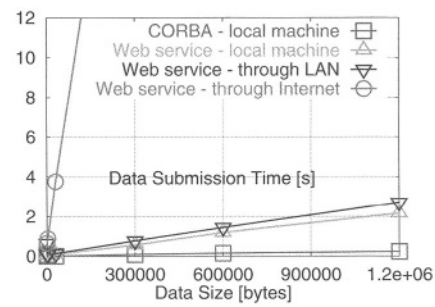


Fig. 3. Write Times without Client Overhead

It is must be noted that data was submitted through Java clients that read it and passed to the WSDSL or CDSL layers respectively. Figures 2 and 3 show the obtained

results for submission of files of the following sizes: 30, 300, 3000, 300000, 600000 and 1200000 bytes. Figure 2 shows the times of single write calls via the client in which case we must account for the client initialization as well. This corresponds to occasional write calls when the initialization overhead is significant. The results are averaged from 10 write calls. Figure 3 shows the write times of successive write calls after the client has already been initialized and the first call is already finished. The results are averaged from 100 subsequent calls without client initialization. This corresponds to many subsequent submissions for which the initialization overhead can be neglected.

Figure 2 shows that the overhead for invoking a client and thus the initialization of necessary Web service and CORBA components is considerable, even for small 30-byte files. We also conclude that this initial overhead is much smaller when data is submitted within the cluster through the CORBA manager than through Web services. Additionally, Figure 3 shows that the Web service implementation puts a limit on the bandwidth even on a local machine compared to the CORBA communication. However, it is also shown that in practice this may not be a determining factor as the low Internet bandwidth is the bottleneck for distributed data submission.

6 Summary and Future Work

We have proposed and implemented a system for distributed data storage and retrieval based on Web services as the top layer, CORBA as middleware and PVM inside clusters. The performance of the implementation for files of varying sizes have been assessed for submissions via both the Web service and CORBA layers. Although the system is fully functional and complements PVMWebCluster in distributed task execution and management, there are many features to be implemented in PVMWebCluster I/O:

- various partitioning and caching techniques for very large files to be stored, tuning parameters and performance measurement for large networks,
- data encryption and decryption for distributed data storage in open environments,
- integration with the PVMWebCluster user management,
- data replication and migration to increase bandwidth,
- node/cluster/network failure handling.

References

1. Wilkinson, B., Allen, M.: Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Prentice Hall (1999)
2. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications **15** (2001) 200–222 http://www.globus.org/research/papers/anatomy.pdf.
3. Noack, J., Mehmaneche, H., Mehmaneche, H., Zendler, A.: Architectural Patterns for Web Applications. In Hamza, M., ed.: 18th IASTED International Conference on Applied Informatics (AI 2000), Proceedings, Innsbruck, Austria, ACTA Press (2000) citeseer.nj.nec.com/260788.html.
4. Streicher, M.: Creating Web Services with AXIS: Apache's Latest SOAP Implementation Bootstraps Web Services. Linux Magazine (2002) http://www.linux- mag.com/2002-08/axis_01.html.

5. Buyya, R., ed.: High Performance Cluster Computing, Programming and Applications. Prentice Hall (1999)
6. Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface Standard. (1997)
7. Jones, M., Berkley, C., Bojilova, J., Schildhauer, M.: Managing Scientific Metadata. IEEE Internet Computing **5** (2001) 59–68
8. Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Weatherspoon, H., Kubiatowicz, J.: Maintenance-Free Global Data Storage. IEEE Internet Computing **5** (2001) 40–49
9. EU-DataGrid (EDG): The DataGrid Project (2003) `http://eu-datagrid.web.cern.ch/eu-datagrid`.
10. Globus: Fundamental Technologies Needed to Build Computational Grids (2003) `http://www.globus.org`.
11. GridLab: A Grid Application Toolkit and Testbed (2003) `http://www.gridlab.org`.
12. Coulouris, G., Dollimore, J., Kindberg, T: Distributed Systems – Concepts and Design. Addison-Wesley (2001)
13. Fagg, G.E., Gabriel, E., Resch, M., Dongarra, J.J.: Parallel IO Support for Meta-computing Applications: MPI_Connect IO Applied to PACX-MPI. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2131 in Lecture Notes in Computer Science, Springer-Verlag (2001) 135–147 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001, Proceedings.
14. Tsujita, Y., Imamura, T., Takemiya, H., Yamagishi, N.: Stampi-I/O: A Flexible Parallel-I/O Library for Heterogeneous Computing Environment. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2474 in Lecture Notes in Computer Science, Springer-Verlag (2002) 288–295 9th European PVM/MPI Users' Group Meeting, Linz, Austria, September/October, 2002, Proceedings.
15. Sunderam, V., Moyer, S.: PIOUS for PVM (1995) `http://www.mathcs.emory. edu/pious`.
16. Vahdat, A.M., Eastham, P.C., Anderson, T.E.: WebFS: A Global Cache Coherent File System. Technical report, Computer Science Division, University of California Berkeley (1996) `http://www.cs.duke.edu/~vahdat/webfs/webfs.html`.
17. Alexandrov, A.D., Ibel, M., Schauser, K.E., Scheiman, C.J.: Extending the Operating System at the User Level: the Ufo Global File System. In: Proceedings of the USENIX Annual Technical Conference, Anaheim, California, USA (1997) 77–90
18. Bester, J., Foster, I., Kesselman, C., Tedesco, J., Tuecke, S.: GASS: A Data Movement and Access Service for Wide Area Computing Systems. In: Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems, Atlanta, GA, ACM Press (1999) 78–88
19. McClanahan, C.R.: Tomcat: Application Developer's Guide. (2002) Apache Jakarta Project, `http://jakarta.apache.org/tomcat/tomcat-4.1-doc/appdev/index. html`.
20. Czarnul, P.: Programming, Tuning and Automatic Parallelization of Irregular Divide-and-Conquer Applications in DAMPVM/DAC. International Journal of High Performance Computing Applications **17** (2003) 77–93
21. Czarnul, P., Tomko, K., Krawczyk, H.: Dynamic Partitioning of the Divide-and-Conquer Scheme with Migration in PVM Environment. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2131 in Lecture Notes in Computer Science, Springer-Verlag (2001) 174–182 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001, Proceedings.
22. Czarnul, P., Krawczyk, H.: Dynamic Assignment with Process Migration in Distributed Environments. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 1697 in Lecture Notes in Computer Science (1999) 509–516

Online Visualization of OpenMP Programs in the DeWiz Environment

Rene Kobler, Dieter Kranzlmüller, and Jens Volkert

GUP Linz, Johannes Kepler University Linz,
Altenbergerstr. 69, A-4040 Linz, Austria/Europe,
rkobler@gup.uni-linz.ac.at,
http://www.gup.uni-linz.ac.at/

Abstract. This papers describes an approach for visualizing the execution of OpenMP programs using a modular and distributed approach. The visualization as well as the execution are integrated into the DeWiz (Debugging Wizard) tool. DeWiz performs debugging activities on the notion of the event graph and divides these activities into different modules. The event graph model is used to describe state changes in programs. We apply this event graph model on OpenMP programs, which establish critical regions using "omp_set_lock" and "omp_unset_lock" operations, to generate a visual representation of the program execution. The OpenMP Program as well as the visualization of the resulting event graph run as separate modules in the DeWiz environment.

1 Introduction

The Grid becomes a more and more important factor in parallel software engineering. As the Grid offers transparent access to high performance computing architectures and simplifies the usage of distributed computing resources, it is essential to provide appropriate tools for debugging Grid applications. DeWiz [5] is such a tool. Its basic idea is to split debugging activities into a set of independent modules, which operate on event graphs. According to the desired tasks, modules can be arranged at will. Data for program analysis are provided by arbitrary monitoring modules which can be placed in a Grid infrastructure. This paper describes the usage of DeWiz for OpenMP programs.

OpenMP has grown up to a quasi-standard for programming shared-memory architectures. OpenMP bindings are currently available for Fortran and C/C++, respectively. Compared to MPI parallelism, OpenMP is not using processes but threads, which communicate via shared memory. Therefore it is well suited for parallel programming on SMP clusters. OpenMP expresses parallelism with the help of directives, library functions and environment variables. So-called *parallel* directives indicate that a certain part of code is executed by a certain number of threads while inside this code special OpenMP-library-functions can be called for i.e. thread synchronization via critical regions (for a full description of OpenMP functionality please refer to [9] and [10]).

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 368–374, 2004.
© Springer-Verlag Berlin Heidelberg 2004

In this paper we show a practical approach for monitoring the execution of an OpenMP module based on "omp_set_lock" and " omp_unset_lock" operations for establishing critical regions using the POMP API [7], which can be located anywhere in a dedicated Grid infrastructure. The monitored data is sent online using functionality of the Grid middleware. Visualization is achieved by a special visualization module which is implemented as a Java Applet, so that the resulting event graph can be inspected e.g. by using a standard web browser.

The paper is organized as follows. Section 2 gives basic information about program states, events and relations between them in the DeWiz environment and shows how DeWiz modules interoperate while Section 3 describes the monitoring and visualization of OpenMP programs by means of a special example. Section 4 concludes and gives a prospective for further activities in this area.

2 The Basics of DeWiz

Essential for program analysis are state changes that transform a given input through statements defined by the program code into the output. In most cases it is not required to store all possible intermediate states of a program. Therefore it is important to select an interesting subset of states that are relevant during debugging. Many analysis tools as well as DeWiz use the notion of an event, which is defined as the instant, when a state change occurs [11]:

Definition 1. *An event e_p^i is defined as an action without duration that takes place at a specific point in time i and changes the state of a process/thread p.*

Based on Lamports "happened-before" relation [6] and the definition of an event, DeWiz describes parallel program behavior applying the event graph model [2]:

Definition 2. *An event graph is a directed graph $G = (E, \rightarrow)$, where E is the non-empty set of events e_p^i of G, while $\rightarrow$ is the "happened before"-relation connecting events, such that $e_p^i \rightarrow e_q^j$ means, that there is an edge from event e_p^i to event e_q^j in G with the "tail" at event e_p^i and the "head" at event e_q^j.*

A more formal description of states, events and relations between them can be found in [4].

As observation and analysis of programs in the DeWiz environment is completely based on the event graph model, we have to map events as well as "happened-before" relations to some data structures which can be investigated by the corresponding DeWiz modules. For events e_p^i (where i denotes the timestamp of occurrence on a particular thread p) we use the structure $(p, i, type, data)$, while "happened-before" relations $e_p^i \rightarrow e_q^j$ are mapped to (p, i, q, j). Each kind of event is uniquely identified by field *type* of the event structure. The content of *data* will not be specified here, since it depends on the type of event. To illustrate the field data, consider message- passing systems in which we may have send and receive operations used for event types, then the

event data could be the message itself, the message size or some other interesting characteristica of the observed operations.

As mentioned before, in our target OpenMP programs we want to analyze "omp_set_lock" and "omp_unset_lock" operations, the corresponding functions for setting up a critical region. Therefore events for DeWiz can be constituted as follows:

- set event: e_p^i = (p, i, omp_set, semaphore variable)
- unset event: e_p^i = (p, i, omp_unset, semaphore variable)

As noted above DeWiz consists of modules which interoperate in some user-defined way. The data structures defined above are the basic data items processed by DeWiz modules. Each module receives a stream of events and relations and performs its dedicated tasks on it. It is possible to set up more than one input interface as well as more than one output interface for a module. In the DeWiz system it is important that one dedicated module takes on the registration of modules in the system, the *Sentinel.* The Sentinel coordinates all modules. If one module wants to communicate with another module in the system, the sentinel acts as a broker to establish a communication channel between them. Specific control messages are used to communicate with the sentinel. Modules written in different programming languages or running on different operating systems can communicate among each other. Therefore it' s crucial to specify an exact byte-wise structure for data stream items as well as for control messages.

3 Visualizing an OpenMP Program in DeWiz

Based on the short overview of the DeWiz system above, we now dwell a little bit more detailly on a concrete example running on this system. Figure 1 gives an overview of the required modules in the DeWiz system to perform the visualization of the OpenMP program execution, additionally the arrows indicate the registration process of a module. The module "Controller" acts like a visual representation of the sentinel. It is used e.g. for establishing communication channels between modules or to eliminate a module from the system.

In our example the OpenMP target is a standard OpenMP program written in C. The following code fragment shows the basic functionality of our OpenMP module:

```
#define SET(s,v) { omp_set_lock(&s);v++; omp_unset_lock(&s); }
...
#pragma omp parallel
{
    if (omp_get_thread_num() == 0) SET(s2,v2);
    else if (omp_get_thread_num() == 1) {
              SET(s1,v1); SET(s2,v2);
              SET(s1,v1);
        }
```

```
    else if (omp_get_thread_num() == 2) SET(s1,v1);
    else if (omp_get_thread_num() == 3) {
              SET(s1,v1); SET(s2,v2);
        }
    else if (omp_get_thread_num() == 4) {
              SET(s1,v1); SET(s2,v2);
              SET(s1,v1); SET(s2,v2);
        }
}
...
```

This OpenMP program will be executed by 5 threads which create critical regions using two different semaphores *s1* and *s2*. Inside these critical regions a variable *v1* and *v2* is increased, respectively. To observe the program's behavior we have to instrument and monitor it during its execution. The instrumentation is done using the OPARI source-to-source translation tool, which is part of the POMP performance tool interface [7]. POMP as well as OMPI (OpenMP Instrumentation Interface) are part of an effort to define a general performance monitoring interface for OpenMP programs submitted to the OpenMP Architecture Review Board [8].

OPARI instrumentates the OpenMP program by replacing or inserting code. The POMP library initially provides empty interfaces of monitoring functions inserted by OPARI. To monitor the program new code has to be added.

We observe the execution of "omp_set_lock" as well as "omp_unset_lock" functions in our example OpenMP program, so we have to insert code into the corresponding POMP monitoring routines "pomp_set_lock" and "pomp_unset_lock". In these functions DeWiz events and happend-before relations must be generated (see Section 2) and sent to a so called consuming module, in our case the Visualization applet. This applet receives the previously generated event graph stream and visualizes it in an appropriate way.

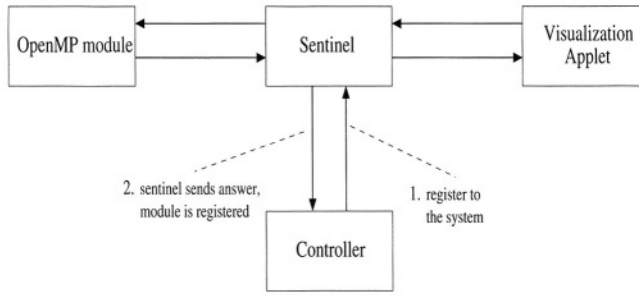


Fig. 1. Registration of modules in the DeWiz system

Before this event graph stream generation and visualization can be carried out, the involved modules must register to the system (see Figure 1). This regis-

tration process is handled in two steps via control messages. In the first step the module to be registered sends a control message to the sentinel. In the second step the module waits for a control message, which indicates that the module is now registered to the system. All modules receive a unique ID within the DeWiz system during the registration process. The registration is done via a special framework, which is currently available in Java and C. Afterwards a communication channel between the OpenMP module and the visualization applet is established using the Controller module (Figure 2 indicates this procedure). The controller also displays the modules and possibly existing connections within the DeWiz system (see Figure 3). When the communication channel exists, the OpenMP module sends the event graph stream during its execution using the before received connection data (from the controller).

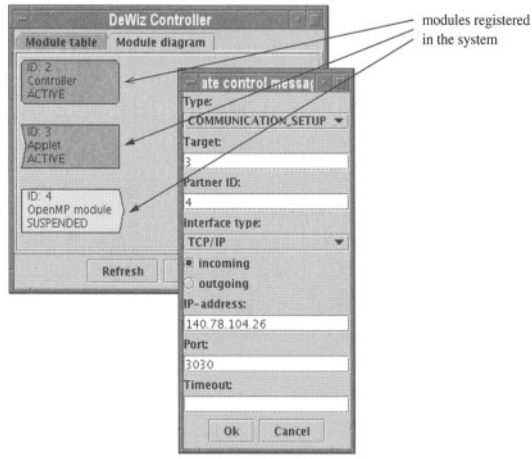


Fig. 2. Establishing a communication channel via control message dialog using the DeWiz Controller

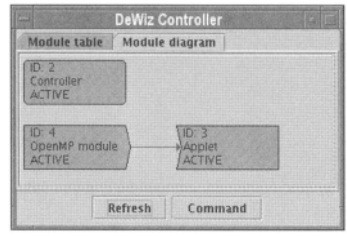


Fig. 3. DeWiz Controller after establishing a communication channel between OpenMP module and visualization applet

The event graph stream is drawn on-the-fly by the attached visualization applet. Program behavior which is not intended can be detected immediately at

visualization side. Figure 4 shows a picture of the visualization applet containing the event graph of a possible execution of our example OpenMP program. The small circles indicate set and unset events on the particular threads, the arrows indicate "happend-before" relations. More event information is provided by doubleclicking on a particular event or "happend-before" relation. "Happend-before" relations are only drawn, when the semaphore locking thread changes.

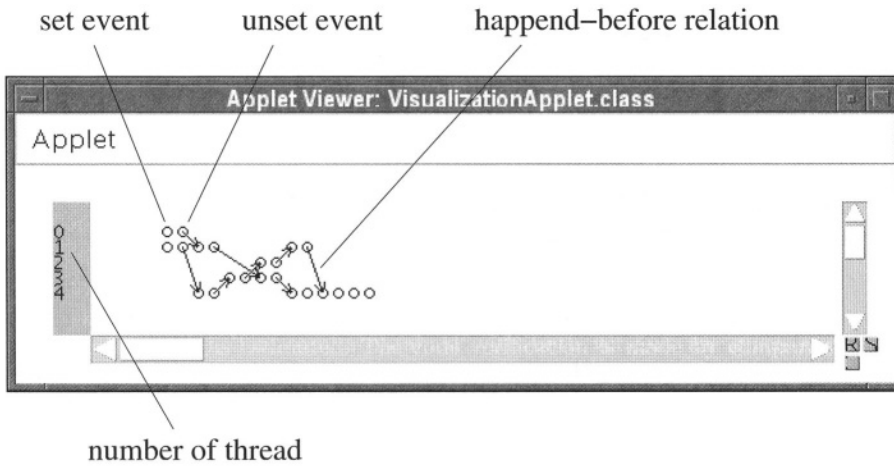


Fig. 4. Event graph display in the applet

It's also conceivable to perform certain operations on the event graph, i.e. the event manipulation technique for testing a nondeterministic OpenMP program (described in [1] and [3]) using the DeWiz framework, although the program may run anywhere else. Our example OpenMP program is such a nondeterministic program, since the involved threads try to lock semaphores to establish critical regions. It depends on the relative speed of the involved threads which semaphore locking order results. Therefore different event graphs may surrender. It is not guaranteed to obtain all feasible executions of such a program, no matter how many executions are initiated. The event manipulation technique deals with the question "What would have happened, if the nondeterministic choices would have been different from what has been observed?" To simplify this, through event manipulation it's possible to generate all feasible executions of a nondeterministic program.

4 Conclusions and Future Work

The DeWiz tool offers an interesting alternative for program visualization. It is possible to run the OpenMP program on a cluster i.e. using a compiler which distributes OpenMP threads among cluster nodes (i.e. Omni OpenMP compiler [12]). The result can be viewed on-the-fly on a completely other PC under

the precondition that the dedicated visualization applet has registered to the DeWiz system. The modularity of DeWiz and its flexible communication protocol (on top of TCP/IP) make the tool an ideal candidate for a grid-enabled debugging service. By placing DeWiz modules on Grid resources, high performance computing power can be used for program analysis activities. This application of DeWiz is currently being investigated for the next version of the program analysis tool using the Globus environment.

Acknowledgements. Contributions to this work have been made by several people, most notable Michael Scarpa, Johannes Hölzl and Bernhard Aichinger.

References

1. Kobler, R., Kranzlmüller, D., Volkert, J., "Debugging OpenMP Programs using Event Manipulation", Proc. 2nd International Workshop on OpenMP Applications and Tools (WOMPAT 2001), West Lafayette, IN, USA, (Jul. 2001).
2. Kranzlmüller, D., "Event Graph Analysis for Debugging Massively Parallel Programs", PhD Thesis, GUP Linz, Joh. Kepler University Linz, http://www.gup.uni-linz.ac.at/~dk/thesis (Sept. 2000).
3. Kranzlmüller, D., Kobler, R., Volkert, J., "Event Manipulation for Nondeterminstic Shared Memory Programs", Proc. HPCN Europe 2001, 9th Intl. Conference, Amsterdam, The Netherlands, pp. 283 - 292 (Jun. 2001).
4. Kranzlmüller, D., "DeWiz - Event-based Debugging on the Grid", Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP'02), Las Palmas de Gran Canaria, Canary Islands, Spain, (Jan. 2002).
5. Kranzlmüller, D., Volkert, J., "Ubiquitous Parallel Program Observation and Analysis with DeWiz", Proc. PASA 2002 (6. Workshop Parallele Systeme und Algorithmen), Karlsruhe, Germany, (Apr. 2002).
6. Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM, pp. 558 - 565 (Jul. 1978).
7. Mohr, B., Malony, A., Shende, S., Wolf, F. , "Design and Prototype of a Performance Tool Interface for OpenMP", Proc. of the 2nd Annual Los Alamos Computer Science Institute Symposium (LACSI 2001), Santa Fe, NM, USA, (Oct. 2001).
8. Mohr, B., Mallony, A., Hoppe, H.-C., Schlimbach, F., Haab, G., Hoeflinger, J. and Shah. S., "A Performance Monitoring Interface for OpenMP", 4th European Workshop on OpenMP (EWOMP'02), Rome, Italy, (Sept. 2002).
9. OpenMP Architecture Review Board. "OpenMP C/C++ Specification Version 2.0", Technical Report, http://www.openmp.org/specs/mp-documents/cspec20.pdf, (Mar. 2002).
10. OpenMP Architecture Review Board. "OpenMP Fortran Specification Version 2.0", Technical Report, http://www.openmp.org/specs/mp-documents/fspec20.pdf, (Nov. 2000).
11. van Rick, M., Tourancheau, B., "The Design of the General Parallel Monitoring System", Programming Environments for Parallel Computing, IFIP, North Holland, pp. 127-137 (1992).
12. Sato, M., Satoh, S., Kusano, K., Tanaka, Y., "Design of OpenMP Compiler for an SMP Cluster", Proc. of the First European Workshop on OpenMP (EWOMP 1999), Lund, Sweden, Sept. 1999.

Cluster Monitoring and Management in the WebCI Environment

Tomasz Kuczynski, Roman Wyrzykowski, and Grzegorz Studzinski

Institute of Computer and Information Sciences,
Czestochowa University of Technology
{tkucz,roman}@icis.pcz.pl, whitei@op.pl

Abstract. In this paper, we present the design and implementation of the WebCI (Web Condor Interface) tool for multiple Condor pools management and monitoring. Our project focuses on communication and persistence, the most important aspects of web interfaces for cluster management. Thanks to the usage of standard tools like SCP and SSH, WebCI distinguish itself from other available cluster web interfaces because of seamless installation of new resources. In addition, WebCI aids users in job preparation, files exchange between client's machine and resource, and other tasks. It also allows to execute any batch command on managed resources, that gives large functionality.

1 Introduction

The fast development of the processing power of high–end PCs together with the availability of open source software such as Linux have made it possible to build very cost–efficient parallel computers [2]. With the addition of high bandwidth and low latency local networks, PC–clusters are the most common and available parallel system now.

An example of such a cluster is ACCORD – Academic Cluster of Czestochowa fOr Research and eDucation built in the Institute of Computer and Information Sciences, Czestochowa University of Technology [8].

The first version of the ACCORD cluster was built on January 2001, as the second high–performance cluster in Poland [4] [5]. It contains 18 Pentium III 750 MHz processors, or 9 server platforms ISP 2150G as SMP nodes. Each node has 512 MB RAM, except for the front–end node with 1 GB RAM. This installation consists also of 150 GB disk memories (SCSI standard). For tightly coupled distributed computations, the interconnect must provide high–data–rate and low–latency communication between nodes. That is why, the Myrinet technology is used in the cluster, for the first time in Poland.

A performance–and–cost–effective alternative to Intel processor is Athlon AMD architecture with its high–effecient multiprocessing support for data movement–intensive applications. That is why, the ACCORD cluster was firstly upgraded with 16 AMD Athlon MP 1.2 GHz processors, and secondly with 16 AMD Athlon MP 2100+ processors. It was the first high–performance cluster in

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 375–382, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Poland based on AMD processors. At this moment, the new nodes are connected by Fast Ethernet. ACCORD is operating under the control of the LINUX system (Debian distribution) and Condor workload management system.

Amongst free accessible software for batch systems utilized in clusters, the Condor workload management system is one of the most popular [3]. Like other full–featured batch systems, Condor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Unfortunately, like other batch systems, Condor has only a text interface. Since text interfaces are not user–friendly, we decided to create a web tool that allows for full and easy access to the Condor functionality. Assumption of access from as many platforms as possible led us to the use of a thin client. Requirements which should be fullfilled by a platform are really low: a web browser with https protocol, JavaScript (JScript) and http cookie support.

The WebCI tool is not the only web interface that was created for a batch system. PBSWeb developed at the University of Alberta [7] is an example of system created for supporting the PBS batch system [6]. As in case of WebCI, much of the work done by PBSWeb for a user is executed through the user account, using SSH. In order for a user to allow PBSWeb for access to its account, the user must place the PBSWeb public key into its authorized_keys file. This allows PBSWeb to run commands.

PBSWeb gives similar functionality as WebCI, but there are two big differences. PBSWeb disadvantages, eliminated in WebCI, are given below:

- PBSWeb installation is not seamless: it is necessary to place the PBSWeb public key into user's authorized_keys file (every user has to do it);
- PBSWeb does not provide the necessary security: authorization solution allows to break into any user's account if someone breaks into the web server.

2 Project Goals and Features

The original goal of the WebCI project is development of a tool that allows monitoring and management of a Condor pool using WWW. The main pressure is put on ease of job submition and control, as well as convenient access to UNIX shell. The key element of the project is the portal security and platform independence. These requirements constrain us to use only the standard system tools. All the above leads to a concept of using SSH sessions and scp tool through pseudo terminals.The usage of SSH and SCP enables us to separate the portal from the access node of the pool. This in turn allows for adding the functionality of multiple Condor pools interaction and monitoring.

Not without importance is the ability of seamless attachment of new Condor pools by simple addition of a domain or IP address of the access node to the WebCI config file. Every pool may be accessed by an unrestricted number of portals, allowing for removing a single point of failure and increasing the system

stability. The use of mainly server–side technologies allows to use a thin client, and develop WAPCI, in the future. This will provide the full support for mobile devices.

WebCI has capability of monitoring public resources and queues that allows to obtain information about any of running jobs, and accessible resources (see Fig.1). A user does not have to be logged–in to check if his jobs are running correctly, or if there are any resources that can be used to run more jobs. Obviously this feature is fully configurable, and can be turned off if some security reasons disallow public monitoring.

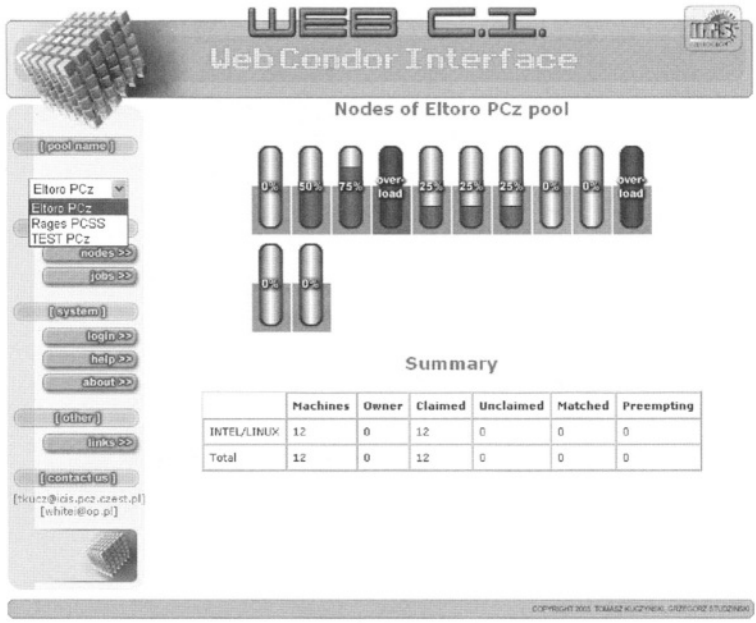


Fig. 1. Monitoring public resources

It is possible to easily adapt the WebCI system architecture to Grid structures, thus creating a secure and efficient WWW interface. Among other tasks, this interface will enable monitoring of resources and job queues, job submition, preparation (see Fig2) and management, exchange of files between a web browser and a user account, management of files and directories on users' accounts. The important advantage of the WebCI Grid portal will be convenient use of shell commands using a tool similar to the Midnight Commander. Also the job repository and possibility of remote file edition is very helpful feature.

At the time of writing this paper, WebCI allows us to monitor and manage multiple Condor pools and queues, also in case of a submit node without a pub-

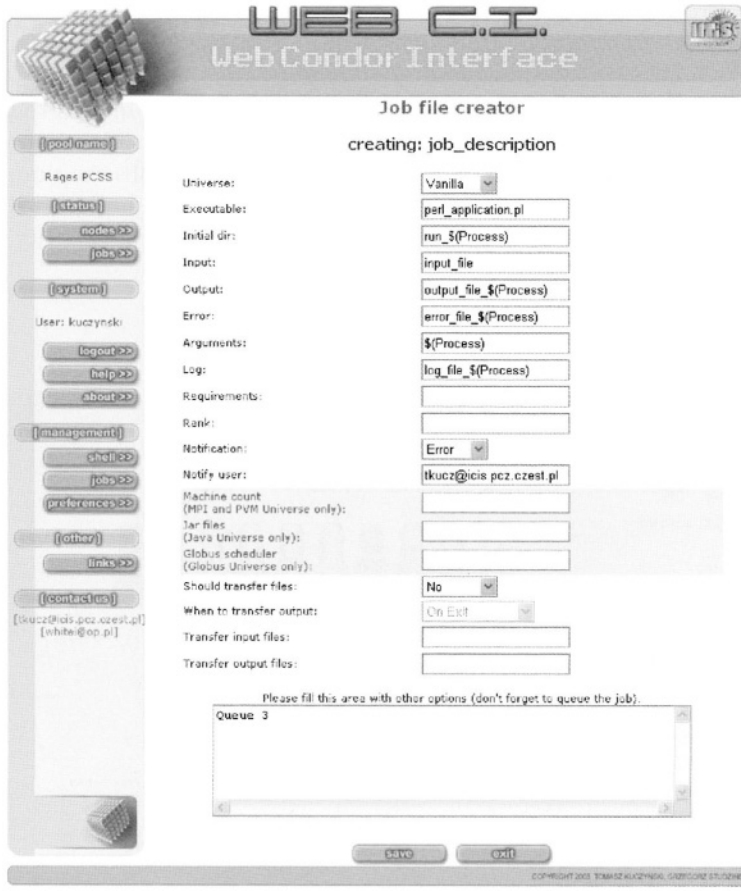


Fig. 2. Support of job preparation

lic IP address. All capabilities described above has been already implemented. The WebCI tool provides GUI support for the following Condor commands: *condor_submit, condor_rm, condor_hold, condor_release, condor_prio, condor_q, condor_Status.*

The supported system commands include: *ls, cd, pwd, cp, mv, rm, mkdir.* Any other batch commands are supported by WebCI by simply displaying their output in the web browser.

3 Design and Implementation

3.1 Architecture

The WebCI software consists of three layers (Fig.3). Each of them is executed on different resources.

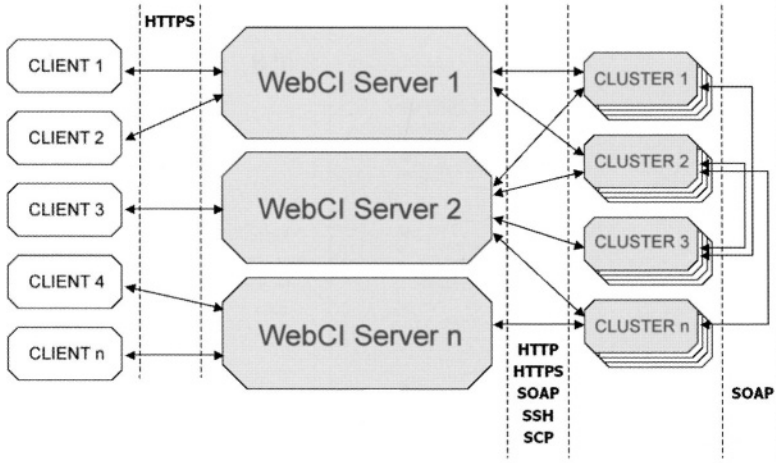


Fig. 3. Three layer architecture of WebCI software

The highest layer is WWW browser. To use WebCI features, the browser must accept cookies, and has to be able to interpret JavaScript scripts. Moreover, for security reasons it must support the https protocol. In nowadays most web browsers match all the above requirements.

The second layer – set of CGI scripts – is the most important part of WebCI. For non–logged–in users, these scripts act as proxy which fetch data from the third layer, and prepare them for presentation in the first layer as web pages. For logged–in users, the scripts are wrapper for SSH sessions (Fig.4) and scp tool; this is the most interesting part of that layer and the whole system. This solution is very flexible, it allows for gathering data from remote resources without any changes in the resource side and in turn it makes possible the seamless installation of new resources. Moreover, as described in details in Section 3.3, such a solution allows for keeping settings on a remote resource, that in turn gives the same environment independently of by which portal the resource is accessed.

Extremely important is also the possibility for simple extension of the WebCI functionality by writing parsers based on regular expressions, for any batch commands, in near future also for interactive ones. This solution allows for simple installation of a personal WebCI server, and to configure it for web access to any (not only Condor) resources. The use of standard tools in combination with pure Perl code enables for running this part of WebCI not only on the Linux platform. Since both SSH and scp have to be executed with the TTY device, utilization of several non–standard, but freely accessible from CPAN [1], Perl modules are required, i.e. IO–Pty and Net–Telnet.

The lowest layer is a machine with Condor installed on the submit node of the pool. In case of using only by logged–in users, this layer does not need any administration efforts. However, for public monitoring one of the following conditions have to be fulfilled, according to what functionality is needed

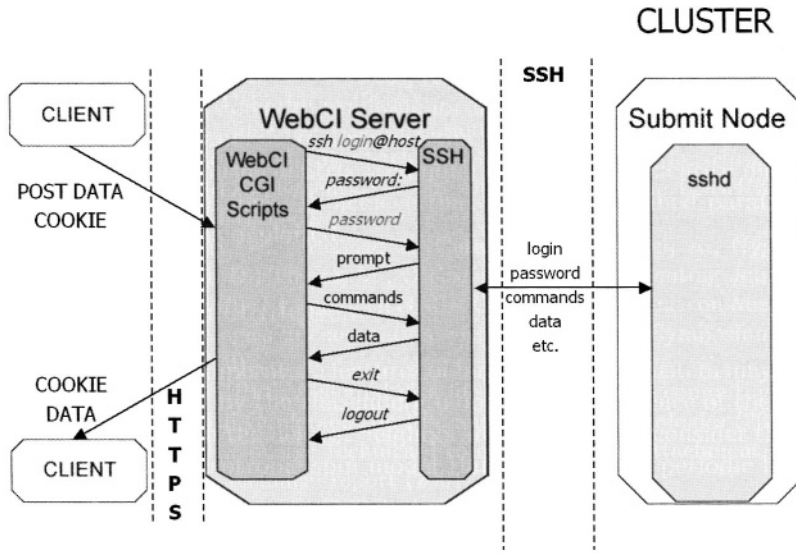


Fig. 4. Transaction through SSH

- installation of web server and simple CGI script – older method;
- launching SOAP proxy–daemons developed by us – for monitoring of queues on submit nodes without public IP addresses;
- both of above; this increases the system stability because WebCI switches for monitoring through CGI script when SOAP proxy–daemons are down.

3.2 Communication

Since there are three layers in the WebCI software, two levels of communication are considered: between web browser and web server, and between web server and resources. All communications between a web browser and a web server use the HTTPS protocol; for security reasons all HTTP requests are automatically redirected. The use of WebCI is impossible unless both server and client support HTTPS. Data that are transferred from a client to a server are divided into 2 classes: minor and major. Minor data, e.g., pool name, queue name, "cluster colors", and so on are transferred using the HTTP GET method. Major data including file names, working directories, histories, shell and Condor commands, file transactions, etc. (also login and password, but only when logging) are transferred with the HTTP POST method. Some of these data are send through hidden form fields. Login and password are transferred in secure cookies.

The communication between Web server and resources is more complex. WebCI fetches data from resources choosing a sutable method dynamically accordingly to the needed functionality. When a user is logged–in, all data are transferred by SSH and scp. After transferring, they are parsed with utilization of regular expressions or simply wrapped to a HTTP response without parsing.

There are two ways of fetching resource information while user is non–logged–in. Using CGI scripts is the older one. The scripts are executed on resources, i.e. submit nodes, in to monitor queues. Scripts can be accessed through both the HTTP and HTTPS protocol; in this case the LWP Perl module on WebCI server is required, as well as the Crypt–SSLeay module if HTTPS is utilized. This method is useful only for monitoring queues on nodes with public IP; also the web server is necessary on the resource side. The second possibility is to fetch information using our SOAP proxy–daemon (see Fig.5); the SOAP–Lite module is required in the WebCI server in this case. The SOAP proxy–daemon can act both as a proxy and end–point, and is implemented as a simple Perl script. Both methods can be used at the same time, but the CGI method is not used unless SOAP is down.

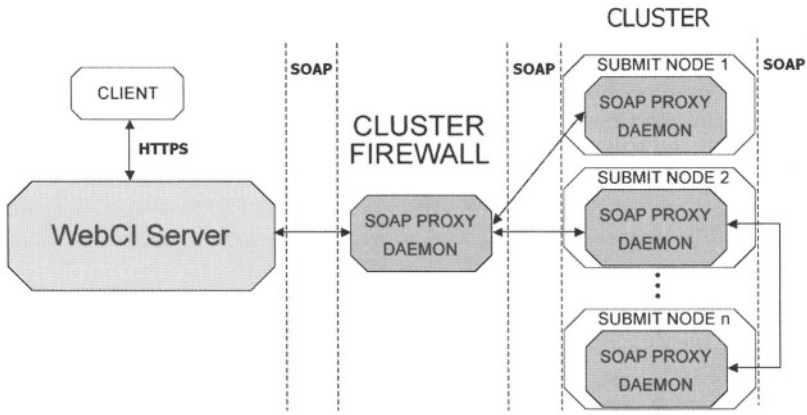


Fig. 5. SOAP communication between WebCI server and resource

3.3 Persistence

Taking into the account non–persistent nature of the HTTP protocol, persistence is very important aspect of WebCI. There are two aspects of persistence. The first one assumes keeping state from one user's session to another one (or from logout of one user's session to login for the next one). The state can not be kept on the client–side, keeping on server–side is also not the best solution since a user would have not the same settings while logging through a portal different from the portal utilized in the last session. That is why keeping data on the user's shell account gives the best functionality, but the worst performance, since data have to be transferred from resources to the server during each HTTP session. This method is suitable for keeping preferences, and job repository. Transparent access to these data is obtained thanks to the usage of tied hash tables in which all data transfers are encapsulated. The Second aspect of persistence assumes keeping data between HTTP sessions. In this case all data are kept on the client–

side. For this aim, WebCI uses the following methods: (i) hidden form fields, (ii) url query string, (iii) http cookies.

4 Future Works

Support of mobile devices is one of the most interesting directions of our future works. We are going to develop a WAPCI service with the full WebCI functionality. Also support of interactive commands will be a great step in the WebCI evolution. We are going to develop a server that keeps opened SSH sessions between sequential HTTP sessions; moreover it will grant us a better performance. An administration panel is also desirable. Please visit the first WebCI system at https://eltoro.pcz.pl/~monitor.

References

1. Comprehensive Perl Archive Network: http://cpan.perl.org/
2. High Performance Cluster Computing: Programming and Applications, Volume 2. Rajkumar Buyya ed., Prentice Hall PTR, 1999
3. The Condor Project Homepage: http://http://www.cs.wisc.edu/condor/
4. Olas, T., Karczewski, K., Tomas, A., Wyrzykowski, R.: FEM computations on clusters using different models of parallel programming. Lecture Notes in Computer Science, Springer–Verlag, 2328(2002) 170–182
5. Olas, T., Lacinski, L., Karczewski, K., Tomas, A., Wyrzykowski, R.: Performance of different communicatio mechanisms for FEM computations on PC–based clusters with SMP nodes. Proc. Int. Conf. on Parallel Computing in Electrical Engineering – PARELEC 2002, Warsaw, Poland, IEEE Computer Society, 2002, pp.305–311
6. OpenPBS Project Homepage: http://www.openpbs.org/
7. PBSWeb Project Homepage: http://www.cs.ualberta.ca/~pinchak/PBSWeb/
8. Wyrzykowski, R., Meyer, N., Stroinski, M.: PC–Based LINUX Metaclusters as Key Elements of Grid Infrastructure. Proc. CGW'02, Cracow, 2002, p. 96–103

Fine-Grained System-Call Scheduling in CEFOS on Commodity Processors

Shigeru Kusakabe, Kentaro Iio,
Hideo Taniguchi, and Makoto Amamiya

Grad. School of Information Science and Electrical Engineering, Kyushu University
6-10-1, Hakozaki, Higashi-ku, Fukuoka, Japan
kusakabe@csce.kyushu-u.ac.jp

Abstract. We are developing an operating system, CEFOS, based on a fine-grain multi-threading model. By consisting a process with fine-grain threads, we can employ flexible scheduling schemes as long as thread dependencies allow. This paper presents WSC mechanism in CEFOS, which buffers multiple system-calls until the number of system-call requests or the time-slice reaches some threshold and then transfers the control to the operating system kernel with a bucket of the buffered system-call requests. We have developed CEFOS based on Linux and measured the system-call performance on commodity Intel processors. While fine-grained system-call scheduling with WSC caused frequent cache misses, it reduced context switches and improved the total execution time thanks to the reduction of TLB misses.

1 Introduction

We are developing an operating system, CEFOS (Communication and Execution Fusion OS), in order to realize an efficient parallel and distributed computing environment based on a fine-grain multi-threading model [6]. Fine-grain multi-threading is one of the important techniques for high performance parallel processing. In addition to the processors of research projects, some commercially available products support hardware-level multi-threading[1][2]. Operating systems play an important role for optimizing the system-level performance even on multi-threading systems [7].

This paper presents WSC (Wrapped System Call) mechanism in CEFOS. While executing programs under the control of operating systems, we use system-calls or upcalls[9] for interaction between user programs and the operating system kernels. The problem in these methods is overhead of context switches, which occur whenever a system-call or upcall occurs [3]. WSC in CEFOS buffers multiple system-call requests until either the time-slice or the number of requests satisfies a threshold and then transfers the control to the operating system kernel with a bucket of the buffered system-call requests. WSC mechanism is effective in reducing the frequency of context switches and communication between user processes and the CEFOS kernel. We have developed CEFOS based on Linux and measured the system-call performance on commodity Intel processors. While

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 383–388, 2004.
© Springer-Verlag Berlin Heidelberg 2004

fine-grained system-call scheduling with WSC caused cache misses, it reduced context switches and the total execution time thanks to the reduction of TLB misses.

This paper organizes as follows. Section 2 briefly introduces CEFOS, an operating system based on a fine-grain multi-threading model. Section 3 describes WSC, which promotes cooperation between user processes and the CEFOS kernel and reduces system-call overhead. Section 4 shows performance evaluation results and discusses the effectiveness of our fine-grained system-call scheduling.

2 CEFOS

Figure 1 illustrates the outline of process/thread control structures in CEFOS. In CEFOS, a thread is a unit of processor allocation, while a process is a unit of resource management. Threads in CEFOS are very fine-grained compared to conventional threads such as Pthreads. Application programs are multi-threaded written in a multi-threading language, FUCE-C[4]. Each thread has a synchronization counter, which is decremented when the thread is triggered, and starts to run when the counter becomes zero. Each thread is non-preemptive running to its completion. A process has a thread scheduler and schedules its ready threads. While the kernel is not multi-threaded, the external-kernel mechanism intermediates interaction between the kernel and thread schedulers in user processes. In order to simplify control structures, process control is only allowed at a point of thread switching. Threads in a process are not totally-ordered but partially-ordered. By consisting a process with fine-grain threads, we can introduce flexible scheduling mechanisms, such as WSC, as long as thread dependencies allow.

3 WSC: Wrapped System Call

While running applications under the control of operating systems, frequent context switches and communication between user processes and the kernel are performed behind the scene. System-calls request the service of the kernel, thus voluntarily cause mode-changes and context switches. One of the key issues to improve system-level throughput is to reduce the frequency of context switches and communication between user processes and the kernel. We develop a mechanism for efficient cooperation between the operating system kernel and user processes.

3.1 Control Flow in WSC

WSC buffers system-call requests from a user program and sends them as a single large system-call. With the cooperation with schedulers, WSC reduces the number of calls to the kernel from a user process, and overhead caused by mode-changes and context switches.

Figure 2 illustrates the control flow in WSC:

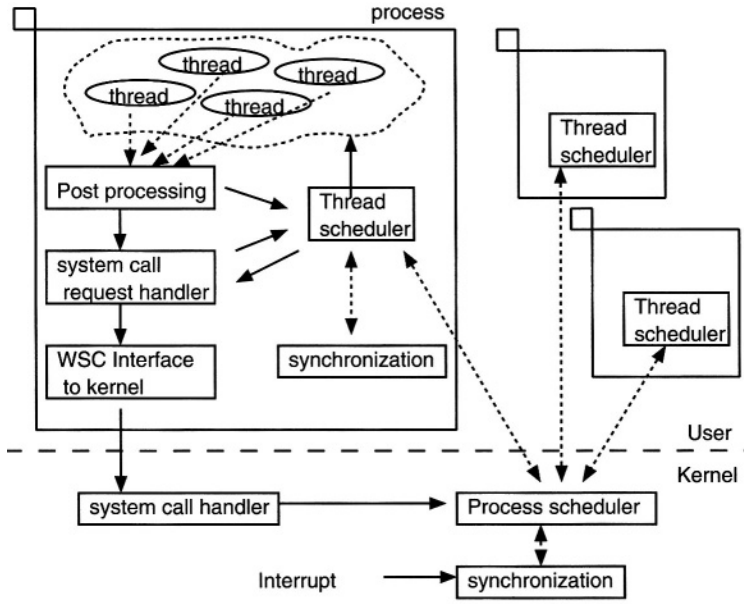


Fig. 1. Outline of process/thread control structures in CEFOS

(1) If a thread terminates with a request of a system-call, the system-call buffering routine (SCBR) is invoked. If the thread terminates with other than a request of a system-call, the thread scheduler is invoked.
(2) SCBR buffers the request of the system-call and checks whether the number of requests or the time-slice has reached the threshold. If the threshold is not reached, the thread scheduler is invoked to select the next thread to run.
(3) If the number of requests or the time-slice has reached the threshold, SCBR sends the requests of system-calls to the operating system kernel in order to actually process the system-calls.
(4) The process scheduler is invoked when the kernel has finished all the system-calls in WSC. Then the process scheduler will select the thread scheduler of the process. The return results of the system-calls may make some threads ready to run.

We use Display Requests and Data (DRD) mechanism [8] for cooperation between user processes and the CEFOS kernel at a fine-grain thread level. Parameters and return results of system-calls under WSC are passed through the common memory area of DRD shared between the kernel and user processes, to avoid frequent switches between user programs and the kernel.

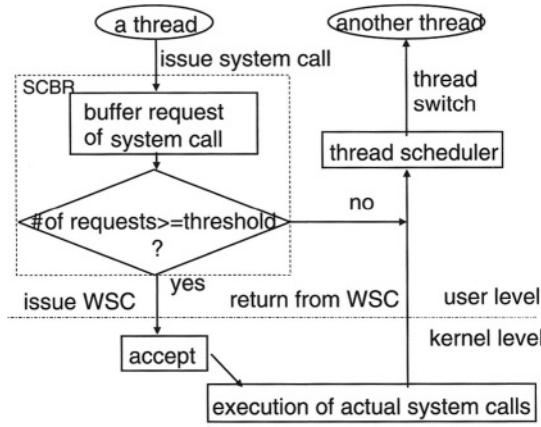


Fig. 2. Control flow in WSC

3.2 Classification of System-Calls

Under WSC, system-call requests are delivered to the kernel with potential delays. Currently, WSC returns after all the system calls sent to the kernel have finished their actual execution. Thus, all kinds of system-calls are not considered to be buffered by WSC.

We classify the system-calls into three types, which will:

- complete after unpredictable waiting time,
- complete after predictable waiting time, or
- complete without waiting time.

We can safely use WSC for the system-calls which complete without waiting time. We do not use WSC for the system-calls which complete after unpredictable waiting time because threads depending on the result of the system-call must wait for the result unpredictably long time. It is possible to use WSC for the system-calls with predictable waiting time while the response time may be degraded.

4 Performance Evaluation of WSC

In order to clarify overhead of system-calls, we measured the elapsed time of a system-call in Linux-2.4 on PCs with commodity Intel processors, such as Pentium and Celeron. As a system-call, we used `getpid`, which has a simple body and is suitable for experiments to clarify overhead common in system-calls. In this experiment, we measured the elapsed time of 64 `getpid` system-calls under WSC changing the threshold of WSC as 1, 2, 4, 8, 16, and 32. We measured the number of clocks using the hardware clock counter of the processors. The elapsed

time in μs is calculated from the clock speed and the number of clocks. Figure 3 shows the elapsed time in μs on Celeron 300AMHz, PentiumIII 800MHz, and Pentium4 1.8GHz. We also measured the elapsed time of successive 64 `getpid` system-calls under the usual system-call convention, and plotted its results as "normal" in Fig. 3.

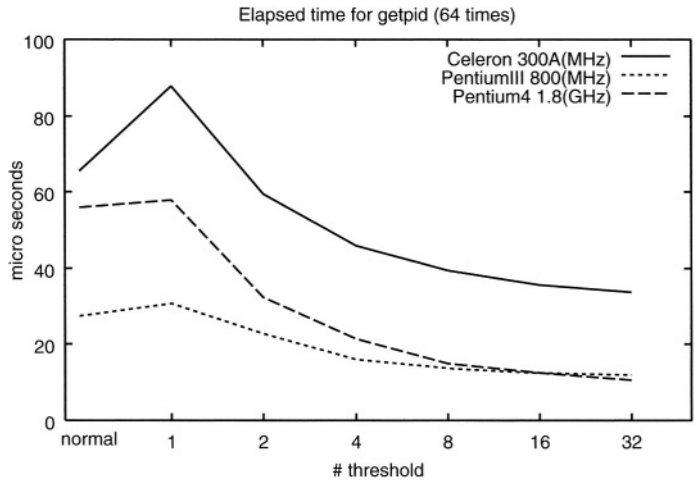


Fig. 3. Elapsed time of `getpid` (64 calls)

In contrast to our intuition, the elapsed time of the normal system-call mechanism on Pentium 1.8GHz is longer than the elapsed time on PentiumIII 800MHz. Ideally, the number of clocks should be the same regardless of the difference of the clock speed, and the faster the clock speed is, the shorter the elapsed time (in μs) should be, as they have the same instruction set architecture. For example, if we compare Pentium4 of 1.8 GHz and PentiumIII of 800MHz, we expect the elapsed time of Pentium4 should be reduced to less than half of the elapsed time of PentiumIII. The latter processor has 10 pipeline stages, but the former processor has deeper 20 pipeline stages. We see that as the processors speed up and have the deeper pipeline, system-calls degrade the performance more seriously. Thus, a system call is more time-consuming on sophisticated processors.

As seen from Fig. 3, we observe the effect of WSC after the threshold becomes 2. The elapsed time in WSC is reduced as the threshold gets larger regardless of the clock speed. Anomalies between Pentium4 1.8GHz and Pentium III 450MHz are also alleviated as the threshold gets larger.

In order to observe more performance monitoring counters, we used *hardmeter* as a performance profiling tool[5]. This tool profiles the results of performance monitoring counters on processors of NetBurst architecture, such as Pentium4. We profiled the results of `getpid` system-calls under WSC and normal conven-

tion on Pentium4 processor. As we were afraid, the fine-grain threaded code for WSC in CEFOS caused more frequent cache misses compared to the sequential code with normal system-calls. The numbers of instructions and memory loads are also increased mainly due to the initialization for fine-grained execution environment.

However, the number of TLB misses is reduced to about 60%, which is rather expensive on modern processors. We observed that our flexible scheduling with WSC at the fine-grain thread level effectively reduce overhead of context switches and TLB misses. This is especially effective for modern high speed processors with sophisticated implementation.

5 Conclusion

This paper presented WSC mechanism in CEFOS, which buffers multiple system-calls until the number of the requests or the time-slice satisfies some threshold and then transfers the control to the operating system kernel with a bucket of the buffered system-call requests.

We implemented CEFOS with WSC mechanism based on Linux-2.4, and evaluated its performance and effectiveness. Cooperated interaction between user-mode and kernel-mode at a fine-grain thread level effectively reduced overhead of context switches and TLB misses. We observed WSC is especially effective for modern high-speed processors with sophisticated implementation. In addition to further examination for various applications, we continue to optimize our CEFOS implementation in order to show the effectiveness of our schemes at the fine-grain multi-thread level.

References

1. G. Alverson, et al "Tera Hardware-Software Cooperation," Proc. Supercomputing, Sam Jose, 1997.
2. "Introduction to Hyper-Threading Technology," Intel, Document Number 250008-002, 2001.
3. J. Lidetke "Latest Development in Operating System, Toward Real Microkernels," CACM, Vol.39, No.9, pp.70-77, 1996.
4. Lee Li, Shigeru Kusakabe, and Makoto Amamiya "Multithreading Language FUCE-C and Its Implementation," Hinokuni Symposium 2002, pp.459-466, 2002.
5. http://sourceforge.jp/projects/hardmeter
6. S. Kusakabe, et al "Parallel and Distributed Operating System CEFOS (Communication-Execution Fusion OS, " IPSJ ISG Technical Notes, Vol.99, No.251, pp.25-32, 1999, in Japanese.
7. J. Redstone, et al "An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture," ACM SIGPLAN Notices, 35(11), pp. 245-256, 2000.
8. H. Taniguchi, "DRD: New Connection Mechanism between Internal Kernel and External Kernel," Transaction of IEICE , VOL.J85-D-1, No2, 2002, in Japanese.
9. E. A. Thomas, et al "Scheduler Activation: Effective kernel Support for the User-Level Management of Parallelism, " Proc, of the 13th ACM Symp. on OS Principles, pp95-109, 1991.

Dynamic Process Communication in the GDE Environment

Jan Kwiatkowski[1,2] and Daniel Abrich[1,2]

[1] University of Missouri – St.Louis
Mathematics and Computer Science Department
8001 Natural Bridge Road, St.Louis, MO 63121, USA
[2] Computer Science Department,
Wroclaw University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
{kwiatkowski, abrich}@ci.pwr.wroc.pl

Abstract. GDE – the Graphic Development Tool is a visual tool, which supports application development process for PVM environment. It is intended to provide a simple human interface to the process of message-passing programs creation. Programmers describe the program structure using two diagrams: process and sequence, which define static and dynamic application structure, respectively. The GDE automatically generates the skeleton of the message-passing program in C++ language. The GDE is implemented on the top of the PVM environment.

1 Introduction

There is a large diversity of parallel/distributed computer organizations, including those made possible by high-speed network. Thus, distributed parallel programming using networked computers became one of the most attractive and cheap way to increase the computing power. In particular, the message-passing paradigm became very popular. PVM (Parallel Virtual Environment) is a software environment, which emulates a distributed-memory multiprocessor in a heterogeneous networked environment. On the other hand, object oriented programming is being in common use. Although, there are a lot of different tools supporting application design process like GRADE, VPE, TRAPPER, CODE, ALWAN [2,5,6] and others working at different environments, there are no tools dedicated for developing object-oriented applications under PVM. These are the main motivation for the work presented in this paper. The paper deals with the short description of the GDE - a Graphical Development Environment, which supports the design process of parallel applications for message-passing environments. In the paper we focuses how different dynamic aspects of design application can be expressed in GDE (dynamic communication and dynamic process creation). The tool assists the programmer in describing the structure of a parallel program by drawing different diagrams. The tool automatically generates the skeleton of the program in C++ language (header's and implementation's prototype files). The prototype of the tool was developed and implemented with

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 389–396, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the C++ language under UNIX system on the top of the PVM. The paper is organized as follows. Section 2 briefly describes PVM environment and related to GDE tools. Section 3 discusses the basic GDE notation and describes the design process when using GDE. Section 4 presents how different dynamic aspects of designed application can be express in the GDE. Section 5 shows an example of using the GDE and finally, section 6 summarizes this work and discusses future extensions planned.

2 Other Graphic Designing Environments

PVM uses message-passing model to allow programmers to exploit a distributed computer. The system transparently handles message routing, data conversion, and task scheduling across a network of incompatible computer architectures. When using PVM, the user writes his application as a collection of co-operating tasks, which access PVM resources through a library of standard interface routines. These routines allow the initialization and termination of tasks across the network as well as communication and synchronization between them. In PVM programmer can utilize functional, as well as a data parallelism [7]. A number of supporting tools based on visual programming languages have been developed for PVM: CODE, VPE, HENCE,GRADE [2,5,6]. All of these environments are based upon the idea that program is represented as a graph which nodes represent computation and arcs represent interaction among nodes. HENCE and CODE are not based upon a traditional message-passing model used in PVM, additionally interaction between nodes which represent sequential computation occurs only at the beginning and end of computations. In CODE and HENCE communication are expressed at a higher level of abstraction not like in VPE where programmers make explicit call to the message-passing library routines. GRADE is an integrated environment equipped with built in number of additional supporting tools as debugger, profiler, etc. The fundamental difference between above environments and GDE is that GDE enables designing object oriented applications. Besides this GDE is based on traditional message-passing model like VPE. Similar to CODE and HENCE com munication in GDE is expressed at a higher level of abstraction. The "communication code" is generated automatically and programmers make no explicit calls to message-passing library routines. Like in VPE in GDE the generation of a new task is performing automatically on base of communication structure express at process structure diagram. Additionally, the decision about message exchange between existing processes can be taken dynamically during program execution (algorithmic dependent) as well as statically directly at the sequence diagram.

3 Application Design Using GDE

GDE allows the programmer to specify the application structure visually. Processes, message source, destinations and flows are specified graphically. The process of dynamic process generating and message exchange during program execution ("late binding") is also supported. GDE's graphical language is explicitly

parallel. A programmer specify directly the parallel structure of the program using two diagrams: process diagram (static structure), which defines the application process structure (processes and static connections between them) and sequence diagram (dynamic structure), which defines the order of process execution and messages exchange. How many processes will be created depends on user specification or is dynamically determine during program execution. Processes communicate via communication ports. Communication ports are distinguishable by it's name, which is compose of prefix "from" or "to" for input and output ports respectively, and the name of the co-operating process. Two connected communication ports create a link. Link is one directional and acts as communication channel between processes. Processes are displayed at process diagram as boxes with single vertical and horizontal lines and contain three fields. The upper field specifies the process name. Used "real or virtual" architecture is specified by the middle field. The optional lower field specifies the number of process instances. The processes communicate via channels using named task ports represented as "arrow head" for an output ports and "arrow tail" for input ports, respectively. Lines connecting task ports represent the channels. Each process can create child processes at any time. The arrows at diagram represent the process of dynamic task creation. Used graphic notation is summarized in figure 1.

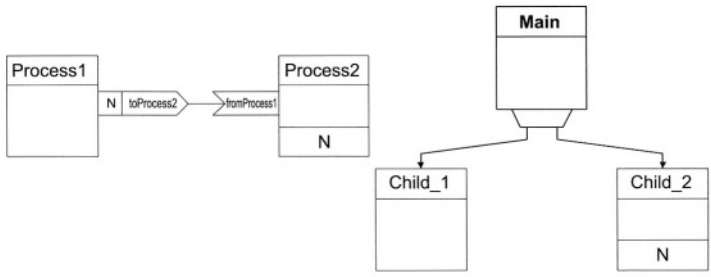


Fig. 1. Graphical notation for processes, communication channels and task creation.

The second diagram - sequence diagram is used for defining the order of process execution. Using sequence diagram the programmer specifies which defined at process diagram communication channels are used and when the communication between processes occurs by defining for each process so called "process life line", which represents the time duration (vertical dashed line). In the similar way he can specifies which process and when create child processes. The multiple processes are displayed using double lines. The moment of process creation is mark at "process life line", however using the "N option" at the process and sequence diagram the number of created child processes is determine during program execution. Used graphic notation is summarized in figure 2.

The process of application design using GDE is very simple and is divided onto three steps. In the first step using graphical editor programmer defines the static and dynamic structure of the designed application using such graph-

ical symbols like: processes, ports, communication channels and child process creation. Tool options are selected easy with the mouse from the toolbar. Additionally, when using graphical editor the following options are also available: Project - serves the common "edit" functions; Program - enables editing properties of the selected process and code generating, Diagram - enables switching between process and sequence diagrams. The most common used commands like: New, Open, etc. are also available through shortcuts.

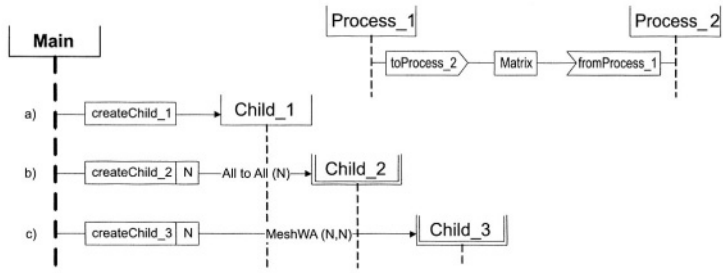


Fig. 2. Communication between processes and process creation.

After defining the static structure at process diagram and dynamic structure at sequence diagram in the second step the tool automatically generates code in C++ language (Generate code option). For each process defined at process diagram: <process_name>.h - class header, <process_name>.cpp - 'init' method - user class constructor; 'execute' method, <process_name>Prog.cpp - process constructor and destructor, "architecture communication" methods and others, which cannot be modified by user are created. For user defined type: <user_type_name>.h - class header, <user_type_name>.cpp - user type's methods, makefile.make - project makefile are created. During code generation basing on the defined diagrams the references to the "PVM specific" functions included in the tool embedded library are added. Then in the third step using embedded programmer's editor the implementation for processes can be added. The user is obligate to define at least the function execute from the process Main. It is a first function, which is executed during program execution. Additional class member function can be added if it is required. Finally after program compilation the designed application is ready for use.

4 Dynamic Communication and Process Creation in GDE

At sequence diagram we can specify when communication between co-operating processes can occur by specifying the "algorithmic condition" at the right side action. The dynamic communication can be used only for communication with multiple processes. There are three main cases that can be distinguish when using dynamic communication: the multiple process sends a message to a single

one - instances of sending process can be chosen, the single process sends a message to the multiple process - instances of receiving process can be chosen, and when message is send between multiple processes. In the last case there are three different opportunities: instances of sending process (all to all/selected), or instances of receiving process (all/selected to alg), or pairs of sending and receiving processes (alg to alg) can be chosen. Below, the first mentioned case is briefly discussed. Figure 3 shows main process, which in actions 0 and 1 creates Process1 and Process 2, respectively, then in action 2 Processl sends the message to selected copies of Process2 and in action 3 Process2 received message from selected copies of Process1.

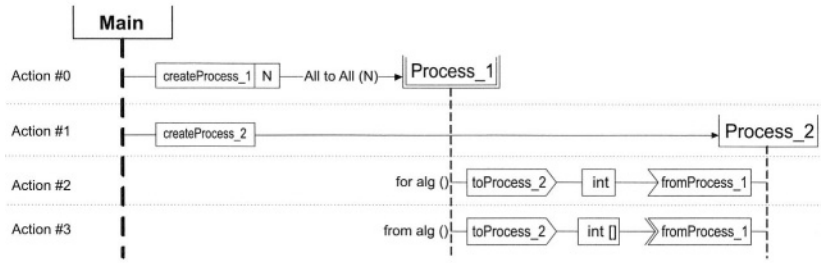


Fig. 3. Dynamic communication between multiple and single processes.

The decision which instances of the multiple process send/receive message depends on the user-defined algorithm and is determine dynamically during program execution. When the value true is return it means that this copy should send a message. The defined by user procedures are later call by the sendAlg or receiveAlg methods. The PVM mini database are used for communication between both above methods to determine if the message should be send or receive, respectively. Generated by the GDE code is give in figure 6, parts, which should be defined by the user, are shaded.

Similarly, as for dynamic communication, the dynamic child process creation can be used only with multiple processes. Figure 4 shows process Main, which creates multiple process Process1 (the number of created instances are defined by createProcess_1N variable) and multiple process Process2. The number of created instances depends on the size of Mesh architecture which is determine by two variables: createProcess_2rows and createProcess_2columms.

The code generated for above example is given in figure 5 (parts, which should be defined by the user, are shaded).

5 An Example of Using GDE

To present the functionalities provided by the tool we present the following simple example. Let's consider the traveling salesmen problem, in which salesmen must visit n cities, returning to his starting point, and is required to minimize the cost of the trip. In the first design step we create the process diagram. Two

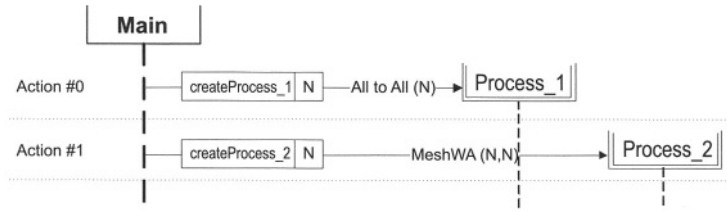


Fig. 4. Sequence diagram for dynamic process creation.

```
int Main::execute (int argc, char *argv[]) {
// Action #0 - begin
// Create child process 'Process_1', You must set that variables:
  createProcess_1N = ;
  createProcess_1N = createChild ("Process_1", createProcess_1N, 0, 0);
// Action #0 – end & Action #1 -begint
// Create child process 'Process_2' you must set those variables:
  createProcess_2rows = ;
  createProcess_2columns = ;
createProcess_2N = createChild ("Process_2", fromProcess_2MeshWAArch (createProc
   ess_2rows, createProcess_2columns), 0, 0, 2, createProcess_2rows, createProcess_2columns);
// Action #1 - end
return 0; }
```

Fig. 5. A code example.

```
int Process_1::toProcess_2Alg0 () {  }
int Process_1::toProcess_2Alg1 () {  }
int Process_1::execute () {
// Action #0 - begin
// created by process 'Main'.
// Action #0 – end  & Action #2 - begin
// Send data to process 'Process_2'.
  if (sendAlg ("Process_2", toProcess_2Alg0
      (), 2)) {
// You must set this variable:
    toProcess_2Data0 = ;
    toProcess_2->send (toProcess_2Data0);  }
// Action #2 – end &Action #3 - begin
// Send data to process 'Process_2'.
  if (sendAlg ("Process_2", toProcess_2Alg1
       (), 3)) {
    // You must set these variables:
    toProcess_2Data1 = ;
    toProcess_2DataSize1 = ;
    toProcess_2->send (toProcess_2Data1,
    toProcess_2DataSize1);  }
// Action #3 - end
 return 0;  }
```

```
int Process_2::execute () {
// Action #1 - start
// created by process 'Main'.
// Action #1 – stop & Action #2 - start
  // Receive data from process 'Process_1'.
  for (int p=0; p<getMult ("Process_1", 0,
        &fromProcess_1); p++) {
    if (receiveAlg ("Process_1", 2, p)) {
// data received 'fromProcess_1Data0'
      fromProcess_1[p]->receive (
            fromProcess_1Data0);  }  }
// Action #2 – stop & Action #3 - start
 // Scatter data from process 'Process_1'.
 // data received  'fromProcess_1Data1'
 // array size received
 // 'fromProcess_1DataSize1'
  scatterData (fromProcess_1, getMult (
      "Process_1", 0, &fromProcess_1),
      &fromProcess_1Data1,
      fromProcess_1DataSize1, 3);
// Action #3 - end
  return 0; }
```

Fig. 6. A code skeleton generated by the GDE.

different processes can be distinguish: TF-Main, which acts as a master process, it distributes the work among slave processes and TF-Worker which is created by the TF-Main, and calculates the shortest path. The number of created child processes depends on the number of available processors and splitting algorithm used by TF-Main process. Figure 7 shows the static structure of the analyzed algorithm.

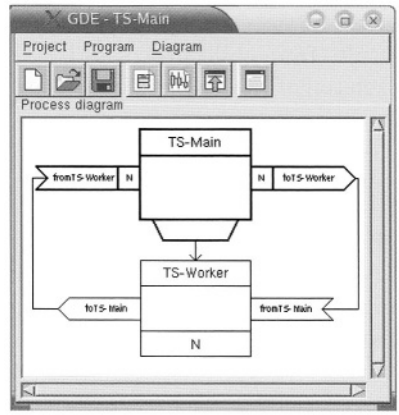


Fig. 7. Process diagram for the parallel search algorithm.

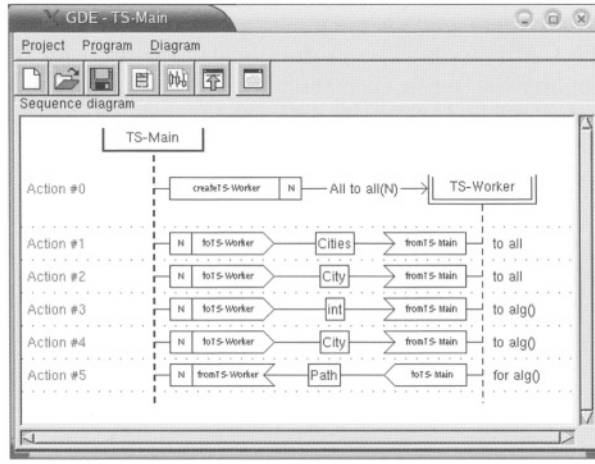


Fig. 8. Sequence diagram for the parallel search algorithm.

In the next step sequence diagram presented in figure 8 is created. The sequence diagram shows the dynamic structure of the presented algorithm - the sequence of message exchange and child process creation. In presented parallel search algorithm we can utilize dynamic behavior only for the message exchange between TF-Worker processes and TF-Main process. The TF-Worker process sends the result only when it is better than the result specified by the current shortest path. Each specified at sequence diagram action are briefly described below:

Action #0 - creation of child processes TF-Worker,

Action #1 - sending selected cities (depends on splitting algorithm) from TF-Main to all instances of TF-Worker process,

Action #2 - sending selected "startCity" to all "TS-Worker" processes

Action #3 - sending the current shorted path (integer value) to all TF-Worker processes,

Action #4 - sending the rest of the "search space" to TF-Worker processes,
Action #5 - collecting results from child processes by TF-Main process.

In the next step the skeleton of the application is generated. The presentation of the code is omitted due to lack of space.

To confirm the usefulness of the tool we performed some experiments using four HP workstations connected by local network. We compare speedup received for the program generated by the GDE tool with the program which was written manually. Received results shows that the overhead causes by the GDE tool is about 3%. For example for the presented in the paper algorithm speedup for program written manually was 3.85, when speedup received using program generated by the GDE tool was 3.73. It means that cost of using the GDE is low and it can be acceptable.

6 Conclusions

Presently the tool covers only a part of the functionalities that normally is supported by the CASE tools. The prototype still misses a lot of features that could be implemented in the further versions. However experiments performed using our prototype indicates that the presented tool will be useful for designing PVM applications using object oriented programming techniques. No specific knowledge about the PVM is required to build the application. The tool helps the developer out in PVM knowledge and allows him to concentrate on the essential part of the implementation. The decision of using OO Technology for coding will contribute to easy the implementation of the full set of desired functionalities. Additionally it gives an opportunity of using the GDE for other distributed environments like for example MPI after changing the internal communication library.

References

1. Beguelin A. & others, Visualisation and Debugging in a Heterogeneous Environment, IEEE Computer, v. 26, No 6, June 1993.
2. Burkhart H & others, ALWAN programmer's Manual, (Institute for Informatic, University of Basel, Basel 1996).
3. Foster I., Designing and Building Parallel Programs (Addison-Wesley Pub., 1995).
4. Kwiatkowski J., Abrich D., GDE – Tool Supporting Application Development for PVM Environment, Proceedings of the IASTED International Conference "Applied Informatics", Innsbruck, Austria, ACTA Press 2002.
5. Newton P., Browne J.C., The CODE 2.0 Graphical Parallel Programming Language, Proceedings of ACM International Conference on Supercomputing, July 1992
6. Newton P, Dongarra J., Overview of VPE: A Visual Environment for Message-Passing Parallel Programming, available at PVM web site.
7. PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Computing (The MIT Press, Cambridge, Massachusetts, 1994).

A Lightweight Message Logging Scheme for Fault Tolerant MPI

Inseon Lee[1], Heon Y. Yeom[1], Taesoon Park[2], and Hyoungwoo Park[3]

[1] School of Computer Science and Engineering,
Seoul National University,
Seoul, 151-742, KOREA
{inseon,yeom}@dcslab.snu.ac.kr
[2] Department of Computer Engineering
Sejong University
Seoul, 143-747, KOREA
tspark@kunja.sejong.ac.kr
[3] Supercomputing Center, KISTI,
Taejon, 305-333, Korea
hwpark@hpcnet.ne.kr

Abstract. This paper presents a new lightweight logging scheme for MPI to provide fault tolerance. Checkpointing recovery is the most widely used fault tolerance scheme for the distributed systems. However, all the processes should be rolled back and restarted even for a single process failure to preserve consistency. Message logging can be used so that the other processes can proceed unaffected by the failure. However, logging all the messages tends to be prohivitively expensive. We note that the applications programmed using MPI follow certain rules and not all of the messages need to be logged. Our logging scheme is based on this observation and only the absolutely necessary information is logged or piggybacked. As a result, it is possible to greatly reduce the logging overhead using our scheme and the experimental results matched well with the expectation.

1 Introduction

MPI is the defacto standard for writing parallel programs running on parallel computers, network of workstations(NOW), and computational GRID[1]. Although programming using MPI is more complicated than programming using distributed shared memory systems(DSM), it is more widely used since it is easy to support on various computing platforms and has far better performance. In today's large scale distributed systems, a node failure is not something that rarely occurs but a frequent event which needs to be carefully dealt with. For the distributed systems to be of any practical use, it is important for the system to be recoverable so that the processes do not have to restart from the beginning when a failure occurs. However, most MPI implementations do not address the fault tolerance issues.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 397–404, 2004.
© Springer-Verlag Berlin Heidelberg 2004

There are quite a few approaches trying to provide fault tolerance to MPI. Cocheck [2] and Starfish [3] provide checkpointing based fault tolerance. These methods rely on the consistent set of checkpoints and all the processes have to be restarted in case of a failure. MPIFT[4] employs pessimistic message logging and MPI-FT[5] provides both pessimistic and optimistic message logging. Other approaches include FT-MPI[6] and MPICH-V[7]. Almost all of these approaches rely on some form of indirect communication to log messages as well as guarantee consistency. FT-MPI is the only exception and shows much better performance. However, application programmer should be aware of the checkpointing/logging activity and corrective operations need to be provided by the programmer. The main advantage of MPI over other message passing interfaces like PVM is the performance which results from direct communication without any mediator. If the message should be relayed in any reason, the performance suffers. We want to provide fault tolerance without any performance degradation.

In this paper, we present a causal logging implementation used in MPICH-GF[8], a fault tolerant MPI implementation based on the MPICH-G2, the Grid enabled MPI. MPICH-GF supports coordinated checkpointing as well as independent checkpointing with message logging. Both Pessimistic message logging and optimistic message logging are supported. Optimistic logging alone can lead to cascading rollback and our implementation is augmented with causal logging. No additional communication layer was added so that the performance advantage can be retained while providing fault tolerance. Our design is focused on providing Globus users a sure way to execute long-running applications without having to worry about failures. A typical Globus user can execute an existing MPI application without modifying the application program source.

The rest of this paper is organized as follows: In Section 2, we briefly discuss the related works. The system model and the definition of consistent logging for the correct recovery are presented in Section 3. The protocols for causal logging and recovery are presented in Section 4. The performance of the proposed protocol is discussed with the experimental results in Section 5 and Section 6 concludes the paper.

2 Related Work

There are basically two ways to program distributed systems. One is to use message passing environment such as MPI or PVM and the other is to use distributed shared memory systems. In order to provide fault tolerance, checkpointing and logging is used in both cases.

Causal logging is one logging approach which is gaining a lot of attention for the message-passing based distributed computing systems [9]. In the causal logging technique, the sender-based logging of data items is performed and the access information is logged at the volatile storage of the dependent processes. Since this scheme completely eliminates the needs for stable logging, logging overhead can significantly be reduced. Also, since the storage of the dependent processes are utilized, concurrent and multiple failures can be handled. However,

in this scheme, the log of the access information has to be causally spread over the dependent processes, which may cause the non-negligible message overhead.

A causal logging scheme for the DSM system based on lazy release consistent(LRC) memory model[10] has been suggested in [11]. In this scheme, to reduce the message overhead, the data structures and operations supported by the LRC model, such as *diff*, write notices, and vector clocks, are utilized. The authors has proposed another efficient scheme in [12] which further reduces the amount information carried by each message. Instead of logging the vector clock for each synchronization operation, the sufficient and necessary information to recreate the corresponding vector clock is inserted into the existing write notice structures. Similar technique can be applied to the causal logging based on MPI communication. The idea of causal logging for the message passing system was first introduced in [13] where the authors identify the necessary information to replay the messages for debugging purposes. Our implementation is based on this idea and MPI specific information is utilized to reduce the amount of data to be logged as was done in case of recoverable DSM systems.

3 Background

3.1 System Model

We consider a Grid system consisting of a number of fail-stop nodes [14], connected through a communication network. Each node consists of a processor, a volatile main memory and a non-volatile secondary memory. The processors in the system do not share any physical memory and communicate by message passing. Globus[15] is used to provide communication and resource management. Failures considered in the system are transient and a number of concurrent node failures may happen in the system. Applications running on the Grid systems is programmed using MPI and each application is executed on fixed number of nodes communicating with one another. The computation of a process is assumed to be *piece-wise deterministic*; that is, the computational states of a process is fully determined by a sequence of data values provided for the sequence of receive operations.

3.2 Consistent Recovery

We define a *state interval*, denoted by $I(i, \alpha)$, as the computation sequence between the $(\alpha - 1)$-th and the α-th synchronization operations of a process p_i, where $\alpha > 1$ and the 0-th synchronization operation means the initial state of p_i. Then, in the Grid system where applications communicate using MPI, the computational dependency between the state intervals can be defined as follows:

Definition 1: A state interval $I(i, \alpha)$ is dependent on another state interval $I(j, \beta)$ if any one of the following conditions is satisfied:
(a) $i = j$ and $\alpha = \beta + 1$.
(b) $I(j, \beta)$ ends with a $send(i, x)$ and $I(i, \alpha)$ begins with an $receive(x)$.
(c) $I(i, \alpha)$ is dependent on $I(k, \gamma)$ and $I(k, \gamma)$ is dependent on $I(j, \beta)$.

Definition 1.(a) indicates the natural dependency within a process, Definition 1.(b) presents the inter-process dependency caused by message passing, and Definition 1.(c) states that the dependency relation is transitive.

Definition 2: A state interval $I(i, \alpha)$ is said to be an orphan, if for any interval $I(j, \beta)$, $I(i, \alpha)$ is dependent on $I(j, \beta)$ and $I(j, \beta)$ is discarded by a rollback.

Definition 3: A process is said to recover to a consistent recovery line, if any state interval of the system is not an orphan after the rollback-recovery.

4 Protocol Description

4.1 Overview

Independent checkpointing in conjunction with causal logging is one way to achieve the consistent recovery. Let $Log(e_k)$ be the information logged to regenerate the exactly same event e_k. Under the assumption of the *piece-wise deterministic* computation, if $Log(e_k)$ for every receive event, which may cause the potential orphan state, can be retrieved at the time of rollback-recovery, the consistent recovery can be guaranteed. For the correct regeneration of an event, $Log(e_k)$ must include the message which have been provided for the receive event e_k, the identifier of the sender which has sent the message and the message sequence number. The causal logging consists of two parts; one is the sender-based logging of the message itself, and the other is the causal logging of the message access information, such as the message identifiers, by the dependent processes.

To uniquely identify a message, each message is tagged with the following information: the sender's id, receiver's id, sender's message sequence number and the receiver's receive sequence number. The sender's message sequence number is assigned when the message is generated. However, the receive sequence number is assigned after the message is actually received by the receiver. Both the sender's message sequence number and the receiver's receive sequence number are kept by each process and incremented whenever there is a send/receive event. These are reset to the checkpointed value in case of failure. For logging of the messages, the *MPI_send_buf* structure maintained by each process can be utilized, since this structure can be regenerated from a correct recovery even after a system failure. As for the event identifiers to trace the data access, the receive sequence number can be used along with the message id(sender id and sender sequence number).

However, in MPI applications, most receive calls are deterministic. Upon re-execution, when presented with several conflicting messages, the receive call may be able to select the same message it has received before without any additional information. When writing MPI applications, programmers assume that the ordering is preserved between communicating nodes. For some MPI implementations using more than one communication channel, it might not be true. However, in that case, programmers have to use caution so that the order reversal would not affect the outcome of the execution. When the receive call is

specified with the source rank and tag, it only receives the messages identified with the same source and tag.

The only problem is the receive calls with no source specification, which receives messages from any source. It is sometimes used when a process expects messages from several different processes in no particular order. It can be re-written using non blocking receives and MPI_wait. If that is not the case, these receives should be dealt with care so that the same message can be delivered when it is re-executed. The message id should be causally logged along with the receive sequence number.

4.2 Checkpointing and Message Logging

Each process in the system periodically takes a checkpoint to reduce the amount of recomputation in case of a system failure. A checkpoint includes the intermediate state of the process and the messages sent after the last checkpoint. Checkpointing activities among the related processes need not be performed in a coordinated way, however, if checkpointing is incorporated into the barrier operation or garbage collection, the overhead of checkpointing can be reduced.

4.3 Causal Logging

The only event that needs to be tracked is the receive operation without the source specification. We call this non-deterministic receive. When MPI_receive is called with non-deterministic receive, the message id and the receive sequence number should be recorded. If there is a checkpoint after the non-deterministic receive, the dependency information can be discarded. However, if a message should be sent out after the non-deterministic receive, the dependency information should be piggybacked to the message so that the dependent process can provide the recovery information in case of failure. It is also possible to save the information to the disk from time to time to reduce the information to be piggybacked.

4.4 Rollback-Recovery

For a process p_i to be recovered from a failure, a recovery process, say p_i', is first created and p_i' broadcasts the *recovery message* to all the other processes in the system. The *recovery message* should contain the process id and the vector clock of the checkpoint it is restoring. On the receipt of the recovery message, each process p_j first determines whether it is a dependent of p_i or not. Being a dependent of p_i means that p_j has received at least one message from p_i after it took the checkpoint which is being restored. If so, it replies with its causality notice structure, which includes p_i's receive sequence number and the corresponding message id. When p_i' collects the reply message from every process, it eliminates the duplicates and reconstructs its own message replay information. The recovery process p_i' then restores the latest checkpoint of p_i and the messages

received from the logging server is enqueued to the POSTED queue. From the reconstructed state, p_i begins the recomputation as follows:

MPI_send(p_j,m): If the last message sequence number received from p_j is bigger than that of m, skip sending the message. Otherwise, the message is sent normally.

MPI_receive(p_j,m): p_i searches the message from the POSTED queue and delivers the message. If the message is not found, it means that recovery is complete and should proceed normally.

MPI_receive(*,m): p_i searches the message replay information and selects the matching message so that the same message that was delivered before the failure can be delivered . If the information is not found, it means that recovery is complete and should proceed normally.

Theorem 1: The rollback-recovery under the proposed protocol is consistent.

Proof Sketch: If for every send/receive event e_α, an event e_β dependent on e_α exists, $Log(e_\alpha)$ can be retrieved after a failure. As a result, the rollback-recovery of a process must be consistent.

5 Performance Study

To evaluate the performance of the proposed logging scheme, we have implemented the logging scheme on top of MPICH-GF. Our experimental environment consists of a cluster of 4 PCs running LINUX 2.4 and Globus 2.2. Our MPICH-GF is based on the MPICH v1.2.3. Each PC has a 1.4GHz Pentium4 processor and 256MB of main memory.

We have run two sets of application programs, the NAS Parallel Benchmarks [16] and the SPLASH [17] applications. The NPB consists of EP, IS, CG, MG, and LU and the application progrmas we used from the SPLASH-2 suite are FFT, BT/SP, Quick Sort, TSP, and Water. By looking at the application programs, we have classified them into three groups. The first group is those applications with deterministic message receive. In other words, all the receive operations in this group specify the source of the receiving message. For these applications, there is no need to record the causality information since all the receive operations are deterministic. All the applications in the NPB except LU belongs to this group. From Splash-2, FFT and BT/SP belongs to this group.

The second group is those applications where there are non-deterministic receive operations. However, even these receive operations are deterministic since each message is uniquely tagged by the sender and processed accordingly regardless of their receiving order. TSP and Water show these characteristics. It is possible to re-write these applications using non-blocking receive and wait operations to create the same effect.

The last group where the causal logging is really needed has only one application, Quick Sort. It is a typical master-slave type parallel program where all the slaves communicate with the master to get the task to execute and the master assigns remaining tasks to the slaves. Only the master executes the non-

deterministic receives and about half of the messages the master received has been causally logged.

The performance of those applications is shown in figure 1. The execution time is normalized against the execution time with no logging. The overhead from message logging is quite high (close to 80 %) for applications exchanging lots of messages such as MG and Water. However, the overhead decreases quite a bit if we apply our protocol. Even for the applications with fewer messages, we can see that handling only *anysource* messages is beneficial.

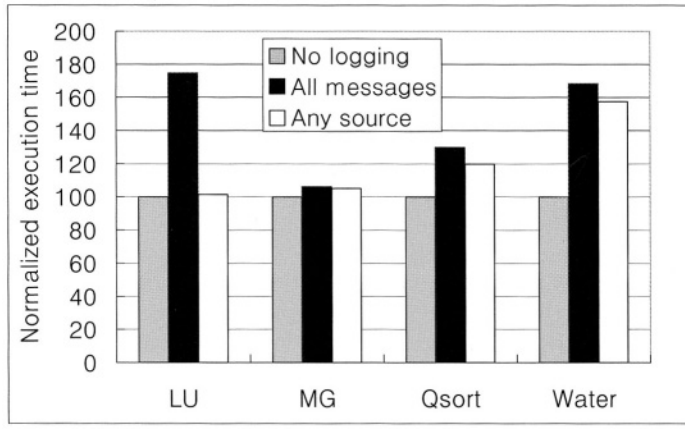


Fig. 1. The effect of Optimization

6 Conclusions

In this paper, we have proposed an efficient causal logging protocol for the Grid enabled MPICH implementation. The notable points of the proposed protocol is to reduce the amount of data to be logged using the MPI specific information. We note that only the messages received without the source specification should be logged and even that can be further reduced by closely looking at the application. As a result, causal logging can be achieved by piggybacking a small information to the outgoing message and the message overhead can be much smaller than the earlier logging schemes. To evaluate the performance of the proposed protocol, the logging protocol has been implemented on top of MPICH-GF, our fault-tolerant MPICH implementation for the Grid. The experimental results show that the proposed scheme can dramatically reduce the logs required for the causality tracking. MPICH-GF is available from http://dcslab.snu.ac.kr/projects/mpichgf/.

References

1. Foster, I., Kesselman, C. In: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Faufmann Publishers (1999)
2. Stellner, G.: CoCheck: Checkpointing and process migration for MPI. In: Proceedings of the International Parallel Processing Symposium. (1996) 526–531
3. Agbaria, A., Friedman, R.: Starfish: Fault-tolerant dynamic mpi programs on clusters of workstations. In: Proceedings of IEEE Symposium on High Performance Distributed Computing. (1999)
4. Batchu, R., Skjellum, A., Cui, Z., Beddhu, M., Neelamegam, J.P., Dandass, Y., Apte, M.: MPI/FT:architecture and taxonomies for fault-tolerant, message-passing middleware for performance-portable parallel computing. In: 1st International Symposium on Cluster Computing and the Grid. (2001)
5. Louca, S., Neophytou, N., Lachanas, A., Evripidou, P.: Portable fault tolerance scheme for MPI. Parallel Processing Letters **10** (2000) 371–382
6. Fagg, G.E., Dongarra, J.: FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In: PVM/MPI 2000. (2000) 346–353
7. Bosilca, G., Bouteiller, A., Cappello, F., Djilali, S., Magniette, G.F., Néri, V., Selikhov, A.: MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes. In: SuperComputing 2002. (2002)
8. Woo, N., Yeom, H.Y., Park, T., Park, H.: MPICH-GF, transparent checkpointing and rollback-recovery for grid-enabled mpi processes. In: Proceedings of the 2nd Workshop on Hardware/Software Support for High Performance Scientific and Engineering Computing. (2003)
9. Alvisi, L., Hoppe, B., Marzullo, K.: Nonblocking and orphan-free message logging protocols. In: Symposium on Fault-Tolerant Computing. (1993) 145–154
10. Keleher, P.J., Cox, A.L., Zwaenepoel, W.: Lazy release consistency for software distributed shared memory. In: The 18th Annual International Symposium on Computer Architecture. (1992) 13–21
11. Yi, Y., Park, T., Yeom, H.Y.: A causal logging scheme for lazy release consistent distributed shared memory systems. In: Proceedings of the International Conference on Parallel and Distributed Systems. (1998) 139–146
12. Park, T., Lee, I., Yeom, H.Y.: An efficient causal logging scheme for recoverable distributed shared memory systems. Parallel Computing **28** (2002) 1549–1572
13. Netzer, R.H.B., Miller, B.P.: Optimal tracing and replay for debugging message-passing parallel programs. In: Proceedings of Supercomputing '92. (1992) 502–511
14. Schlichting, R.D., Schneider, F.B.: Fail-stop processors: An approach to designing fault-tolerant computing systems. ACM Trans, on Computer Systems **1** (1983) 222–238
15. Foster, I., Kesselman, C.: The globus project: A status report. In: Proceedings of the Heterogeneous Computing Workshop. (1998) 4–18
16. NASA Ames Research Center: Nas parallel benchmarks. Technical report, http://science.nas.nasa.gov/Software/NPB/ (1997)
17. Woo, S., M. Ohara, E. Torrie, J.S., Gupta, A.: The SPLASH-2 programs: Characterization and methodological considerations. In: Proceedings of the 22nd International Symposium on Computer Architectures. (1995) 24–36

Improving the Performances of a Distributed NFS Implementation*

Pierre Lombard, Yves Denneulin, Olivier Valentin, and Adrien Lebre

Laboratoire Informatique et Distribution-IMAG
51 avenue J. Kuntzmann, 38 330 Montbonnot Saint-Martin, France
{plombard,denneuli,ovalenti,lebre}@imag.fr

Abstract. Our NFS implementation, NFSP *(NFS Parallèle)* aims at providing some transparent ways to aggregate unused disk space by means of dividing a usually centralized NFS server into smaller entities: a metadata server and I/O servers. This paper illustrates the issues related to increasing the performances of such an implementation. Two different approaches have been taken: distributing the load across several servers and implementing the server in a more efficient and intrusive way (in kernel mode). The results obtained with both versions are given and compared to the ones of the first user-mode implementation.

1 Introduction

Today's low-cost clusters are often built by using off-the-shelf hardware: each node has its own storage capability, usually only used to store the operating system and the runtime environment. As the hard disk capacity increases, most of the disk space of the nodes remains unused but for temporary files since the users prefer having their files available on every nodes. Partial solutions imply investing in an expensive storage architecture (SAN or RAID servers), yet the disk space is still wasted on the disks of the nodes. Systems providing an aggregation of the unused disk space and the existing ones often implement new protocols or file system types, which may not be considered as a seamless integration for the clients.

Such issues try to be solved by the NFSP project. When the NFSP project was started in mid 2001[1], we chose to use standard and well defined protocols to implement a new kind of NFS server. The first prototype implemented was based on the Linux user-mode server. The first experimental results we got with this implementation highlighted the cost of running the daemon in user-mode. To improve this we tried two methods: balancing the load between several servers and making a more efficient implementation of the server itself. This paper presents these two approaches and compares them from a performance point of view. After this introduction, some related works in the distributed file systems field are

* This work is a part of the research project named "APACHE" which is supported by CNRS, INPG, INRIA and UJF. Some resources were provided by the ID/HP *i-cluster* (More information is available at http://icluster.imag.fr/)

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 405–412, 2004.
© Springer-Verlag Berlin Heidelberg 2004

shown in section 2. Then the NFSP principles are explained in section 3 and the two methods for improving performances are detailed in sections 4 and 5 which contain performances evaluation. Eventually, some future tracks of research will be tackled in section 6.

2 Related Works

A large amount of work has been carried out in the network file system since the 1980s. Among the first ones, still used nowadays are Sun NFS and Carneggie Mellon's AFS. NFS is aimed at sharing files among nodes in the same LAN whereas AFS is more suited for WAN architecture. A NFS [2,3] server is made of a node exporting its local file system to the clients who access it through a remote mounting operation. NFS is a stateless protocol, no state is kept on the server side so every operation is self sufficient. This gives NFS some protection against temporary faults. However since the access point is unique for all clients the implementation is inherently centralized and so the storage space is limited to the one on the server. This is not the case for AFS which is a fully distributed file system: servers across different sites cooperate to share the same space and offer all the data they contain to their clients which use as a mounting point a server node part of the global architecture. Contrary to NFS, AFS is a stateful system and so coherency is different from the one found in NFS: when a node opens a file a memory of this operation is kept on the server so when another node access the same file for a write operation a cache invalidation message is sent to all the nodes who opened it. However, this strong coherency implies in high cost in terms of network latency, and thus requires a fast network.

In both cases, the goal of these systems is to provide shared storage for users, which is usually different from the needs of current cluster workloads. Indeed, the development of scientific applications has incurred in new constraints (huge amount of data, level of coherency, fine-grained sharing) on the previous file systems, which led to the design of new storage systems.

A first group of solutions, in order to meet the above needs, might be seen as hardware-based.

File systems such as Sistina's GFS[4] and IBM's GPFS[5] are thought for specialized SAN architectures. Both systems have their data and metadata distributed across the SAN and offer advanced locking and sharing facilities of files. However, the performances of such a system is intimately related to the performances of the storage system underneath. For instance, the GFS handling of coherency relies on an extended SCSI instruction sets. As for GPFS, providing things such as fine-grained coherency by means of software requires a fast and low-latency network like those of the SAN's. Another quite promising new system, LUSTRE, being developed since 2000[6,7] by ClusterFS Inc. aims at satisfying huge storage and transfers requirements as well as offering a Posix semantics. To achieve these goals, clients, meta-data servers (MDS) and object storage targets(OST)[1] are connected by means of a fast network.

[1] Some kind of specialized smart storage.

Unlike GFS and GPFS being based on very specific hardware, Berkeley's xFS[8], as well as LUSTRE, only requires a fast network in order to implement its cooperative multiprocessor cache. This serverless design results from LFS[9] and Zebra[10] file systems. It is built as a totally distributed system where data and meta-data are spread (and may migrate) among the available trusted machines. A different approach is Frangipani[11]/Petal[12] which aims at providing a distributed shared file system (similarly to GFS). The lower-level layer, Petal, implements a logical disk distributed over physical disks. The Frangipani part builds a file system on top of it.

All those systems each offer interesting performances heavily depending on the underlying hardware which doesn't make them well-suited for Beowulf clusters built with common hardware. So another way was developed using purely software solutions and thus, more suited to Beowulf clusters. For example, Intermezzo[13] is a distributed file system relying upon concepts developed in CODA[14], which intends to solve high-availability and scalability issues. A kernel module on the client side handles local and remote operations by means of user-mode helpers, it makes this solution somewhat intrusive since it supposes modifications of the configuration on the client nodes.

On the other hand, the omnipresence of NFS centralized servers has led to develop new designs to improve the throughput without tackling the other specificities, such as temporal coherency, security and fault tolerance. The most common solution has been to aggregate several nodes, either by putting some smartness into the client (Bigfoot-NFS[15], Expand Parallel File System[16]) or by putting some kind of load balancer between the client and the servers (NFS^2[17]).

An alternative is to modify the NFS server by using a meta-data server and storage daemons similarly to the PVFS [18] architecture. Standing from this point, this led us to develop NFSP as a way to offer non-intrusive use and administration.

3 `NFSP` Overview

`NFSP` [1] is a NFS server implementation using techniques developed in PVFS. The architecture falls into three parts: the clients, the meta-data server (referred to as meta-server or `NFSPd` for simplicity) and the storage servers (referred to as `iod(s)`, which stands for I/O daemon(s)).

Figure 2 illustrates the sequence of events occurring when a client wants to access a file. The numbers *1-2-3* and *4-5-6* correspond to clients accessing files. This figure also illustrates the fact that a same physical machine may host a client and a storage entity. For both sequences, the meta-server acts as a fixed point (the client only knows it) as it knows to which storage server it has to forward the request to have it processed.

In the NFS protocol (see figure 1), the first step to manipulate files is always to get a NFS handle on the file. This operation is achieved by sending a `LOOKUP` request to the server which will reply by computing a unique file handle based on

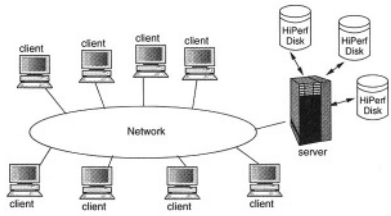


Fig. 1. Architecture of a NFS system

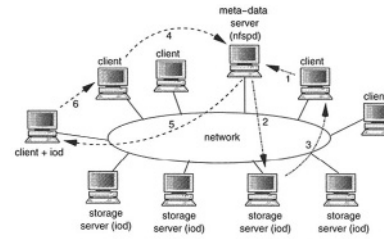


Fig. 2. Architecture of a NFSP system

some file layout properties (inode, device, etc ...). The way the handle is found does not matter to the client as this field is opaque, which means the client has only to use this field as a file identifier. Once the client has managed to get a file handle, the following sequence of events occurs to read the file: 1) it sends a request to the server containing the file handle, the offset, the size and 2) it receives the results sent by the server. As we have chosen to split the server into smaller entities (a meta-data server and storage servers), this scheme is slightly modified (from an overall view):

1. send a request to the server containing the file handle, the offset, the size, ... (See fig. 2, arrow #1 or #4)
2. the server receives the client's request and checks the meta-data it holds,
3. based on some information stored in the meta-data it looks for the storage node that holds the requested data,
4. the request is then modified (a copy of the meta-data required to generate the NFS reply is added) and forwarded to the storage node (see fig. 2, arrow #2 or #5),
5. the storage node processes the request (I/O) it has just received from the server on behalf of the client,
6. the storage node sends the reply to the client (see fig. 2, arrow #3 or #6).

This scheme is inherently synchronous and adds obviously some latency since a network hop is added. Yet, one has to keep in mind that there will most likely be several clients that want to access some files on the server, which permits to have overall performance gains by handling several requests at the same time using multi-threading techniques.

Another source of performances increase in the NFSP model comes from the fact that the time spent to forward the request on the meta-server is much smaller than the time required to process the I/O's. If the meta-data happen to be cached on the meta-server – which is most likely as they are only a few bytes – then the meta-server does not even have to do slow disk I/O. Another performance boost dwells in the fact that by having several storage servers (iods), we have indeed much more cache available than on a single server.

The first implementation of NFSP [1] was done extending an existing user-mode NFS server. Unfortunately we found the performances disappointing due to

a saturation of the metaserver processor and I/O. More precisely, for 18 iods on our cluster, described in section 4, the optimal aggregated bandwidth is roughly 180MB/s but the bandwidth we obtained was only 55MB/S with the CPU of the metaserver used at 100%. In the following parts, two approaches to improve the performances of our prototype will be studied.

4 Multiplying NFSPd (or the Number of Access Points)

Our preliminary work has shown that the main limitation of the current implementation lies in the fact that all the clients use the same meta-server as a mounting point, which causes contention. To bypass this bottleneck, we chose initially to multiply the number of entry points, that is NFS servers. The natural idea, in this approach, is to introduce several NFSPd that would share the same pool of iods. However, the main underlying problem, though eased by the NFS coherency, consists in keeping a synchronization between several meta-data servers. We also tried to keep the level of modifications on the meta-servers as low as possible in order to maintain the good level of performances of the current implementation. This preliminary work has been carried out to implement such a policy by mixing NFS and NFSP exports and re-exports: the basic idea is that a set of iods is handled by a NFSPd server only and that other NFSPd's can mount it as a regular NFS server. By using this technique, it is possible to share several sets of iods with different NFSPd servers while keeping it completely transparent for the user that always use a single mount point. Of course, if performances are mandatory, it is important that a client mounts the NFSPd that will contain most of the data it will access to minimize communication overhead.

Our tests have been launched on the i-cluster (Intel Pentium III 733MHz CPU's - 256MB RAM - 100Mb/s switched network). The bench we use is quite simple: a 1GB file is stored on a NFSP volume and is then read again concurrently by a varying number of clients. The aggregated bandwidth is found by dividing the total amount of data served by the time of completion of the last client. The graph in figure 3 contains three curves illustrating the aggregated bandwidth of a NFSP system composed of 16 iods and successively 1,2 and 4 meta server. As expected, the use of several meta-servers is much more efficient using only one. The simple nfspd curve tends to stagnate then decrease slowly as the number of clients increases. The 2meta-mode curve has almost the same behavior yet the figures are often around at least twice higher. The curve for the optimal bandwidth indicates the maximal peak performance expected (we considered 11.5MB per Ethernet 100 card) and grows till 16 (there need to be at least 16 client to saturate all the iods). The 4meta-mode curve decreases as the number of clients increases. The growing communication cost implied by the message passing between meta servers could explain this. An attempt with 20 iods, 64 clients and 12 meta servers (4 clients per server) gave 80% of the optimal throughput. Nevertheless, in this particular case, each server was saturated and that's a real issue from scalability point of view. Hence, even if a good balance between the number of clients and meta-server nodes could considerably improve

the performance, the meta-server is still the bottleneck. We try to address this issue in the next section which presents a different implementation of the meta-server itself.

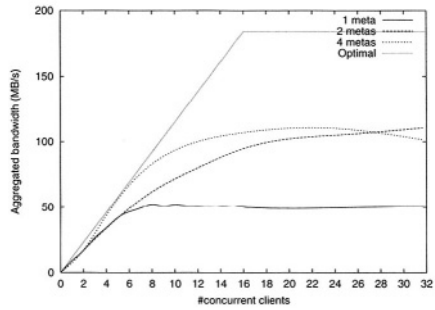


Fig. 3. Aggregated bandwidth – varying number of meta-servers and clients - 16 iods

5 Kernel Implementation

Another way to improve performances is to make a more efficient implementation, especially to avoid time consuming context switches between user and kernel mode by porting the `NFSPd` daemon to kernel. This section describes some specificities and provides some indications on how the issues related to this kernel port have been solved.

The kernel mode port has been devised to alleviate the performance limitations observed with the user-mode port. Indeed, this has been found to be necessary as extended tests have shown that at maximal speeds the user server has its CPU completely saturated. The main reasons for this are the high user-mode overheads (memory copies, I/O system calls and context switches). Moreover, for architectural designs and history, `UNFSPd` is a mono-threaded application and performant servers are nowadays based on a multithreaded paradigm. As the NFS implementation of the Linux kernel was already multi-threaded (for obvious reasons), it has been much more easier to start directly with a multi-threaded architecture for `KNFSP`. To manage several types of exports, the existing set of `nfs-tools` have been extended by setting an unused bit when for a `NFSP` type export. This way, the meta-server is able to handle both NFS and `NFSP` file systems exports at the same time.

We only present results for read operations, as write is mainly limited by the meta-server bandwidth (currently 100Mbps). The bench we use is quite simple: a 1GB file is stored on a `NFSP` volume and is then read again concurrently by a varying number of clients. The aggregated bandwidth is found by dividing the total amount of data served by the time of completion of the last client.

The graph in figure 4 contains three curves illustrating the aggregated bandwidth of a `NFSP` system composed of 16 `iods`. As expected the kernel version

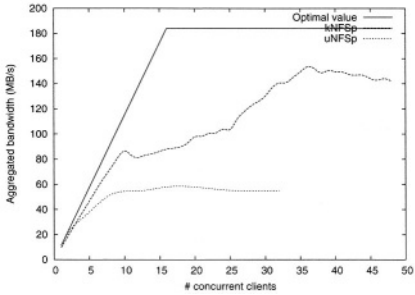


Fig. 4. User-mode vs. kernel-mode server - 16 iods

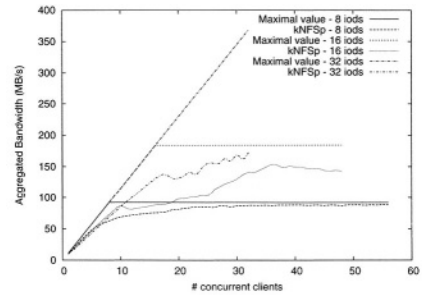


Fig. 5. User-mode vs. kernel-mode server - varying #iods

is much more efficient than the user-mode one. The user-mode curve tends to stagnate then decrease slowly as the number of clients increases. The kernel-mode curve has almost the same behavior yet the figures are often around at least twice higher. The curve for the optimal bandwidth indicates the maximal peak performance expected (we considered 11.5MB per Ethernet 100 card) and grows till 16 (there need to be at least 16 client to saturate all the `iods`). The irregularity of the `KNFSP` curve is due to the timeout policy of the NFS protocol. The slowdown from 10 to 25 clients may match with the first timeouts as the meta-server is being increasingly stressed, then as these wait costs are recovered when there are more clients, it grows again.

The figure 5 illustrates the performances reached as the number of `iods` varies. For 8 `iods` the performances soon become quite good, yet for a higher number the optimal level is much higher. We think this is due to the saturation of the Ethernet switch. Nevertheless the performance increases compared to the user-level version are significant.

6 Conclusion and Future Works

This paper has shown the evolution through which our previous prototype has gone. It also illustrates the costs of a pure user-level implementation of the meta-server against a kernel one. There are currently several improvements underway: NFSv3 port, implementation of the kernel meta-server replication and developing a RAID mode to improve fault tolerance regarding `iods`. Some work is also currently being carried out to add a GRID support to WAN transfers between 2 `NFSP` clusters. Assuming each node of a cluster may be linked to a switch and that they may be IP-connected to another cluster, we expect to obtain efficient cluster-to-cluster data transfers by connecting directly remote and local `iods` thus filling more easily the multi-gigabit pipes within a WAN.

References

1. Lombard, P., Denneulin, Y.: nfsp: A Distributed NFS Server for Clusters of Workstations. In: 16'th IPDPS, IEEE (2002)
2. IETF: NFS: Network file system specification. RFC1094 (1989)
3. Pawlowski, B., Juszczak, C., Staubach, P., Smith, C., Lebel, D., Hitz, D.: NFS version 3, design and implementation. In: Proceedings of the USENIX Summer 1994 Conference. (1994) 65–79
4. Preslan, K.W., Barry, A., Brassow, J., Catalan, R., Manthei, A., Nygaard, E., Oort, S.V., Teigland, D., Tilstra, M., O'Keefe, M.T.: Implementing Journaling in a Linux Shared Disk File System. 8th NASA Goddard Conference on Mass Storage Systems and Technologies in cooperation with the 7th IEEE Symposium on Mass Storage Systems (2000)
5. Schmuck, R.L.F.B.: GPFS: A shared-disk file system for large computing clusters. Proceedings of the 5th Conference on File and Storage Technologies (2002)
6. Braam, P.J., Zahir, R.: Lustre Technical Project Summary (Attachment A to RFP B514193 Response). Technical report (2001)
7. Cluster File System Inc.: LUSTRE: A Scalable, High-Performance File System (2002)
8. Anderson, T.E., Dahlin, M.D., Neefe, J.M., Patterson, D.A., Roselli, D.S., Wang, R.Y.: Serverless Network File Systems. Computer Science Division, University of California at Berkeley, CA 94720 (1995)
9. Ousterhout, J., Douglis, F.: Beating the I/O Bottleneck : A Case for Log-Structured File Systems. Computer Science Division, Electrical Engineering and Computer Sciences, University of California at Berkeley, CA 94720 (1992)
10. Hartman, J.H., Ousterhout, J.K.: Zebra striped network file system. Computer Science Division, Electrical Engineering and Computer Sciences, University of California at Berkeley, CA 94720 (1993)
11. Thekkath, C.A., Mann, T., Lee, E.K.: Frangipani: A Scalable Distributed File System. In: Proceedings of the 16th ACM Symposium on Operating Systems. (1997)
12. Lee, E.K., Thekkath, C.A.: Petal: Distributed Virtual Disks. In ACM, ed.: Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems. (1996) ASPLO-7.
13. Braam, P.J., Nelson, P.A.: Removing Bottlenecks in Distributed Filesystems : Coda and Intermezzo as examples. Carnegie Mellon University and Western Washington University (1999)
14. Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C.: CODA: A Highly Available File System for a Distributed Workstation Environment. IEEE Transactions on computers, Vol 39, N 4 (1990)
15. Kim, Minnich, McVoy: Bigfoot-NFS: A Parallel File-Striping NFS Server (1994)
16. Calder n, A., Garc a, F., Carretero, J., P rez, J.M., Fern ndez, J.: An Implementation of MPI-IO on Expand: A Parallel File System Based on NFS Servers. In: 9th PVM/MPI European User's Group. (2002)
17. Muntz, D.: Building a Single Distributed File System from Many NFS Servers. Technical Report HPL-2001-176 (2001)
18. Carns, P.H., Ligon III, W.B., Ross, R.B., Thakur, R.: PVFS: A parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, USENIX Association (2000) 317–327

Testability of Distributed Objects*

Magdalena Sławińska

Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology
Narutowicza 11/12, 80-952 Gdansk, Poland
magg@eti.pg.gda.pl

Abstract. Assuring testability of distributed objects is especially important because today testers usually view object applications from the perspective of *native* (with the full access to the source code) and *foreign* objects (only on-line access to object methods). Taking it into account, the paper defines three levels of object accessibility, i.e., limited, partly limited and full access levels. It discusses possibility of testing objects at each level. The article reviews existing mechanisms for object testability in the context of the current CORBA standard. The paper also discusses possibility of increasing the testability degree by adding means for observability of the internal states of objects and using existing mechanisms for monitoring communication actions among distributed objects.

1 Introduction

Modern distributed object-oriented applications should provide high quality to fulfil users' requirements. However, growing size and complexity of developed software makes this goal difficult to achieve. One solution to that problem is to provide mechanisms for *testability* of created software products.

Although novel distributed object-oriented platforms provide some mechanisms for testability (interceptors, monitors, loggers), they can be insufficient for testing concrete applications. For instance, sometimes it is useful for a tester to ask objects under test for their state. If the object has a diagnostic interface, this information probably can be accessed, otherwise not. Assuring standard diagnostic interfaces is of greater significance in the context of complexity and heterogeneouity of components which constitute novel software applications.

This paper reviews existing mechanisms for assuring testability of distributed objects in OMG's *(Object Management Group)* standard *Common Object Request Broker Architecture* (CORBA) [1] and proposes new mechanisms for improving testability of foreign objects in distributed applications. Section 2 describes native and foreign objects in a distributed object-oriented application. In Section 3 three levels of object availability are presented. Section 4 reviews existing mechanisms for testability in CORBA. In Section 5 additional techniques for testability are presented and discussed. Finally, the paper is summarised and concluded in Section 6.

* Funded in part by the State Committee for Scientific Research (KBN) grant T-11C-004-22

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 413–418, 2004.
© Springer-Verlag Berlin Heidelberg 2004

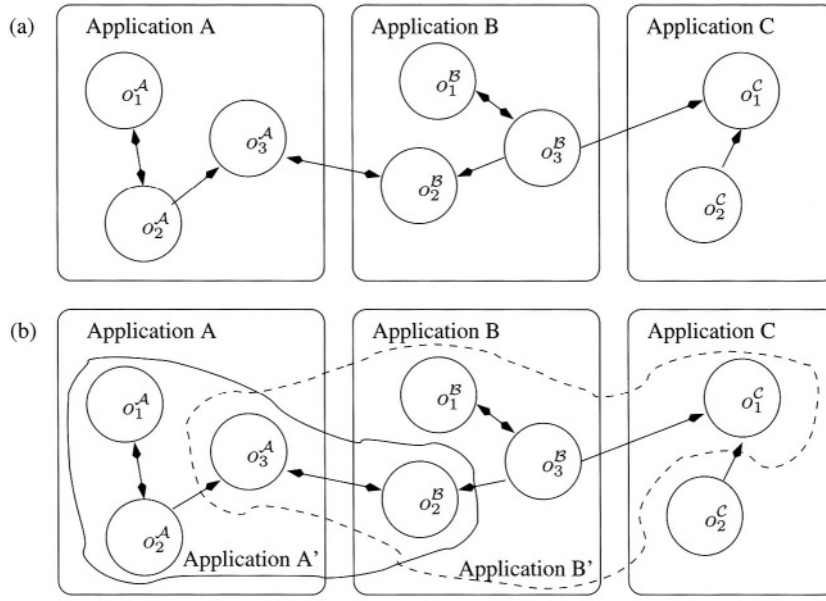


Fig. 1. Distributed object-oriented applications without considering interactions with other applications (a) and considering such interactions (b). Arrows denote object interactions e.g. object $o_3^{\mathcal{A}}$ interacts with object $o_2^{\mathcal{B}}$ and $o_2^{\mathcal{B}}$ interacts with $o_3^{\mathcal{A}}$ (two arrows) but $o_3^{\mathcal{B}}$ interacts with $o_1^{\mathcal{C}}$ and $o_1^{\mathcal{C}}$ does not interact with $o_3^{\mathcal{B}}$ (the single arrow).

2 Native and Foreign Objects

It is assumed that application $\mathcal{A}$ consists of a dynamic number of objects $o_1^{\mathcal{A}}$, $o_2^{\mathcal{A}}, \ldots, o_n^{\mathcal{A}}$ where $n = 1, 2, \ldots$ and $n(t) \neq const$, where t denotes time. Objects interact with other objects by invoking their methods. It is designated by arrow $\rightarrow$. Figure 1(a) shows three applications, namely $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, at time T. Since they are distributed applications, objects can be located on different machines. They communicate over a middleware layer (for clarity reasons not depicted in the figure (e.g. it can be *Object Request Broker* – ORB)). Suppose that applications $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ were developed by company A, B and C respectively, and each company has full access only to its source code, i.e., company A has access to application $\mathcal{A}$ and has no access to the source code of applications $\mathcal{B}, \mathcal{C}$ but application $\mathcal{A}$ can use services of applications $\mathcal{B}, \mathcal{C}$ over well-defined interfaces. Thus, it is reasonable to say that: $\mathcal{A} = \{o_1^{\mathcal{A}}, o_2^{\mathcal{A}}, o_3^{\mathcal{A}}\}$, $\mathcal{B} = \{o_1^{\mathcal{B}}, o_2^{\mathcal{B}}, o_3^{\mathcal{B}}\}$ and $\mathcal{C} = \{o_1^{\mathcal{C}}, o_2^{\mathcal{C}}\}$. However, from the dynamic point of view of application $\mathcal{A}$ object $o_3^{\mathcal{A}}$ interacts with object $o_2^{\mathcal{B}}$ (i.e. $o_3^{\mathcal{A}} \rightarrow o_2^{\mathcal{B}}$) which belongs to application $\mathcal{B}$. So in natural way objects can be divided into two groups: *native objects* and *foreign objects.* In this context all objects listed in $\mathcal{A}$ are native objects from the perspective of company A since it is their owner, and object $o_2^{\mathcal{B}}$ is the foreign object because it is required for application $\mathcal{A}$ to function properly. So from the point of view company:

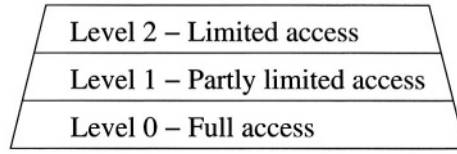


Fig. 2. Levels of object accessibility

- A: native objects are $\{o_1^{\mathcal{A}}, o_2^{\mathcal{A}}, o_3^{\mathcal{A}}\}$ and foreign ones: $\{o_2^{\mathcal{B}}\}$
- B: native objects are $\{o_1^{\mathcal{B}}, o_2^{\mathcal{B}}, o_3^{\mathcal{B}}\}$ and foreign ones: $\{o_3^{\mathcal{A}}, o_1^{\mathcal{C}}\}$
- C: native objects are $\{o_1^{\mathcal{C}}, o_2^{\mathcal{C}}\}$ and no foreign ones.

In fact, during execution there are three applications bound by interactions, namely $\mathcal{A}' = \{o_1^{\mathcal{A}}, o_2^{\mathcal{A}}, o_3^{\mathcal{A}}, o_2^{\mathcal{B}}\}$, $\mathcal{B}' = \{o_1^{\mathcal{B}}, o_2^{\mathcal{B}}, o_3^{\mathcal{B}}, o_3^{\mathcal{A}}, o_1^{\mathcal{C}}\}$, and $\mathcal{C}$, as it is depicted in Figure 1(b). Notice that the access to the source code of objects in application $\mathcal{A}'$ and $\mathcal{B}'$ is different.

3 Access Levels of Distributed Objects

Information about object behaviour depends on the level of object accessibility. There can be defined three levels of object accessibility. They are presented in Figure 2. The area of each access level illustrates the amount of information of a given object the tester is able to get.

3.1 Level 2 – Limited Access

The uppermost level, named *Level 2 – Limited access,* concerns objects with general access. In order to use an object, programmers should know at least the object reference and the prototype of the method they want to invoke (the method name, parameter types, returned values). In CORBA *Interface Definition Language* (IDL) describes interfaces to distributed objects. References to those objects are published in such a way that they are commonly known (e.g. dumped to the file in the commonly known place)[1]. At the limited access level all objects can be regarded as foreign objects with general information concerning the object reference and lists of operations which can be called. For example in CORBA it is possible to get the description of the interface of a given object if a tester has the reference of the object (by invoking standard method `get_interface()` defined for each CORBA object[1]).

3.2 Level 1 – Partly Limited Access

Level 1 – *Partly limited access* is the intermediate level between the limited level and full access one. At this level a compiled code of the object is available to the tester, e.g., `*.class` files in Java or executable ones.

In fact, there can be distinguished two sub-levels in Level 1 depending on whether additional information has been included during compilation of source files. For instance, the GNU C/C++ compiler – GCC/G++ puts additional debugging information to object files if source files are compiled with option '`-g`' [2]. Thus, at Level 1 executable files can contain debugging information or not. For example, in case of GDB [3] – the popular GNU symbolic debugger supporting languages C and C++, including debugging information enables the tester to connect remotely to the object being debugged with the purpose of effective interactive debugging [4].

Level 1 refers to foreign objects (with debugging information included into executables or without such information).

3.3 Level 0 – Full Access

Finally, there are situations where testers have full access to the source code of application components. In that case they are free to inspect the source code to look for bugs. They can also make any necessary modifications to the source code. It is up to the tester to recompile the source code with options including debugging information or not. This level concerns native objects.

Fortunately, the CORBA standard provides some mechanisms for testability of distributed objects' behaviour.

4 Testability Mechanisms for Distributed Objects

This paper defines term *testability* as ability to reveal failures in software during testing [5] [6] [7] [8] [9]. CORBA provides some mechanisms for testability of distributed objects like interface and implementation repositories, and interceptors.

- *Interface Repository* (IR) – it contains the object hierarchy with detailed IDL descriptions of registered interfaces (operations, parameters, exceptions, types, etc.) [1]. IR allows for dynamic modifications of registered interfaces. It is important that registering and unregistering interface descriptions do not require any modifications of the source code of the registered objects. The tester can always write a client program that puts to or gets from IR desired information.
- *Implementation Repository* (ImplR) – it is similar to IR but ImplR concerns object implementations (not object interfaces). Since it is not a subject of the CORBA standard[1], it is vendor specific. This paper considers Borland Visibroker with Borland Enterprise Server Visibroker Edition 5.1 for the CORBA implementation [10]. In case of Visibroker, the actual ImplR is *Object Activation Daemon* (OAD). OAD contains: interface names, instance names, full path name of the server implementation's executable, activation policy, reference data and the list of arguments and environment variables passed to the server at activation time. In order to make use of OAD, the object implementation should be created with structure `CreationImplDef` and registered in OAD. It must be done on the server side.

– *Interceptors* – they enable intercepting communication between clients and CORBA objects [1]. Testers can investigate (and modify) operations invoked by clients on CORBA objects and results returned by CORBA objects to clients *(client interceptors)* and operations performed by CORBA objects and results returned to clients (*server interceptors*). In order to use interceptors, firstly they must be loaded. In case of Visibroker installing interceptors on the server side requires implementing two classes (`Loader` and `ServerInterceptor`) and initializing server ORB with property `org.omg.PortableInterceptor.ORBInitializerClass.Loader` set to class `Server`.

Repositories provide information about "external" structures of objects while interceptors allow for getting some information about object "internals". However, if testers are at Level 2 of object accessibility (with foreign objects) they have no possibility for using server interceptors (unless the programmer had incorporated an interceptor to the application).

Although interceptors provide some information about communication, unfortunately they do not provide direct information about the internal state of the CORBA object.

5 Making Distributed Objects More Testable

In order to observe states of foreign and native objects it would be reasonable to add two methods to standard `CORBA::Object` interface: `get_state()` and `set-State()`. Currently such methods are defined for interface `Checkpointable` in *Fault Tolerant CORBA* [1]. Objects which inherit from `Checkpointable` may log their current state but it is not obligatory. It is because logging mechanisms (in case of `Checkpointable`) are assumed for group processing associated with removing members from a group or adding newcomers to a group. The priority is not the history of object state changes but the most up-to-date and consistent state of all group members. Apart from that not all of application objects in real software are required to be fault tolerant. However, it would be useful for the testability reason if each CORBA object could show its state.

If `CORBA::Object` had operations `get_state()` and `set_state()`, clients of any CORBA object could ask for object state information. The programmer of a CORBA object would be responsible for implementing those methods. It is reasonable since the programmer knows exactly what data constitute the object state. In this way it would be possible to query remote objects for their state even if they are foreign objects.

Availability of information about object states simplify writing test scripts. Objects can be regarded as automata where invoked methods modify object states. With standard operations `get_state()` and `set_state()` test scripts can simply compare the observed state with the predicted state. If they are distinct the bug should be signalized. Since these activities can be automated testability of distributed objects is increased. Moreover, if each object can record its history of state changes to a log and interceptors can record information about

communications of distributed objects it would be possible to recover a given (potentially erroneous) situation and replay it in order to find a bug. This would make distributed object-oriented applications more testable especially at Level 1 and 2. Currently the work on the prototype system that enables diagnosability of foreign objects with state recording and replaying the recorded scenario is carried on, however the prototype system is not ready yet.

6 Conclusions

As it is presented in the paper modern distributed object-oriented applications consist of objects scattered over the network. Objects can be divided generally into two groups: foreign and native ones. Taking it into account three levels of accessibility can be distinguished. Level 2 concerns accessibility of foreign objects while Level 0 concerns native ones. Level 1 is an intermediate access level where testers have access to executables that can be exercised with a debugger (assuming that debugging information was included to the compiled file).

CORBA provides some mechanisms for getting information of scattered objects like IR, ImplR or interceptors. However they describe rather the external environment of distributed objects. Some mechanisms are required that will be available for all distributed objects in order to make them more testable. This paper proposes two such mechanisms. The first mechanism concerns equipping objects with two methods for object state management. The second mechanism concerns using interceptors for communication monitoring.

References

1. *Common Object Request Broker Architecture: Architecture and Specification, v3.0.* Object Management Group `http://www.omg.org`, December 2002.
2. J. Menapace, J. Kingdon, and D. MacKenzie, *The "stabs" debug format.* Free Software Foundation, Inc., 2002.
3. R. M. Stallman and R. H. Pesch, *Debugging with GDB – The GNU Source-Level Debugger.* Free Software Foundation, Inc, ninth ed., December 2001.
4. M. Sujecka and B. Wiszniewski, "Remote debugging of CORBA objects," in *Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing,* pp. 396–401, IEEE Computer Society, February 2001.
5. H. Krawczyk and B. Wiszniewski, *Analysis and Testing of Distributed Software Applications.* Industrial Control, Computers and Communications Series, Research Studies Press LTD., 1998.
6. S. Jungmayr, "Design for Testability," in *Proceedings of CONQUEST 2002,* (Nuremberg, Germany, Sept. 18th-20th, 2002), pp. 57–64, September 2002.
7. J. Voas and K. Miller, "Software testability: investing in testing," *EuroStar 96, Amsterdam,* December 1996.
8. B. Pettichord, "Design for Testability," in *Pacific Northwest Software Quality Conference,* (Portland, Oregon), October 2002.
9. J. M. Voas and K. W. Miller, "Software Testability: The New Verification," *IEEE Software,* vol. 12, pp. 17–28, May 1995.
10. *VisiBroker Developer's Guide.* Borland Software Corporation, 2002. Version 5.1.

Dynamically Reconfigurable Scientific Computing on Large-Scale Heterogeneous Grids

Boleslaw Szymanski[1], Carlos Varela[1], John Cummings[2], and Jim Napolitano[2]

[1] Department of Computer Science,
{szymansk,cvarela}@cs.rpi.edu
[2] Department of Physics, Applied Physics, and Astronomy
Rensselaer Polytechnic Institute, Troy NY 12180, USA,
{cummij,napolj}@rpi.edu,
http://www.cs.rpi.edu/

Abstract. Many scientific applications require computational capabilities not easily supported by current computing environments. We propose a scalable computing environment based on *autonomous actors.* In this approach, a wide range of computational resources, ranging from clusters to desktops and laptops, can run an application programmed using actors as program components in an actor language: *SALSA.* SALSA actors have the ability to execute autonomously in dynamically reconfigurable computing environments. We develop the corresponding "Internet Operating system" *(IO)* to address run-time middleware issues such as permanent storage for results produced by actors, inter-actor communication and synchronization, and fault-tolerance in a manner transparent to the end-user. We are using this worldwide computing software infrastructure to solve a long outstanding problem in particle physics: *the missing baryons,* originally identified over thirty years ago.

1 Introduction

Motivated by the needs of many interesting scientific problems, we are developing a solution to finding an inexpensive and abundant computational resource. We intend to target the large collections of laptops, desktops, workstations and clusters intermittently available via the Internet and propose to provide a system that will utilize them efficiently by employing task migration, load balancing [15], and replication at the system level [12]. We also offer users a programming language with convenient and high-level abstractions enabling dynamic system reconfiguration. In addition, we address the issues of portability and security by executing the application within the Java Virtual Machine [11]. To achieve these goals, we are using actors that enable us to separate execution environment issues from application programmer concerns. In that, we differ both from traditional grid computing technologies [6,9], that do not provide explicit programming and system support for dynamic reconfiguration –therefore, being restricted to mostly static networks of clusters– and from traditional worldwide computing applications (e.g. [13,7]) that target embarrassingly parallel computation, usually with a single application support.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 419–430, 2004.
© Springer-Verlag Berlin Heidelberg 2004

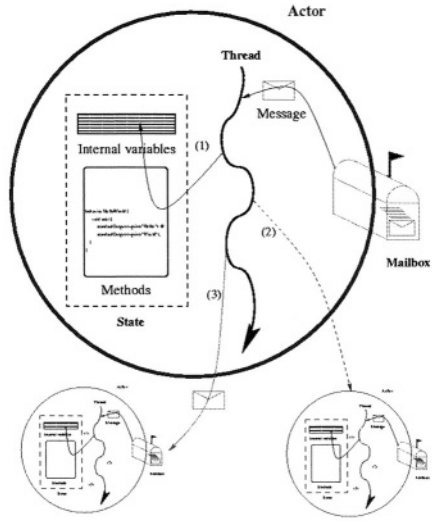


Fig. 1. The actor model of computation

2 Autonomous Agents over Adaptive Grids

Scientific computing is turning to standard computer networks in its search for high performance computing resources, for their price-performance ratio, and their self-upgrading nature. The main challenge of this approach arises from the fact that computer networks are very dynamic and heterogeneous entities with constant node additions, failures, and changes in communication topology. Therefore, there is a need for programming paradigms, models, languages, and software systems that facilitate building dynamically reconfigurable high-performance distributed systems.

2.1 SALSA Platform

SALSA (Simple Actor Language, System, and Architecture) [15] is an actor programming language (see Figure 1) with high-level constructs for remote messaging, universal naming, migration, and coordination. An actor [1] is a unit of encapsulation for both a state (procedures and data) as well as processing of such a state (a thread of control). All communication between actors is through asynchronous message passing. While processing a message, an actor can carry out any of three basic operations: (1) alter its state, (2) create new actors, or (3) send messages to peer actors. Actors are therefore inherently independent, concurrent, and autonomous, which enables efficiency in parallel execution and facilitates mobility [2].

SALSA programs are compiled into Java code [8], leveraging the existence of virtual machine implementations in multiple heterogeneous platforms and operating systems. We therefore view a heterogeneous network of physical machines as a homogeneous network of Java virtual machines. While Java's main perceived drawback is its lack of performance –due to its bytecode interpretation overhead– recent advances in just-in-

time (JIT) and adaptive compilation, make Java a very attractive platform for scientific applications [4].

The World-Wide Computer (WWC) [2] run-time architecture consists of naming servers and virtual machines running as Java applications on different Internet nodes. The virtual machines, called *theaters,* provide an environment for execution of universal actors using local resources. High-level programming language abstractions enable actors to create remote communication links with peer actors running on other WWC theaters. Furthermore, actors can easily migrate with their full state to other WWC theaters as they become available, supporting load balancing and scalability. The naming servers keep track of universal actor locators, so that communication remains transparent to actor location and migration.

2.2 Autonomous Actors

We create an extension to actors, *autonomous actors,* that dynamically and opportunistically migrate to nearby theaters, making use of profiled information about their computing and communication patterns [5]. The goal is to enable autonomous actors to find their optimal execution environment without the need for the application programmers to explicitly embed in their codes procedures for load balancing, fault-tolerance, replication, and coordination algorithms.

When a system is composed of mobile actors, it can be reconfigured arbitrarily, as long as all its used resources are ubiquitous. Autonomous actors extend actors by:

- profiling computing and communication performance,
- including a *satisfaction threshold* to trigger migration, and
- introducing message priorities –to enable, e.g., a migration message to take precedence over other messages.

We develop an "Internet Operating system" (IO) middleware layer to help us evaluate different adaptive data and task migration and replication algorithms. If an actor is processing messages at a speed faster or equal to the speed that it is receiving them, it can maintain a constant mail-queue. This means that the actor's current location has enough system resources to handle its tasks. However, when this condition is not met, the messages in an actor's mailbox begin to accumulate. In this case, the unsatisfied actor attempts to migrate to a more appropriate location. Likewise, if new resources become available in the network, or resources are going to be temporarily unavailable, an autonomous actor can choose to migrate to improve overall system performance.

Figure 2 displays a high-level architecture for the proposed IO system. SALSA programs are at the application layer –a program creates and initiates a set of actors which compute and collect results. IO coordinates the mapping of these application-level actors into actual run-time environments (theaters) at the system layer to support autonomous reconfiguration.

In short, our project implements a middleware system based on actors implemented in SALSA. This middleware supports computations that can be abstracted as a multi-level and dynamic computational tree. In this paradigm the application domain is iteratively divided into subdomains, each creating a branch rooted in the parent domain. Each leaf

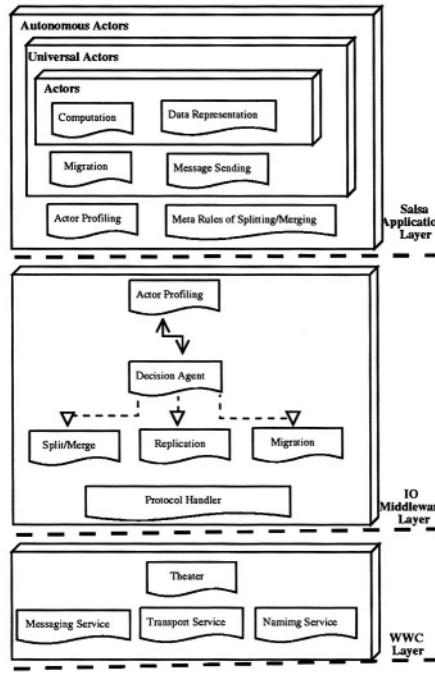


Fig. 2. System Layers

can either span a subtree or execute its program over its subdomain. Each subdomain computation can be iterated over with synchronization of each iterative step. The communication requirement that allows for efficient execution in such an environment is of course technology-dependent; however, given the current ratio of communication to computation speed, there is a whole class of important applications that will execute efficiently in such a scheme.

3 Strategies for Autonomous Load Balancing

In this section, we describe various methodologies for load balancing that vary by the amount of profiling done and the complexity of the decision agents. The simplest decision agents take into account the load of the individual theaters and autonomous actors, while the more complex agents consider additional factors such the network and actor topologies. All the network protocols are peer-to-peer in nature to allow for maximum scalability. In all cases, a theater joins the autonomous network by registering with a peer server and receiving addresses of other peers in the network from it. Peer servers are not centralized, as many may serve addresses of peers for a single network.

Before describing the strategies for autonomous load balancing, the following concepts are used to describe the attributes of an actor system over a heterogeneous network.

Actor Satisfaction. Actor satisfaction is a measure of an actor's ability to process and send messages. If an actor is not satisfied, it cannot process messages as quickly as

it is receiving them. This includes the cost of message sending, because processing a message also involves sending messages. When an actor is unable to handle the load of messages it is receiving, the size of its message queue begins to increase. An actor with an increasing message queue is unsatisfied.

Theater Load. Every theater hosts a group of active actors. A theater is considered lightly loaded if all its actors are satisfied, whereas a theater is considered heavily loaded if at least one of its actors is not satisfied.

3.1 Load-Sensitive Random Stealing (RS)

The simplest strategy is based on random work stealing, a simple but effective algorithm described by [3]. We modified this algorithm to work in a peer-to-peer network by randomly propagating a random steal packet over the network. A lightly loaded theater chooses a neighbor at random and sends it a steal packet. This continues from theater to theater until a candidate for migration is chosen or the packet's time to live has been reached. When either occurs a notification is sent back to the originating theater. This prevents a theater form performing multiple steals simultaneously. One benefit of random steal propagation is that it avoids costly broadcasts to the network, reducing the impact of the middleware on the application. In RS, a peer theater finds its first unsatisfied actor (if one exists) and selects that as its candidate for migration. Also, since only lightly loaded theaters send steal packets, with high loads the overhead for RS becomes almost non-existent.

3.2 Actor Topology Sensitive Random Stealing (ARS)

Actor topology sensitive random stealing builds on the previous strategy by using additional profiling information. Actors monitor the number of messages they send to remote theaters, allowing this strategy to find a actor placement in the network according to the communication patterns in the application. This approach enables actors with high frequencies of communication to be co-located or located in nodes with low latencies, according to the results of a decision function.

The decision function estimates the increase in the performance of an actor if it migrates to a specific foreign theater. Random steal packets now also contain the available processing power of their origin theater. Let $\Delta(l, f, a)$ denote the normalized increase in performance of actor a that results from migrating a from the local theater l to the foreign theater f. The normalized increase in performance is determined by the normalized increase in communication, $\Delta_c(l, f, a)$ and the normalized increase in processing, $\Delta_p(l, f)$ that would occur in that actor after this migration. Migration only happens when the estimated change in throughput is positive ($\Delta(l, f, a) > 0$).

The following equations illustrate how the decision function is evaluated.

$$\Delta(l, f, a) = \Delta_p(l, f) + \Delta_c(l, f, a) \tag{1}$$

where

$$\Delta_p(l, f) = \frac{\mathcal{P}(f) - \mathcal{P}(l)}{\mathcal{P}(f) + \mathcal{P}(l)} \tag{2}$$

and

$$\Delta_c(l, f, a) = \frac{\mathcal{M}(f, a) - \mathcal{M}(l, a)}{\mathcal{M}(a)} \tag{3}$$

We used the following notation: $\mathcal{M}(t, a)$ denotes the number of messages communicated between an actor a and theater t, while $\mathcal{M}(a)$ denotes the total number of messages sent by actor a. $\Delta(l, f, a)$ stands for the normalized change in actor performance obtained by migrating actor a from theater l to theater f, while $\Delta_c(l, f, a)$ and $\Delta_p(l, f)$ denote actor performance change due to communication and message processing, respectively. $\mathcal{P}(t)$ is the processing power available in theater t.

This decision function was chosen because while it is not very precise, it does provide very reasonable results with a minimal amount of overhead. It places a strong emphasis on inter-actor communication and tries to co-locate tightly coupled actors (actors which frequently communicate). Both the difference in processing and communication have been normalized to a value between –1 and 1, and in the future we could add weights to these values based on empirical research.

3.3 Network Topology Sensitive Random Stealing (NRS)

In addition to resource availability, NRS takes into consideration the topology of the network. In the IO network a peer might belong to local, regional, national, or international clusters [10]. In these cases, while bandwidth may be high, latency will play a large factor in the throughput of messages between theaters. NRS locates tightly coupled actors close together in the IO network, but allows loosely coupled actors to migrate more freely, as they do not need this restriction.

NRS classifies its neighbors into four groups: local, regional, national and international. These groups are classified into locales by the following ping times [10]:

- Local: 10 ms or less
- Regional: 11 ms to 100 ms
- National: 101 ms to 250 ms
- International: 251 ms and higher

The algorithm then proceeds similar to cluster-aware random stealing described by [14]. Random steal packets specify which locale they are to travel. A theater first selects a local peer randomly and sends a local random steal packet. A theater will only propagate a steal packet to its specified locale. If a local random steal packet fails (the theater receives a terminated packet without an actor), the theater will then attempt a regional random steal, and so on.

Using this method to propagate random steal packets through the network keeps groups of coupled actors close together in the network. NRS uses the previously mentioned methods for determining the best candidate actor when a random steal packet reaches a theater, thus NRS comes in two versions: RS and ARS.

4 Preliminary Results

We ran a series of tests on our IO system using a manual round robin placement of actors (RR), peer-to-peer random stealing (RS) and the actor topology sensitive random stealing (ARS) strategies.

We ran four simulations each pertaining to a level of inter-actor communication. The unconnected actor graph had actors simply process messages over and over, with no inter-actor communication. The sparse actor graph linked actors randomly, providing a moderate amount of inter-actor communication. The tree simulation linked actors in a tree structure, for a higher amount of inter-actor communication. Lastly, the hypercube provided a very high amount of inter-actor communication. (see Figures 3 and 4. We compared throughput of RS and ARS to manual load balancing to measure the overhead that the IO middleware incurred on the computation. All actors were loaded in a round robin fashion across the eight theaters, then were allowed to compute until their throughput leveled off. Throughput is the number of messages processed by all actors in a given amount of time – the higher the throughput, the faster a computation is running.

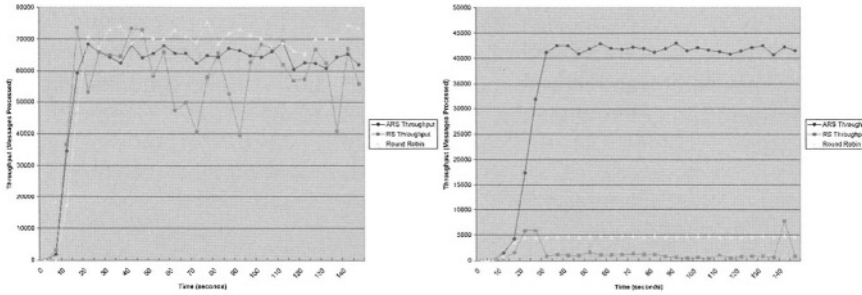


Fig. 3. Unconnected and Sparse Graph Actor Topologies.

Figure 3 shows that both ARS and RS imposed a minimal amount of overhead for the simulation, as a round robin placement of actors is the optimal load balancing solution for an unconnected graph of actors in a homogeneous network, and the round robin placement imposed no middleware overhead. ARS and RS performed comparatively to RR in this test. On the more communication-bound simulations (see Figure 4), ARS outperformed both the manual load balancing and RS. On a sparsely connected graph, ARS performed superbly, bringing throughput to nearly the level of an unconnected graph. In all simulations involving inter-actor communication, ARS highly outperformed RR and RS, showing that the co-location of actors significantly improves message throughput. RS was shown to be too unstable in all these simulations and did not outperform either RR or ARS. Our conjecture is that because the Java thread scheduling mechanism is not fair, actors are found to be unsatisfied when they are actually not, leading to the unstable migration behavior of actors when IO uses RS.

To show how IO can handle a dynamically changing network, the same simulations were ran on a changing network of peer theaters. The simulations were loaded entirely

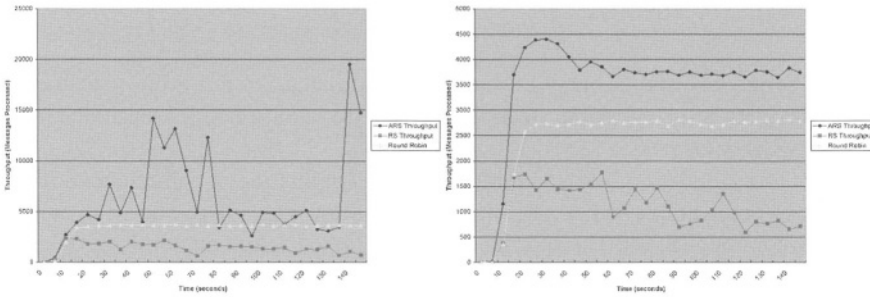


Fig. 4. Tree and Hypercube Actor Topologies.

onto one peer theater, then every 30 seconds an additional peer theater was added to the computation. After eight peer theaters had joined the computation, IO was allowed to run for two minutes to balance the load, after which a peer theater was removed every 30 seconds, until the computation was entirely running on the last peer theater added to the computation.

With the unconnected graph join/leave simulation (see Figure 5), both RS and ARS performed well in distributing the load across the peer theaters (see Figure 6), and increased the throughput by a factor of about six when all eight theaters had joined the simulation. The addition and removal of peer theaters shows that IO can rebalance load with removal and addition of nodes without much overhead.

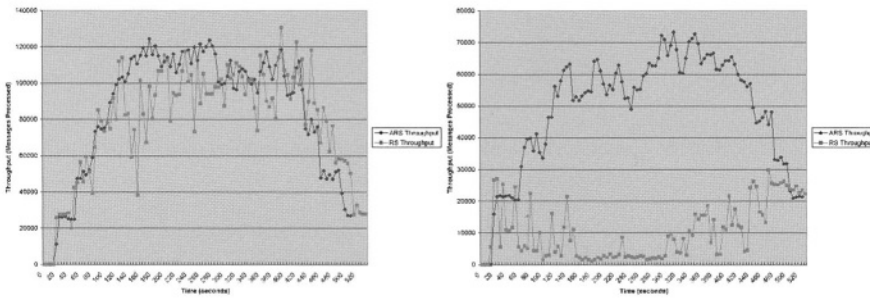


Fig. 5. Unconnected and Tree Graph Actor Topologies on a Dynamic Network.

The graphs of actor placement (see Figure 6) show that while both ARS and RS managed to distribute the actors evenly across the network of theaters, ARS co-located actors more appropriately according do their connectivity, significantly improving overall throughput.

These preliminary results show that the IO system with ARS performs well in most situations for load balancing of a distributed actor system. While the more traditional strategy of random stealing does not fare so well in an autonomous system of actors, a more intelligent strategy can exploit the properties of the actor model to provide autonomic solutions for load balancing across a dynamic network. The results also show

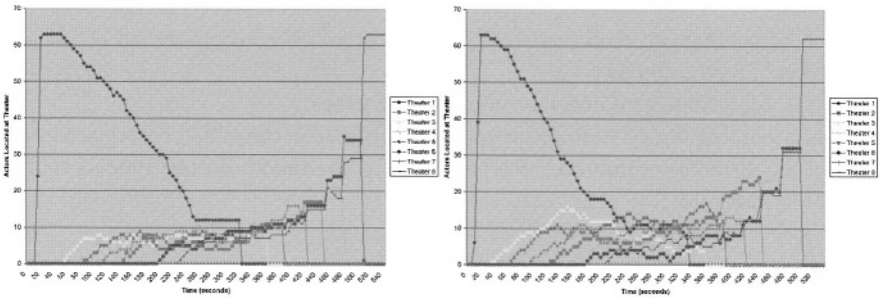


Fig. 6. Actor Distribution for Tree Graph Actor Topology with ARS and RS.

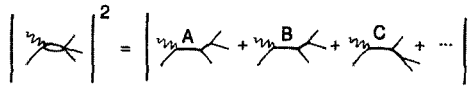
that IO can handle the addition and removal of nodes from a computation without any central coordination, a necessity for large dynamic heterogeneous networks.

5 Missing Baryons

Baryons are the family of spin one half states that the well known proton and neutron belong to. In spite of the fact that at present we don't know how to calculate the baryon spectrum with QCD, properties such as symmetry rules can be used to make some general predictions. Baryon states are formed from three quarks bound strongly, and treating the three quarks equally, it is not difficult to write down a list of allowed states in terms of fundamental properties such as the states angular momentum and parity. Interestingly, this list includes quite a few states that have not been observed. Since many of these extra predictions seem to be states that would have been seen by previous and current experiments, they are referred to as "Missing Baryons". The measurement of the baryon spectrum is a extremely helpful, if not necessary, component to understanding Quantum Chromodynamics, QCD, a field theory that describes the strong nuclear force.

The measurement of baryon spectra is the measurement of quantum mechanical states. This means we actually observe a probability distribution, and so to make an accurate measurement we must observe many interactions. To study states produced in a photon-proton interaction, a beam of photons is directed at a target containing many protons. Each photon in the beam has a chance to excite the proton into another state of the baryon spectrum. If such an interaction happens, the produced state very quickly decays into more stable particles which can be seen in a detector. By studying the distribution of these detected particles angles and energies, the quantum mechanical properties of the produced, short-lived, states can be determined.

There are many possibilities for short-lived state, and since the system is described by quantum mechanics, these are described by interfering amplitudes. A technique known as Partial Wave Analysis is used to write down a general expression for the intensity distribution in terms of the various possibilities for this short-lived intermediate state. The amount of the different intermediate states is then varied by a fitting program to match the observed distributions. Thus the intensity distribution, expanded this way, can

be represented pictorially as 

where A, B, C,... represent different intermediate states.

A maximum likelihood fit finds the mix of intermediate states that best describes the data. The usual trick is to minimize the negative log likelihood; for this type of analysis the function looks like:

$$-\ln(\mathcal{L}) = \sum_{i}^{n} \left| \sum_{\alpha}^{N} \mathrm{V}_{\alpha} \psi_{\alpha}(\tau_i) \right|^2 - n \sum_{\alpha\alpha'} \mathrm{V}_{\alpha}^{*} \Psi_{\alpha\alpha'} \mathrm{V}_{\alpha'} \tag{4}$$

where α is summed over the N intermediate states, the V's are the fit parameters – the amount of state α required to describe the data. The first sum over i is over the number of interactions observed, n.

Since missing baryon states may be "small" rather than "missing", a detailed search for them requires many interactions. Data samples on the order of $n = 10^6$ are already available to be analyzed, and future experiments expect to increasing this by *two orders of magnitude.* Typically *N,* the number of intermediate states required, is in the range of 10-100. The numbers in the likelihood calculation are all complex due to the quantum mechanical nature of the problem. So the calculation of the likelihood for a single set of parameters V, requires $n(N+1) + 2N^2$ complex multiplications and nN complex additions. Consider that a minimization may require thousands of evaluations of $-\ln(\mathcal{L})$ in the search for the optimal set of parameters, and the net result is that complete studies are limited by available computational resources. Our model for grid evaluation of this function is essentially the trivial one: the large sum over n data points is split into m smaller sums which are delegated to actors in the World-Wide Computer.

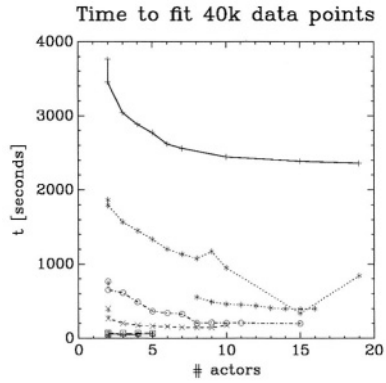


Fig. 7. Timing measurements for χ^2 fits to 4×10^4 data points

We have recently completed development of a maximum likelihood fitter for partial wave analysis in SALSA and are beginning to verify and performance analyze the package. Some preliminary results of measurements done using an early version which

performed χ^2 fits is shown in Figure 7. A large number of data points, 4×10^4, were fit. The time to converge decreases like $1/m+$ constant with increasing m, the number of actors (computers) participating in the computation. The different curves correspond to varying the size of packet transferred between actors, small blocks producing the slow curves on top. These small packet sizes lead to a large overhead due to latency as is demonstrated in the high value of the horizontal asymptote for those curves. Similar timing measurement will be done with the maximum likelihood fits, pushing the tests to much higher number of data points and actors. Each maximum likelihood evaluation requires more computation than the corresponding χ^2 evaluation, and will probably benefit even more from our SALSA based evaluation model.

6 Conclusion

Our research addresses the following major questions: (i) how to provide a programming environment that allows computing non-specialists to develop new parallel and distributed applications using their programming skills? (ii) how to provide high availability, i.e., adding, removing and reorganization of computers without directly involving application programmers? (iii) how to support parallelism management, i.e., the dynamic management of parallel processes and cluster resources? (iv) how to provide transparency to all programmers of a cluster or network of computing resources and relieve them of fine tuning their applications, which is very time consuming, error prone and largely irrelevant? (v) how to automatically and transparently provide fault tolerance to parallel applications?

The developed software infrastructure[1] provides a modular, extensible, scalable infrastructure to prototype research ideas and new algorithms and methodologies for load balancing, fault-tolerance and coordination in highly reconfigurable distributed systems. Ultimately, we want to provide an efficient middleware infrastructure that enables scientists to develop large-scale high-performance computing applications executing over dynamic grids in a way as transparent as possible to system developers and users.

Acknowledgments. Many ideas presented here are the result of countless discussions in the IO research and development group at RPI. In particular, we would like to express our gratitude to Travis Desell and Kaoutar El Mahgraoui for implementing the ARS and NRS strategies, and to Chris Kona, Heschi Kreinick, and Zack Goldstein for implementing the SALSA fitting programs. The work described here has been supported in part by a Rensselaer Polytechnic Institute Seed Funding Grant.

[1] Source code and documentation available for downloading at http://www.cs.rpi.edu/wwc/io/

References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.
2. G. Agha, N. Jamali, and C. Varela. Agent naming and coordination: Actor based models and infrastructures. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications,* chapter 9, pages 225–246. Springer-Verlag, Mar. 2001.
3. R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94),* pages 356–368, Santa Fe, New Mexico, November 1994.
4. J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman. Benchmarking java against c and fortran for scientific applications. In *Proceedings of ACM Java Grande/ISCOPE Conference,* pages 97–105, 2001.
5. T. Desell, K. E. Maghraoui, and C. Varela. Load balancing of autonomous actors over dynamic networks. In *Proceedings of the Adaptive and Evolvable Software Systems: Techniques, Tools, and Applications Minitrack of the Software Technology Track of the Hawaii International Conference on System Sciences (HICSS'37),* January 2004.
6. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing,* 11(2):115–128, Summer 1997.
7. P. Fry, J. Nesheiwat, and B. Szymanski. Computing Twin Primes and Brun's Constant: A Distributed Approach. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing,* pages 42–49. IEEE Computer Society, July 1998.
8. J. Gosling, B. Joy, and G. Steele. *The Java Language Specification.* Addison Wesley, 1996.
9. A. S. Grimshaw and W. A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM,* 40(1):39–45, Jan. 1997.
10. T. T. Kwan and D. A. Reed. Performance of an infrastructure for worldwide parallel computing. In *13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing,* San Juan, Puerto Rico, 1999.
11. T. Lindholm and F. Yellin. *The Java Virtual Machine Specification.* Addison Wesley, 1997.
12. M. Nibhanapudi and B. K. Szymanski. *High Performance Cluster Computing,* volume I of *Architectures and Systems,* chapter BSP-based Adaptive Parallel Processing, pages 702–721. Prentice Hall, New York, 1999.
13. W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on project serendip data and 100,000 personal computers. In *Proceedings of the Fifth International Conference on Bioastronomy,* 1997. Available at http://setiathome.ssl.berkeley.edu/learnmore.html.
14. R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal. Efficient load balancing for wide-area divide-and-conquer applications. *ACM,* 36:34–43, 2001.
15. C. Varela and G. Agha. Programming dynamically reconfigurable open systems with SALSA. *ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings,* 36(12):20–34, Dec. 2001. http://www.cs.rpi.edu/~cvarela/oopsla2001.pdf.

Parallelization of Large Scale Adaptive Finite Element Computations

Krzysztof Banaś

Section of Applied Mathematics, Institute of Computer Modeling,
Cracow University of Technology, Warszawska 24, 31-155 Kraków, Poland
Krzysztof.Banas@pk.edu.pl

Abstract. The paper discusses general principles for parallelization of finite element codes and presents a model architecture for computational kernels of parallel adaptive programs. The model is based on message passing and domain decomposition. Parallel adaptivity, load balancing and multi-level (multigrid) solvers for systems of linear equations are included into the model. Numerical examples, computed using a prototype implementation of the proposed architecture, illustrate capabilities of the approach.

1 Introduction

The complex character of parallel adaptive finite element computations poses serious problems for the construction of efficient and maintainable finite element software. There have been proposed several concepts of object oriented design, most of them for special problems (see articles in [1,2]). On the other hand there is a growing experience with using complex, parallel codes for complex applications [3,4]. These codes are built based on some principles of object orientedness, but with the primary goal of comprehensiveness and efficiency.

The present paper focuses on the modularization of finite element codes, considered to be the most important ingredient of the design of maintainable, extensible and efficient, parallel (as well as sequential), adaptive (as well as classical) finite element systems. The aim of the research is to develop an architecture for parallel adaptive codes, together with techniques for their implementation.

The paper is organized as follows. The next section presents the adopted model of programming and the target hardware environment for execution. Section 3 describes requirements posed to finite element codes subject to parallelization. Section 4 presents briefly the general strategy for parallelization. Parallelization of a solver for linear equations is shown in Section 5. The next section presents parallelization of two important components of computational kernels of finite element programs - parts related to finite element meshes and to approximation fields. A new architecture is introduced with an important new module for parallel execution, a domain decomposition manager. Interfaces with message passing libraries are briefly considered in Section 7. Numerical examples illustrating the use of a prototype code built based on presented principles is contained in the last section.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 431–438, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Target Hardware Environment and Programming Model

Message passing is used as the programming model for parallelization of finite element codes considered in the paper. Related to the choice of the programming model is the granularity of parallelism in the code. It is assumed that grains should be large, meaning that the size of tasks solved by individual processors should not decrease below certain limits. The size of the problem is defined as the number of degrees of freedom in the approximate finite element solution. As a consequence, the performance of codes is better expressed using the scaled speed up, with fixed size per processor.

In the design stages it has been assumed that the number of unknowns (degrees of freedom) assigned to a single processor should be in the range of thousands, up to tens of thousands. These numbers are effectively dealt with by modern processors equipped with local memories. Below these numbers the use of parallel hardware is questionable.

The assumption in creating the presented model for parallel execution was that the code should scale into the range of tens and hundreds of processors. This means that the envisaged size of problems is in the range from hundreds of thousands to tens of millions of unknowns.

3 Assumptions on Finite Element Codes Subject to Parallelization

The parallelization process described in the paper concern only tightly coupled components belonging to the computational kernel of finite element codes. The routines in the kernel are responsible for creation of the system of linear equations, by means of numerical integration, and the solution of the system. The practice of parallel finite element calculations indicates that it is the algorithm for the solution of systems of linear equations that dictates the requirements for the implementation. Depending on the method employed, the finite element mesh is divided into subdomains, overlapping or non-overlapping, with the aim of making as much data as possible available locally for computations. The parallelization process described in the paper concerns the situation when overlapping domain decomposition, with an arbitrary overlap, is used for solving linear equations. Apart from parallelization of the solver, such situation seems to create the most general requirements for handling meshes and approximation fields in parallel. Handling of non-overlapping subdomains should be possible to obtain by simplifying the more general case, i.e. neglecting the overlap.

4 General Strategy for Parallelization

The whole strategy for parallelization is based on the assumption that the sequential part of the code is modified as little as possible. The computational

domain is split into subdomains and subdomains are assigned to particular processors. Processors executing sequential parts of the code operate on local data. The approach adopted for parallelization requires that the algorithms employed in the parallel program are obtained by only slight modifications of sequential algorithms. Thus parallel algorithms should also operate mostly on local data and have some small number of global steps. Next sections describe sequential algorithms and their modifications for parallel execution, starting from the solver of linear equations, the most important ingredient from the performance point of view.

5 Parallel Linear Equations Solver

The requirements of using as much as possible of sequential procedures are well realized by overlapping domain decomposition methods. In fact the solution of the global system is a suitable combination of solutions obtained locally by different processors operating on their local data. The combination of local solutions involves the exchange of data between subdomains, which is a procedure relatively easy to parallelize. Depending on the storage scheme for the system matrix, the suitable parts of distributed vector of unknowns are indicated and standard message passing procedures used for sending and receiving data. Many options exist for organizing such an exchange, one of them ([5]) is used in the example implementation.

6 Finite Element Core

The novelty of the approach presented in the current paper consist on creating a separate module, domain decomposition manager, designed for handling all parallel tasks. In such a way sequential, mesh manipulation and approximation, modules can remain almost unchanged from their original versions.

6.1 Finite Element Mesh and Mesh Services

It is assumed that there may exist several meshes handled by the code, each mesh having its own unique identifier (ID) and composed of elements of arbitrary type. The data structure related to meshes is not accessed by procedures that manages parallel execution. They interact with interface procedures that provide them with data on meshes and their components. All these procedures operate on local data and identifies mesh components using local (for a given processor or subdomain) identifiers (LIDs). Depending on the representation of mesh in the data structure there may exist objects corresponding to different mesh entities: elements, faces, edges, vertices [6]. The parallelization described below may be applied, with small modifications, to different mesh representations.

The set of procedures forming a mesh manipulation module includes routines for specifying adjacency and “family” information for mesh entities as well as procedures for breaking and clustering elements and other mesh entities. These procedures are used in the parallel code with no modifications.

6.2 Approximation Fields

The data related to approximation fields is considered to be accessible separately from data on finite element meshes, although discrete field values (degrees of freedom) are associated exclusively with mesh entities. Each approximation field is assigned to a particular mesh, however, a single mesh may have several associated fields. Degrees of freedom (DOFs) are equipped with their local IDs and may be associated with any kind of mesh entities (elements, faces, etc.). The only requirement for handling approximation fields in parallel is to be able to read from data structure and write back degrees of freedom indicated by a specific local ID.

6.3 A Model for Parallel Implementation

The basis for parallel implementation of the mesh and approximation related parts of the code is formed by the assumption that every mesh entity and DOF data structure is equipped with a global (inter-processor) identifier (IPID). This identifier can be understood as a substitute for a global address space used in sequential codes. The IPID is composed of a processor (subdomain) number and a local (to a given processor) identifier. IPIDs are not known to sequential modules of the code. There is a special module, called domain decomposition manager, that handles all situations where the access to non-local data is necessary. The domain decomposition manager module perform multiple tasks and interacts with all other kernel modules. It is convenient to describe its operations for subsequent stages of computations.

6.4 Domain Decomposition and Distribution of Data

It is assumed that the initial domain decomposition (mesh partition) is performed by an external module or a separate program - a mesh partitioner. The mesh is decomposed into non-overlapping submeshes. There has to be a mechanism for ensuring the coherency of description of subsequent submeshes, such that they form a single mesh. This task is left for an interface between the mesh partitioner and the domain decomposition manager.

Using the data obtained from a mesh partitioner, domain decomposition manager has to create an overlap for each subdomain. Mesh entities initially assigned to a given subdomain (internal mesh entities) and overlap mesh entities are stored locally. The access to these entities is the same as in the sequential code. Whenever data on external (non-local) entities are necessary for computations, domain decomposition manager can find the processor on which data is stored and request the data using suitable calls.

6.5 Adaptivity

Parallel adaptivity consist of two separate processes: mesh modifications and mesh transfer, both requiring different communication mechanisms.

Mesh modifications. In order to decrease the communication between processors, all data concerning local and overlap mesh entities and DOF structures are stored locally. Hence, breaking and clustering of mesh entities can be performed locally using sequential procedures. The only modifications to sequential procedures concern handling of IPIDs.

Load balancing and mesh transfer. In the parallelization model it is assumed that there is a separate submodule that computes a new mesh partition, whenever load imbalance is encountered. It is furthermore assumed that, for each subdomain, the submodule indicates which elements (and in consequence faces, edges and vertices) should be transferred to which subdomains. Given this map of transfers, the domain decomposition manager starts exchange of mesh entities between subdomains.

When sending data related to an entity the conversion of local IDs has to be performed. This conversion concerns all IDs stored in the data structure, including, whenever applicable, family data (father and sons IDs), adjacency data (neighbors), as well as inclusion data (entities included in entities of higher dimension and, in reverse, entities including entities of lower dimension).

7 Interfaces with Communication Libraries

This issue is relatively simple and consist in selecting a set of generic send/receive and group operations. Additionally initialization and finalization procedures are specified. All these has to be implemented for various communication libraries.

In the example implementation a model of buffered send/receive operations is employed. The data to be send are first packed into the buffer and then the whole buffer is send. Procedures in that model can easily be implemented for the MPI standard as well as packages like PVM.

8 Numerical Examples

Two numerical examples showing capabilities of the described approach are presented in this section. The computational environment consist of a set of Linux workstations connected using a standard 100 Mbit Ethernet network.

8.1 Simulating Diffusion

The first example is Laplace's equation, $\Delta u = \Delta u_{ex}$, where u_{ex} is the known exact solution, $u_{ex} = \exp(-x^2 - y^2 - z^2)$ The computational domain consist of the box $[0, 0.1] \times [0, 1] \times [0, 10]$ and boundary conditions are chosen to match the exact solution. Discontinuous Galerkin approximation [7] and the preconditioned GMRES method are used for solving the problem.

Table 1. Results for 10 iterations of the preconditioned GMRES method and discontinuous Galerkin approximation used for solving Laplace's equation in a box domain (description in the text).

Single level preconditioner						
N_{DOF}	N_{proc}	Error*10^9	Rate	Time	Speed up	Efficiency
391 168	1	9.313	0.626	17.85	1.00	100%
	2	10.173	0.632	8.93	1.99	100%
	4	10.252	0.633	4.53	3.94	98%
	8	11.183	0.638	2.34	7.63	95%
3 129 344	2	48.041	0.738	70.76	1.00	100%
	4	47.950	0.738	35.63	1.98	99%
	8	48.748	0.739	17.71	3.99	100%
Three level preconditioner						
N_{DOF}	N_{proc}	Error*10^9	Rate	Time	Speed up	Efficiency
391 168	1	0.018	0.335	26.18	1.00	100%
	2	0.017	0.334	14.18	1.85	92%
	4	0.018	0.335	9.08	2.88	72%
	8	0.024	0.346	7.60	3.44	43%
3 129 344	2	0.027	0.350	111.16	1.00	100%
	4	0.027	0.350	57.76	1.92	96%
	8	0.027	0.348	33.15	3.35	84%

Table 1 presents results for a series of computations corresponding to the described problem. Two preconditioners are employed, both use the combination of additive Schwarz preconditioning for the whole problem and multiplicative Schwarz within subdomains [8,9]. The first is single level preconditioner and the second use three consecutive mesh levels to achieve multigrid preconditioning. For each preconditioner problems of different sizes, corresponding to subsequently uniformly refined meshes, are considered. For each combination preconditioner/problem size results of computations using 1, 2, 4 and 8 workstations are shown. For the largest problem the reference number of processors to compute speed up and efficiency is two, since the problem did not fit into a memory of a single computer.

Results are reported for 10 iterations of the preconditioned GMRES method to focus on the efficiency of parallel implementation, not considering the influence of parallelization on the convergence of GMRES (nevertheless the latter is reported for completeness). Subsequent meshes are obtained by uniform refinements and for each mesh N_{DOF} is the number of degrees of freedom. N_{proc} is the number of workstations solving the problem. *Error* is the norm of residual after 10 GMRES iterations (within a single restart) and *Rate* is the total GMRES convergence rate during solution. Execution time *Time* is a wall clock time. Speed-up and efficiency are computed in the standard way.

For the same problem a large scale simulation has been performed to show scalability of the code. A mesh with more than $6 \cdot 10^6$ elements and $25 \cdot 10^6$ DOFs has been obtained by uniform parallel refinements. The problem was solved using

a single level Gauss-Seidel preconditioning in approx. 200 iterations. The run on a cluster of 32 Pentium II processors required 16 GBytes of memory and 20 minutes of time.

Table 2. Characteristics of mesh transfers during parallel simulation for the convection problem.

	Time step number				
	100	101	102	103	104
Average number of DOFs	5086	5086	5086	5086	5086
Maximal number of DOFs	5636	5120	5372	5596	5120
Minimal number of DOFs	4468	5012	4732	4508	4996
Number of transfered vertices	300	0	0	390	0
Number of transfered edges	1212	0	0	1671	0
Number of transfered faces	1284	0	0	1863	0
Number of transfered elements	438	0	0	657	0

8.2 Simulating Convection

The second example is a simple convection problem in the box $[0, 38] \times [0..1000] \times [0..18]$. A rectangular pattern is traveling from left to right (along the y-axis). GMRES with single level Schwarz preconditioning is used, once again with discontinuous Galerkin approximation. The only interesting process for this example, that will be described in more detail, are the subsequent parallel mesh adaptations and load balancing achieved through transfer of mesh entities. There are four workstations used for simulation and the computational domain is divided into four subdomains. Subdomains have two element overlap to enable mesh adaptations and overlapping Schwarz preconditioning. After each time step (in the example run there were 120 time steps) the mesh is adapted in parallel.

After each mesh adaptation, the number of degrees of freedom in each subdomain is checked against the average number of DOFs (it is assumed that processors are of the same speed). If imbalance larger than 10% is encountered, mesh repartitioner is called, to provide new domain decomposition. According to the new assignment of elements to processors and two element overlap requirements, mesh entities are marked respectively, and the transfer between subdomains takes place. To enable clustering, mesh transfers consider always whole element families - initial elements that are marked for a transfer and all their antecedents.

Table 2 presents characteristics of mesh transfers for five subsequent time steps, from 100 to 104. The average number of DOFs in a subdomain remains constant since the same number of elements appears due to refinements and disappears due to derefinements. Since refinements and derefinements takes place in different regions the difference between the subdomain with the greatest number of DOFs and the subdomain with the smallest number of DOFs grows after each time step.

For the whole simulation, the speed up obtained using 4 processors was equal to 2.67, giving the efficiency of 67%. For the overhead that includes mesh repartitioning, mesh transfers and the fact that, according to the overall strategy, the load for processors is not perfectly balanced, the results appear to be reasonable.

8.3 Conclusions

The presented examples show feasibility of the proposed modular architecture for parallel adaptive codes. The prototype implementation fulfills design requirements and offers flexibility and maintainability of the code. Despite the fact that the implementation is not yet fully optimized, the obtained efficiencies of parallel execution, as well as the ability to solve large scale problems, are satisfactory. There are several deficiencies, that will be improved in the fully developed code.

Acknowledgment. The support of this work by the Polish State Committee for Scientific Research under grant 7 T11F 014 20 is gratefully acknowledged.

References

1. *Engineering with Computers,* **15,** (1999).
2. H.P. Langtangen, A.M. Bruaset, and eds. E. Quak, *Advances in software tools for scientific computing,* Springer, Berlin/Heidelberg, 2000.
3. P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. Rentz-Reichert, and C. Wieners, 'Ug - a flexible software toolbox for solving partial differential equations', *Computing and Visualization in Science,* **1**(1), 27–40, (1997).
4. M.W. Beall and M.S. Shephard, 'An object-oriented framework for reliable numerical simulations', *Engineering with Computers,* **15,** 61–72, (1999).
5. J. Płażek, K. Banaś, and J. Kitowski, 'Comparison of message passing and shared memory implementations of the GMRES method on MIMD computers', *Scientific Programming,* **9**, 195–209, (2001).
6. J.-F. Remacle, B.K. Karamete, and M.S. Shephard, 'Algorithm Oriented Mesh Database', Report 5, SCOREC, (2000).
7. J.T. Oden, I. Babuska, and C.E. Baumann, 'A discontinous hp finite element method for diffusion problems', *Journal of Computational Physics,* **146,** 491–519, (1998).
8. K. Banaś and M.F. Wheeler, 'Preconditioning GMRES for discontinuous Galerkin approximations', *to appear in Computer Assisted Mechanics and Engineering Science.*
9. C. Lasser and A. Toselli, 'Overlapping preconditioners for discontinuous Galerkin approximations of second order problems', in *Proceedings of the XIIIth International Conference on Domain Decomposition Methods, Lyon, October 2000,* eds., N. Debit et al. DDM.org, (2001).

A Multi-agent System Based on the Information Metabolism Theory

Andrzej Bielecki and Dominika Nowak

Institute of Computer Science, Jagiellonian University,
ul. Nawojki 11, 30-072 Kraków, Poland
uibielec@cyf-kr.edu.pl

Abstract. The information metabolism approach to multi-agent systems is discussed. The organization and design methodology of such systems is studied.

1 Introduction

Development in the area of synthetic autonomous agents has promoted the study of a cognitive basis of behavior (see [6], [13] and references given there). Both the computational models and implementations of emotional agents are described in literature ([13]). Such approach to modelling an agent or a community of agents is based on models of animal, human and social cognitive abilities - compare [9] and [14]. There are attempts to create mathematical models of some aspects of human consciousness considering also consciousness dynamics ([6], [9]) and implementing them using artificial neural networks ([4]), which makes it possibile to use neural networks to create agents having some cognitive properties ([5], [14]).

In this paper a model of MAS which is based on information metabolism is introduced. This theory was founded at the beginning of seventies by a Polish psychiatrist Antoni Kępiński ([10]) and used by him to describe the structure and dynamics of human psyche both in a normal state and in pathology. Applications to psychiatry and psychology have been being systematically developed by Kokoszka (see [11] and references given there). The significance of Kępiński's concepts goes far beyond psychology and psychiatry. The model can be applied to a very wide class of cybernetic objects - self controlling open systems.

Kępiński was aware of the correspondence of his concepts to physics, while writing about negentropy and to cybernetics, using in his theory the concept of a feedback in information processes and analysing mental life in the context of self-control systems.

Another direction of studies is connected with creating cybernetic and mathematical descriptions of the information metabolism theory in order to apply the idea to computer systems ([3], [4], [9]). This paper is a continuation of this approach and, as it was mentioned above, is an attempt to apply the Kępiński's concepts to MASes design.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 439–446, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Information Metabolism Approach to Multi-agent Systems

As the information metabolism theory was treated very utilitarialy by its founder, it is not described in any article nor monography but it is spread in a few works. Furthermore, nobody has tried to describe metabolism of information as a complete theory. This is the main problem in studies concerning Kępiński's concepts. Let us specify the key points of the theory. According to Kępiński, the metabolism of information has its own boundaries, a control center, a system of information distribution and processing and a source of energy. Functional structures of information metabolism are shaped early in life maintain order in space and time and the layers of systems of value. The amount, complexity and plasticity of functional structures increases along with the development. The hierarchy of values governs the mechanism which select and filter the information reaching any particular decision making level. The process on each level proceed simultaneously and are conjugate to each other.

In this paper the aspects which seemed to be useful in MASes are specified explicitely and possibilities of application are discussed. The considerations are led paragraph by paragraph and each paragraph consists of two subpoints - the first one concerns metabolism of information whereas in the second one applications to MASes are discussed.

1. **Openess**
   a) Living creatures are open systems. This means that exchange of energy and information with the environment is necessary for their existence. The energy is used to maintain the information metabolism. Energetic metabolism dominates in primitive creatures like bacteria whereas information metabolism dominates in such creatures as mammals.
   b) It is one of the basic assumption that in a MAS an agent exchanges information with its environment. Modelling such information flow is an interesting problem. The dynamical systems theory has already been used for modelling interactions between an agent and its environment ([1]) and it seems to be a convenient tool for the description of this process. Furthermore, the theoretical frame of information metabolism based on dynamical systems and including energetic aspects has been created as well - [3], [9].
2. **Hierarchism**
   a) Metabolism of information has a few branches: biological, emotional and a logical one. Each of them is multileveled and continuous in time. They run simultaneously and are conjugate to each other. The lower levels are subconscious, whereas the upper ones are conscious. Information metabolism has its own center, system of information processing and distribution and boundaries separating it from the environment. Internal structural constructions are responsible mainly for reception, selection and transmission of information.

b) Multi-agent systems are organized in a hierarchical way. They have at least two levels of organization: a single agent level and a social one. Though the hierarchical organization of MASes is often taken into consideration in theoretical studies, the theory of information metabolism creates possibility to work out the problem in details. Thus, we can specify the following levels of information metabolism. The signal level and the level of single commands are most basic ones. The level of simple, non-structured algorithms consisting of only a few commands can be specified as a succeeding one. The levels of single structured algorithms and algorithms acting simultaneously and communicating with each other are the two most complex ones. The simplest levels are responsible for performing simple, fast actions. In real time systems they play a key role if the system is in danger. The third level performs tasks which are automatic from the MAS's point of view. The algorithm levels allow to achieve the goal which has been specified as a MAS's one. The specified levels, especially the complex ones, can have their own inner structures. The information processing on each level runs simultanously, communicating with the neighbouring levels.

As it was mentioned, there are subconscious and conscious levels of information metabolism. Subconscious ones can be implemented as neural networks because they model functional input-output dependencies without casual context - generally it is not known why neuronal weights reach their actual values after learning process. The conscious levels can be modelled by such structures as rule sets or semantic nets.

3. **Energetic Aspect**

a) Information metabolism needs energy centers which supply the energy necessary for preservation of metabolism i.e. proper stimuli reception, selection and integration as well as decision making. Energy is produced in energy centers and then it is transported into each branch of information metabolism. The structure of energy transport affects information metabolism.

b) Energy flows could be modelled in such a way that an agent would get virtual *energetic units* as payment for performing a task efficiently. These units would be necessary for an agent to live, i.e. an agent's amount of units would decrease systematically in consequence of agent existence and activity. These units could also be used to hire other agents for performing subtasks or to reproduce and create a new agent. The energy distribution to the branches of information metabolism and its influence on metabolism should also be taken into account - it is considered in the dynamical model - see [3].

4. **Evolution of Cognitive Structures**

a) The information metabolism theory underlines that the main role in psychical processes is played by some interacting structures evolving in time. The amount, complexity and plasticity of internal structures increase along with the development of an individual. There are structures,

called elimination centers, responsible for removing useless and unimportant information.

b) At the highest level of information metabolism functional structures are given as sets of rules with the probability distribution. At the lower levels they could be implemented as neural networks (see the point concerning hierarchism). During the shaping process the connections between structures would be created or destroyed, probability distributions would be changed, new neural networks and rules would be created or removed, neural weights would be changed. The way in which it would be done is connected, among others, with bahnung phenomenon described in the next point.

5. **Bahnung**

a) Bahnung is a phenomenon connected with facilitation of impulse transmission along nervous pathways. At the higher levels this means that for the once chosen decision the probability to be chosen in future in similar circumstances increases.

b) If a given structure has been chosen, the distribution of probability is changed. A structure which is used frequently is automatizied - it is moved to a lower level. This means that at a lower level there is created a structure performing the same tasks and the old one is removed from the higher level.

3 Application of Information Metabolism Theory in MAS

As the term 'Computer Aided Design' shows, computer systems in general still remain an aid for human designers, whose main ability is creativeness, unachieved by any computer so far. However, in many specific tasks the specialised computer systems can work unattended basing on initial information or/and an occasional feedback from the designer, giving satisfying results. Extensive research in the field of CAD is carried out, including application of genetic algorithms ([2]), expert systems ([7]), graph grammars ([8]) etc. Using multi-agent expert systems in design is a relatively new approach. In this paper an example of a model of such system, basing on the information metabolism theory, is introduced.

3.1 General Assumptions

The described system perfoms three main tasks:

1. Designs furniture sets;
2. Designs arrangements of furniture in the room;
3. Aids production of the furniture.

It is planned as a multi-user distributed commercial system, realizing designs of non-standard furniture sets for individual customers on demand. Basing on

information supplied by the customer, the system creates adequate designs, possibly passing the order to manufacturing, if needed. Input information includes description of the required furniture set, the room where it should be placed, acceptable materials (wood, metal, glass, laminated panels) and the time limits. Ready designs are evaluated, and the system receives feedback about the price.

3.2 Structure of the System

There are a few kinds of agents active in the system, acting simultaneously and performing different tasks. A separate entity is the database, collecting information about created designs and arrangements.

The designing agents: Each agent in this group designs one piece of the furniture and then decides about its position. Agents are specialized in designing specific types of furniture, having adequate knowledge in their disposal. The 'facts' in knowledge bases of these agents are represented by a set of three-dimensional primitives (basic solids), a set of predefined atomic furniture elements (such as a door knob), and graphic representations of a piece furniture or its elements (e.g. drawers). An example of such knowledge base is shown in [12]. The space in the room, which is to be furnished, can be considered as the agents' pool of resources. The base of rules for each agent in this group divides into four different sets of rules:

1. Graph grammar rules for designing more complex pieces of furniture with variable number of elements (terminal symbols for these grammars are placed in base of facts). A simple graph grammar is shown [12].
2. Technical rules concerning resizing and reconstructing the furniture. Should also include information about possibility of element scaling;
3. Interpretation rules, describing relations between physical elements - necessary to build a construction from a graph representation. This information concerns mainly graph's edges.
4. Rules relating to the arrangement of furniture in the room. The user should be able to override these rules, if he needs to, or fix positions for some pieces of furniture.

The individual agent's activity manifests itself in four main levels, having reflection in Kępiński's hierarchical levels of informational metabolism:

1. On the first level the agent looks for the desired piece of furniture in its knowledge base. This could be sufficient in case of a very simple piece of furniture, the one which was designed earlier, or when the user cares just for its arrangement - the primitive solids can be used instead complex furniture designs in this case;
2. If the required piece of furniture exists in the knowledge base as a structure, but in the inappropriate size, the goal of the agent is to resize it. This happens on the second level, and might require reconstruction of the structure, according to the technical rules. If it turns out that the piece of furniture

cannot be resized (without loss of stability, for instance), the third level should be activated.

3. On the third level the agent makes use of graph grammar defined in its base of rules for a particular piece of furniture. The possible scheme of action is to evaluate all possible words generated by the grammar, checking if they meet the demands (and cutting the branches in the derivation tree which would not lead to satisfying solutions). The agent can stop when first matching word is found, or choose between all possibilities basing on some additional criteria (like symmetry), or eventually let the user choose in an interactive way. Finally, a piece of furniture must be resized, and if the result is satisfying, it might be added to the base of facts.
4. On the fourth level all agents cooperate to create a satisfying arrangement of furniture in a given room. Agents representing dependent pieces of furniture (like a desk and an office chair) should act together in this case. Each agent has a function of space evaluation - some places in the room might be more valuable, depending on the kind of furniture. The arrangement should maximize the sum of such values for all agents. There could be different ways of achieving this goal, from a simple backtracking algorithm to much more complicated rules of negotiations between agents.

Agents Creating Arrangments: This group of agents deals with arranging furniture sets in rooms, adding new information to the base of arrangements. However, these agents do not work for a specific order, and consequently are not bounded by time limits. They might develop alternative arrangements for finished projects, or arrange designed furniture sets in different rooms.

Agents Working With Database: These agents work with the database containing arrangements and designs created so far by the system. Agents should query the database in a way to find the most useful designs for current problem, taking into consideration cases when arrangement may be a part of an existing one or its transformation.

Technical Agents: The ready furniture designs are passed to this group of agents. Their task is to prepare a precise plan of work for a piece of furniture, a plan of optimal material usage (in agreement with storekeeper agent), or even making instructions for mechanical devices (e.g. to cut boards). In case of producing DIY furniture, agents should be able to make assembly instructions for the furniture.

Storekeeper Agent: There is only one agent of this kind in the system. Its task is to trace consumption of material and prefabricated products in the system, alarming about shortages, possibly even preparing orders. Storekeeper agent exchanges information with technical agent, notifying them about available materials and collecting information about predicted usage.

Ready designs of sets of furniture along with the suitable arrangement schemes (information concerning the room) may be collected in the system in an external database. As the system's experience increases, it would be reasonable to search this database for an earlier solution to a problem similar to given. Sepa-

rate group of agents should interact with such database, passing the information to agents designing furniture.

Information flow in the system is shown in Fig. 1.

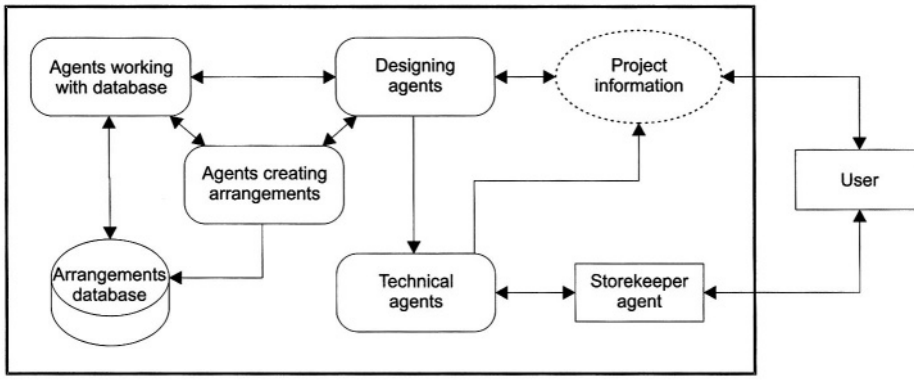


Fig. 1. Information flow between groups of agents and the database.

As it was described, such system has much in common with theory of information metabolism. The activity of agents divides into several hierarchical levels with increasing degree of complication, and exchange of information between individual levels occurs. There is also a common goal for all agents (which is solving the problem of room furnishing), and there are processes in the system, which are connected with achieving the goal only indirectly - that could be compared with subconscious processes in human's mind. Finally, there is a possibility of changing the knowledge base of each agent through the structure evolution resulting from the system's experience.

Energetic aspect of information metabolism could be simulated in the system by assigning virtual money to the agents. Ready designs and arrangements are evaluated outside the system, and information concerning project's commercial value is fed back. Then all design agents which worked for the project might receive payment, adequate to their contribution. Money should be spent by agents on 'costs of living' (which would be maintenance of the database) or hiring another agent to share their tasks.

4 Concluding Remarks

As it was mentioned in the introduction section, new theoretical frames for MASes have been being looked for. It seemed that Kępiński's ideas are promising ones in this context. In the paper a theoretical frame based on the metabolism information theory is sketched. It should be stressed that the presented study, according to its volume limitations, is far from exhausting. Not only at least a few other points can be added to the second section but also deeper analysis of

applications can be led in every specified point. However, only an effective implementation of a MAS based on information metabolism theory can constitute the ultimate test of usefulness.

References

1. Beer R.D., *A Dynamical Systems Perspective on Agent-Environment Interaction,* Artificial Intelligence, vol.72, 1995, 173-215.
2. Bentley, P. J & Wakefield, J. P., *The Table: An Illustration of Evolutionary Design using Genetic Algorithms* . Proc. Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications, IEE Conference Publication No. 414, 12-14 Sept. 1995.
3. Bielecki A., *A Mathematical Model of Kępiński's Information Metabolism Theory,* Proceedings of the Sixth National Conference " Application of Mathematics in Biology and Medicine", Zawoja, Poland, 2000, 9-13.
4. Bielecki A., *A Neuronal System for Simulation of Consciousness Dynamics Based on Information Metabolism Theory,* Proceedings of the Seventh National Conference "Application of Mathematics in Biology and Medicine", Zawoja, Poland, 2001, 15-19.
5. Bielecki A., *Agent's Adaptivity - Some Aspects of Theoretical Foundations of a Neural Agents Training Process,* Lecture Notes in Artificial Intelligence, vol.2296, 2002, 47-53.
6. Canamero D., *Modeling Motivations and Emotions as a Basis for Intelligent Behavior,* Proceedings of the First International Conference on Autonomus Agents, Marina del Rey, California USA, 1997, 148-155.
7. Gero, J. S., *Design Prototypes: A Knowledge Representation Schema for Design.* AI Magazine, Special Issue on Design, (Eds) J. S. Gero & M. L. Maher, AAAI, Vol. 11, No. 4, Winter 1990, 27-36.
8. Hliniak G., Strug B., *Graph grammars and evolutionary methods in graphic design* , Machine GRAPHICS & VISION, 9,1/2 (2000), 5-13.
9. Kokoszka A., Bielecki A., Holas P., *Mental Organization According to Metabolism of Information and its Mathematical Description,* International Journal of Neuroscience, vol 107, 2001, 173-184.
10. Kępiński A., *Melancholia (Melancholy),* Wydawnictwo Literackie, Kraków, 2001 (first edition: PZWL, Warszawa, 1972) (in Polish).
11. Kokoszka A., *Information Metabolism as a Model of Human Experience,* International Journal of Neuroscience, vol.97, 1999, 169-178.
12. Nowak D., *Zastosowanie wieloagentowego systemu ekspertowego w projektowaniu (An application of a multi-agent expert system in design),* Inżynieria Wiedzy i Systemy Ekspertowe, Z. Bubnicki, A. Grzech (eds.), Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2003, 114-121 (in Polish).
13. Velasquez J.D., Maes P., *Cathexis: A Computational Model of Emotions,* Proceedings of the First International Conference on Autonomus Agents, Marina del Rey, California USA, 1997, 518-519.
14. Zrehen S., Gaussier P., *A Neural Architecture for Motivated Navigation Behavior in an Animat,* Proceedings of the First International Conference on Autonomus Agents, Marina del Rey, California USA, 1997, 478-479.

Application of Cellular Automata for Cryptography

Pascal Bouvry[1], Franciszek Seredyński[2,3], and Albert Y. Zomaya[4]

[1] Faculty of Sciences, Technology and Communication
Luxembourg University
6, rue Coudenhove Kalergi, L-1359 Luxembourg-Kirchberg, Luxembourg
pascal.bouvry@ist.lu http://www.ist.lu/users/pascal.bouvry
[2] Polish-Japanese Institute of Information Technologies
Koszykowa 86, 02-008 Warsaw, Poland
[3] Institute of Computer Science of Polish Academy of Sciences
Ordona 21, 01-237 Warsaw, Poland
sered@ipipan.waw.pl http://www.ipipan.waw.pl/~sered
[4] School of Information Technologies, University of Sydney
Sydney, NSW 2006 Australia
zomaya@it.usyd.edu.au http://www.cs.usyd.edu.au/~zomaya

Abstract. New results concerning application of cellular automata (CAs) to secret key cryptography is described in this paper. One dimensional nonuniform CAs are considered for generating pseudo-random number sequences (PNSs) used in a secret key cryptographic system. The quality of PNSs highly depends on the set of applied CA rules. The search of rules relies on an evolutionary technique called cellular programming. Different rule sizes are considered. As the result of collective behavior of discovered set of CA rules very high quality PNSs are generated. Indeed the quality of PNSs outperforms the quality of known one dimensional CA-based PNS generators used for secret key cryptography. The extended set of CA rules proposed in this article makes the cryptography system much more resistant on attacks.

1 Introduction

Today there is no need for describing the increasing needs in terms of security. The emergence of ad-hoc and ubiquitous networking requires new generations of lightweight security solutions. Cryptography techniques are essential component of any secure communication. Two main cryptography systems are used today: secret and public-key systems. An extensive overview of currently known or emerging cryptography techniques used in both type of systems can be found in [13]. One of such promising cryptography techniques consists of applying cellular automata (CAs). Let's also highlight the massively parallel characteristic of CA-based solutions and their efficiency.

CAs were proposed for public-key cryptosystems by Guan [1] and Kari [5]. In such systems two keys are required: one key for encryption and the other for

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 447–454, 2004.
© Springer-Verlag Berlin Heidelberg 2004

decryption, and one of them is held in private, the other rendered public. The main concern of this paper are however cryptosystems with a secret key. In such systems the encryption key and the decryption key are the same. The encryption process is based on generation of pseudorandom bit sequences, and CAs can be effectively used for this purpose. CAs for systems with a secrete key were first studied by Wolfram [17], and later by Habutsu *et al.* [3], Nandi *et al.* [10] and Gutowitz [2]. Recently they were a subject of study by Tomassini & Perrenoud [15], and Tomassini & Sipper [16], who considered one and two dimensional (2D) CAs for encryption scheme. This paper is an extension of these recent studies and concerns of application of one dimensional (1D) CAs for the secret key cryptography.

The paper is organized as follows. The next section presents the idea of an encryption process based on Vernam cipher that is used for CA-based secret key cryptosystem. Section 3 outlines the main concepts of CAs, overviews current state of applications of CAs in secret key cryptography and states the problem considered in the paper. Section 4 outlines evolutionary technique called cellular programming and section 5 shows how this technique is used to discover new CA rules suitable for the encryption process. Section 6 contains the analysis of results and the last section concludes the paper.

2 Vernam Cipher and Secret Key Cryptography

Let P be a plain-text message consisting of m bits $p_1p_2...p_m$, and $k_1k_2...k_m$ be a bit stream of a key k. Let c_i be the $i-th$ bit of a cipher-text obtained by applying *XOR* (exclusive-or) enciphering operation:

$$c_i = p_i \; XOR \; k_i.$$

The original bit p_i of a message can be recovered by applying the same operation *XOR* on c_i using the same bit stream key k:

$$p_i = c_i \; XOR \; k_i.$$

The enciphering algorithm called Vernam cipher is known to be [8,13] perfectly safe if the key stream is truly unpredictable and used only one time. From practical point of view it means that one must find answers on the following questions: (a) how to provide a pure randomness of a key bit stream and unpredictability of random bits, (b) how to obtain such a key with a length large enough to encrypt practical amounts of data, and (c) how to pass safely the key from the sender to receiver and protect the key.

In this paper we address questions (a) and (b). We will apply CAs to generate high quality pseudorandom number sequences (PNSs) and a safe secret key. CAs have been used successfully to generate PNSs. We will show that by using 1D CAs, the quality of PNSs for secret key cryptography and a safety of the key can be increased.

3 Cellular Automata and Cryptography

One dimensional CA is in the simplest case a collection of two-state elementary automata arranged in a lattice of the length *N,* and locally interacted in a discrete time *t.* For each cell i called a central cell, a neighborhood of a radius r is defined, consisting of $n_i = 2r + 1$ cells, including the cell i. When considering a finite size of CAs a cyclic boundary condition is applied, resulting in a circle grid.

It is assumed that a state q_i^{t+1} of a cell i at the time $t + 1$ depends only on states of its neighborhood at the time t, i.e. $q_i^{t+1} = f(q_i^t, q_{i1}^t, q_{i2}^t, ..., q_{ni}^t)$, and a transition function f, called a *rule,* which defines a rule of updating a cell i. A length L of a rule and a number of neighborhood states for a binary uniform CAs is $L = 2^n$, where $n = n_i$ is a number of cells of a given neighborhood, and a number of such rules can be expressed as 2^L. For CAs with e.g. $r = 2$ the length of a rule is equal to $L = 32$, and a number of such rules is 2^{32} and grows very fast with L. When the same rule is applied to update cells of CAs, such CAs are called uniform CAs, in contrast with nonuniform CAs when different rules are assigned to cells and used to update them.

The first who applied CAs to generate PNSs was S. Wolfram [17]. He used uniform, 1D CAs with $r = 1$, and rule 30. Hortensius *et al.* [4] and Nandi *et al.* [10] used nonuniform CAs with two rules 90 and 150, and it was found that the quality of generated PNSs was better that the quality of the Wolfram system. Recently Tomassini and Perrenoud [15] proposed to use nonuniform, 1D CAs with $r = 1$ and four rules 90, 105, 150 and 165, which provide high quality PNSs and a huge space of possible secret keys which is difficult for cryptanalysis. Instead to design rules for CAs they used evolutionary technique called cellular programming (CP) to search for them.

In this study we continue this line of research. We will use finite, 1D, nonuniform CAs. However, we extend the potential space of rules by considering two sizes of rule neighborhoods, namely neighborhood of radius $r = 1$ and $r = 2$. To discover appropriate rules in this huge space of rules we will use CP.

4 Cellular Programming Environment

4.1 Cellular Programming

CP [9] is an evolutionary computation technique similar to the diffusion model of parallel genetic algorithms and is introduced [14] to discover rules for nonuniform CAs. In contrast with the CP used in [15] the system has the possibility to evaluate nonuniform rules of two types. The system consists of a population of N rules (left) and each rule is assigned to a single cell of CAs (right). After initiating states of each cell, i.e. setting an initial configuration, the CAs start to evolve according to assigned rules during a predefined number of time steps. Each cell produces a stream of bits, creating this way a PNS.

After stopping CAs evolution, all PNSs are evaluated. The entropy E_h is used to evaluate the statistical quality of each PNS. To calculate a value of the entropy each PNS is divided into subsequences of a size h. In all experiments

the value $h = 4$ was used. Let k be the number of values which can take each element of a sequence (in our case of binary values of all elements $k = 2$) and k^h a number of possible states of of each sequence ($k^h = 16$). E_h can be calculated in the following way:

$$E_h = -\sum_{j=1}^{k^h} p_{h_j} log_2\ p_{h_j},$$

where p_{h_j} is a measured probability of occurrence of a sequence h_j in a PNS. The entropy achieves its maximal value $E_h = h$ when the probabilities of the k_h possible sequences of the length h are equal to $1/k^h$. It is worth to mention that the entropy is only one of possible statistical measures of PNSs. It will be used as a fitness function of CP. To decide about final statistical quality of PNSs and suitability of the discovered rules for cryptography purposes some additional tests must be conducted.

A single PNS is produced by a CA cell according to assigned rules and depends on a configuration c_i of states of CAs. To evaluate statistically reliable value of the entropy, CAs run with the same set of rules C times for different configurations c_i, and finally the average value of entropy is calculated and serves as a fitness function of each rule from the population of rules.

After evaluation of a fitness function of all rules of the population genetic operators of selection, crossover and mutation are locally performed on the rules. The evolutionary algorithm stops after some predefined number of generations.

4.2 Genetic Operators

In contrast with standard genetic algorithm population, rules - individuals of CP population occupy specific place in the population and have strictly defined neighborhood. We assume that rules are either of type 1 ($r = 1$, short rules) or of type 2 ($r = 2$, long rules).

Additionally to a neighborhood associated with two types of rules we introduce for rules an evolutionary neighborhood, i.e. the neighborhood of rules which are considered for mating when genetic operators are locally applied to a given rule. The size and pattern of this neighborhood may differ from the neighborhood associated with types of rules.

A sequence of genetic operators performed locally on a given rule depends on values of fitness function of rules from the evolutionary neighborhood of this rule. Genetic operators are applied in the following way:

1. if the $k - th$ rule is the best (the highest value of the fitness function) in its evolutionary neighborhood then the rule survives (selection) and remains unchanged for the next generation; no other genetic operators are performed
2. if in the evolutionary neighborhood of the rule k only one rule exists which is better than considered rule then the rule k is replaced by better rule (selection) only if both rules are of the same type, and next mutation on this rule is performed; the rule remains unchanged if the better rule is of the other type

3. if two rules that are better than the rule k exist in the neighborhood then a crossover on the pair of better rules is performed; a randomly selected child from a pair of children replaces rule k, and additionally the mutation operator is performed
4. if more than two rules better than the rule k exist in the neighborhood then two randomly selected better rules create (crossover) a pair of children; on a randomly selected child a mutation is performed, and the child replaces the rule k.

Two types of rules existing in a CP population can be considered as two species of a coevolutionary algorithm. Therefore to perform a crossover between rules special regulations are required. It is assumed that two parental rules of the same species create a single child rule of the same species, which can replace either the first type of a rule or the second type of the rule. If rules of different types take part in the mating then a species of a child depends on species of a replaced rule, and is the same as a species of a rule to be replaced.

The short rule $P1$ taking part in crossover consists of 8 genes $(n = 0, ..., 7)$ which values correspond to values of transition function defined on 8 neighborhood states {000,001,..., 111} existing for $r = 1$. The long rule $P2$ consists of 32 genes, each corresponding to values of transition function defined on 32 neighborhood states existing for $r = 2$. The long rule is folded because there is a strict relation between a state order number which corresponds to $j - th$ gene of $P1$ and states' order numbers corresponding to genes $2j, 2j+1$ and $2j+16, 2j+17$ of $P2$. These order numbers of states of $P2$ are just an extension of corresponding order number of a gene from $P1$. For example, the gene $n = 7$ of $P1$ corresponds to the neighborhood state {**111**}, and genes 15, 14 and 31, 30 of $P2$ correspond to states respectively {0**1111**, 0**1110**} and {1**1111**, 1**1110**} containing the state of $P1$ (marked in bold).

Last genetic operator is a flip-bit mutation performed with the probability $p_m - 0.001$.

5 Discovery of Rules in 1D, Nonuniform CAs by Using CP

In all conducted experiments a population of CP and the size of nonuniform CAs were equal to 50 and the population was processed during 50 generations. The CAs using an initial random configuration of states and a set of assigned rules evolved during M = 4096 time steps. Running CAs with a given set of rules was repeated for C = 300 initial configurations.

A typical result of a single run of an evolutionary process starting with a random rules assigned to cells of CAs is discovering by CP a small set of good rules which divide the cellular space of CAs into domains - areas where the same rules live together. Evolutionary process is continued on borders of domains where different rules live. This process may result in increasing domains of rules which are only slightly better than neighboring rules, which domains will

decrease and finally disappear. This happens in particular when two neighboring domains are occupied respectively by the same short rules and the same long rules. The search space of short rules $(r = 1)$ is much smaller than the search space of the long rules $(r = 2)$. Therefore better short rules are discovered faster than better long rules, and for this reason long rules are gradually replaced by short rules. To limit this premature convergence of short rules, the short and long rules are initially randomly assigned to cells in the proportion of 1:3 in all subsequent experiments.

To find out what is the influence of a shape of the evolutionary neighborhood on the quality of PNSs generated by CAs, each experiment with a given shape of the neighborhood was repeated 10 times, and the average value of the entropy over each set of experiments was considered. The experiments have shown that while for each shape of the neighborhood very good rules with the entropy equal or close to 3,989 were observed, the average value of the entropy over sets of experiments ranged from 3,946 to 3,956 for neighborhoods 111 and 1__1__1, and from 3.960 to 3.975 for the remaining neighborhoods. For this reason only neighborhoods 11111, 1111111, and 11_1_11 were considered in next experiments.

The purpose of the experiments which followed was to discover an enlarged set of rules (to enlarge the key space of cryptography system) that working collectively would produce very high quality PNSs. It was noticed that in a single run of CP the evolutionary algorithm produces typically a set of four rules with a very high value of the entropy, but the quality of a rule depends on a neighborhood of the rule. As the result of experiments 8 short rules $(r = 1)$ was selected: the rules 30, 90, 105, 150 and 165 discovered previously by [15] and additionally new rules 86, 101 and 153, and also 39 long rules $(r = 2)$ were discovered.

6 Analysis and Comparison of Results

The entropy used as a fitness function for evolution of high quality CA rules is only one of existing statistical tests of PNSs. None of them is enough strong to claim statistical randomness of a PNS in the case of passing a given test. Passing by a PNS of n statistical tests increases certainty about degree of its randomness but there is not any guarantee that the PNS will not fail on the next test. For this reason discovered sets of rules need to be verified by additional number of statistical sets. Even passing all statistical tests does not exclude a possibility that the PNS is not suitable for cryptographic purposes. Before a PNS is accepted it should pass special cryptographic tests.

To check statistical quality of discovered rules and their cryptographic suitability some additional testing of rules has been performed. For this purpose uniform CAs consisting of 50 cells evolved during 65536 time steps with each single rule. Each of the 50 PNSs was divided into 4-bit words and tested on general statistical tests such as the entropy, chi-square test, serial correlation test [6], and on a number of statistical tests required by the FIPS 140-2 standard [11], such as monobit test, poker test, runs test, and long runs test. The best results

were achieved by rules 30, 86, 101, 153 ($r = 1$) and 8 long rules. Rules 90, 105, 150 and 65 [15] working separately in uniform CAs obtained good results in test of entropy and long runs test, quite good results in serial correlation test and monobit test but were weak in chi-square test, poker test and runs test. However this set of rules working together in nonuniform CAs achieves good results. For this reason only 10 rules were removed from discovered set of rules, which were worse than Tomassini & Perrenoud rules.

Rules which passed tests were next expressed to a set of Marsaglia tests [7] - a set of 23 very strong tests of randomness implemented in the Diehard program. Only 11 tests passed all 23 Marsaglia tests. These are short rules 30, 86, 101, and long rules 869020563, 1047380370, 1436194405, 1436965290, 1705400746, 1815843780, 2084275140 and 2592765285.

The purpose of the last set of experiments was a selection of a small set of short and long rules for nonuniform CAs to provide a generation of very high quality RNSs suitable for the secret key cryptography. Simple combinations of different rules which passed all Marsaglia tests in nonuniform CAs have shown that resulting PNSs may have worse statistical characteristic than PNSs obtained with use of rules in uniform CAs. On the other hand, experiments with Tomassini & Perrenoud rules show that rules working separately worse can provide better quality working collectively. For these reasons rules 153 and some long rules which obtained very good results in general tests but not passed all Marsaglia tests were also accepted for the set of rules to search a final set of rules.

In the result of combining rules into sets of rules and testing collective behavior of these sets working in nonuniform CAs the following set of rules has been selected: 86, 90, 101, 105, 150, 153, 165 ($r = 1$), and 1436194405 ($r = 2$). The proposed set of rules give similar results in terms of general and FIPS140-2 tests with the ones proposed by Tomassini & Perrenoud [12]. However, the main difference between these results can be observed at the level of Marsaglia tests: while the new discovered set of rules passes all 23 Marsaglia tests, the Tomassini & Perrenoud set of rules passes only 11 tests.

The secret key K which should be exchanged between two users of considered CA-based cryptosystem consists of a pair of randomly created vectors: the vector R_i informing about assigning 8 rules to N cells of CAs and the vector $C(0)$ describing an initial binary state of CA cells. The whole key space has therefore the size $8^N * 2^N$. The key space is much larger than the key space of 1D CA-based system [15] $(4^N * 2^N)$ and and slightly greater than 2D CA-based system [16]. Therefore the proposed system is much more resistant for cryptographic attacks.

7 Conclusions

In the paper we have reported results of the study on applying CAs to the secret key cryptography. The purpose of the study was to discover a set of CA rules which produce PNSs of a very high statistical quality for a CA-based cryptosystem which is resistant on attempts of attacks. The main assumption of our approach was to consider nonuniform 1D CAs operating with two types of

rules. An evolutionary approach called CP was used to discover suitable rules. After discovery of a set of rules they were carefully selected using a number of strong statistical and cryptographic tests. Finally, the set consisting of 8 rules has been selected. Results of experiments have shown that discovered rules working collectively are able to produce PNSs of a very high quality outperforming the quality of known 1D CA-based secret key cryptosystems, which also are much more resistant for breaking cryptography keys that known systems.

References

1. P. Guan, Cellular Automaton Public-Key Cryptosystem, *Complex Systems* 1, 1987, pp. 51-56
2. H. Gutowitz, Cryptography with Dynamical Systems, in E. Goles and N. Boccara (Eds.) *Cellular Automata and Cooperative Phenomena,* Kluwer Academic Press, 1993
3. T. Habutsu, Y. Nishio, I. Sasae, and S. Mori, A Secret Key Cryptosystem by Iterating a Chaotic Map, *Proc. of Eurocrypt'91,* 1991, pp. 127-140
4. P. D. Hortensius, R. D. McLeod, and H. C. Card, Parallel random number generation for VLSI systems using cellular automata, *IEEE Trans. on Computers* 38, October 1989, pp. 1466-1473
5. J. Kari, Cryptosystems based on reversible cellular automata, personal communication, 1992
6. D. E. Knuth, *The Art of Computer Programming,* vol. 1 & 2, *Seminumerical Algorithms,* Addison-Wesley, 1981
7. G. Marsaglia, Diehard, *http://stat.fsu.edu/~geo/diehard.html,* 1998
8. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography,* CRC Press, 1996
9. A. Mroczkowski, Application of Cellular Automata in Cryptography, Master Thesis (in Polish), Warsaw University of Technology, 2002
10. S. Nandi, B. K. Kar, and P. P. Chaudhuri, Theory and Applications of Cellular Automata in Cryptography, *IEEE Trans. on Computers,* v. 43, December 1994, pp. 1346-1357
11. National Institute of Standards and Technology, Federal Information Processing Standards Publication 140-2: *Security Requirements for Cryptographic Modules,* U.S. Government Printing Office, Washington 1999
12. F. Seredynski, P. Bouvry, A. Y. Zomaya, Cellular Programming and Symmetric Key Cryptography Systems, in E. Cantu-Paz et al. (Eds.), *Genetic and Evolutionary Computation-GECCO 2003,* LNCS 2724, Springer, pp. 1369-1381
13. B. Schneier, *Applied Cryptography,* Wiley, New York, 1996
14. M. Sipper and M. Tomassini, Generating parallel random number generators by cellular programming, *Int. Journal of Modern Physics C,* 7(2), 1996, pp. 181-190
15. M. Tomassini and M. Perrenoud, Stream Ciphers with One- and Two-Dimensional Cellular Automata, in M. Schoenauer at al. (Eds.) *Parallel Problem Solving from Nature - PPSN VI,* LNCS 1917, Springer, 2000, pp. 722-731
16. M. Tomassini and M. Sipper, On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata, *IEEE Trans. on Computers,* v. 49, No. 10, October 2000, pp. 1140-1151
17. S. Wolfram, Cryptography with Cellular Automata, in *Advances in Cryptology: Crypto '85 Proceedings,* LNCS 218, Springer, 1986, pp. 429-432

A Monte Carlo Study of Continuous Non-Ising Phase Transitions in the 3D Ashkin-Teller Model Using the OpenMosix Cluster of Linux PCs

Lech Dębski[1], Grzegorz Musiał[1], and Jos Rogiers[2]

[1] Institute of Physics, A. Mickiewicz University, ul. Umultowska 85, 61-614 Poznań, Poland, gulf@spin.amu.edu.pl, gmusial@amu.edu.pl

[2] Instituut voor Theoretische Fysica, Katholieke Universiteit Leuven, Celestijnenlaan 200D, B-3001 Leuven, Belgium, jos.rogiers@fys.kuleuven.ac.be

Abstract. OpenMosix as the computing platform is presented and applied to the Monte Carlo study of continuous phase transitions, whose character was not unambigously resolved yet, in 3D Ashkin-Teller model on a cubic lattice. Calculating the critical exponent y_h, we show that these phase transitions are not of Ising character and we give some indications to establish their universality class. It is demonstrated that OpenMosix ensures stability and good load-balancing of the Linux PCs cluster, but the best performance is achieved when applying it to parallelized jobs, which allowed us to perform calculations with larger samples.

1 Introduction to OpenMosix

The OpenMosix cluster composed of Linux machines belongs to the group known under the name Single System Image (SSI) [1]. The first SSI clusters were IBM SysPlex and DEC ones. In these systems we could log into the cluster and perform the calculations not worrying about where the calculations will be performed and knowing that they will be carried out on the node providing the best conditions. OpenMosix works in the Linux system on computers compatible with the Intel and AMD processors. It provides a new dimension in scaling and construction of highly effective clusters with Linux installed. The main advantage of OpenMosix over other cache coherent (CC) systems is its ability to respond at runtime to unpredictable and irregular resource requirements by many users. The most noticeable properties of executing applications on OpenMosix are its adaptive resource distribution policy and the symmetry and flexibility of its configuration. Parallel calculations can be performed with OpenMosix allocating the best nodes to processes, similarly as in symetric multiprocessing.

The overall goal is to maximise the performance by efficient utilisation of the network-wide resources. This capability to assign and reassign processes is particularly important for ease-of-use and to provide an efficient multiuser, time-sharing execution environment. OpenMosix does not have a mater node or slave nodes. Each node works autonomously and takes decisions independently of the others. This permits a dynamic configuration of the system, each node

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 455–460, 2004.
© Springer-Verlag Berlin Heidelberg 2004

can take part in the cluster and share its work, or can leave the cluster without posing a threat of the system's instability. This also permits a construction of large scalable clusters or small clusters composed of a few nodes. We make use of the latter, consisting of 4 heterogeneous PCs. The scalability is achieved by incorporating randomness in the system control algorithms, where each node decides on the basis of partial knowledge of the states of the other nodes.

In this paper we demonstrate that OpenMosix is very useful as the computing platform for our numerical simulations, because it ensures stability, load-balancing, transparency and scalability of the cluster as well as accessibility of all its resources. We apply it for both, sequential and parallelized jobs.

2 The Ashkin-Teller Model

The Ashkin-Teller (AT) model [2] is one of the important reference points in statistical physics for many decades as it is nontrivial generalization of the Ising model. The interest to this model much increased after the Fan's paper appeared [3] in which he expressed the Hamiltonian H of this model $-H/(k_BT) = \sum_{[i,j]}\{K_2(s_is_j + \sigma_i\sigma_j) + K_4s_i\sigma_is_j\sigma_j\}$, by means of two independent Ising degrees of freedom, s_i and σ_i, residing on each lattice site. $[i,j]$ denotes the summation over nearest neighboring lattice sites, $K_i = -J_i/k_BT$, with $i = 2$ or 4, and T is the temerature. Moreover J_2 is the coupling of the nearest neighbor interaction between the degrees of freedom s_i as well as for σ_i, whereas J_4 is the coupling between the product of these degrees of freedom $s_i\sigma_i$.

The three-dimensional (3D) standard AT model has been analysed by the short series analysis and (for the first time) by the Monte Carlo (MC) method by Ditzian et al. [4] to obtain the approximate phase diagram. More precise results have been obtained by Arnold and Zhang [5] in the interval $0 < K_4/K_2 \leq 1$ for the first order phase transitions and in our previous papers [6] for the Ising-like phase transitions and in the regions where the phase transitions are also of the first order [7,8]. These results are summarized in Fig. 1 where all phases are shown and explained, using the notation of Ditzian et al. [4].

In our paper we take advantage of the OpenMosix cluster of Linux PCs for a Monte Carlo study of the character of continuous phase transitions in the 3D AT Model on the line AHK in the phase diagram presented in Fig. 1. Although the picture of this phase diagram seems to be rather complete, and in paper [8] it was shown that the phase transitions on the line AHK are continuous, the character of the transitions on this line was never determined and our preliminary results are the first real indication in this matter on the line HK.

3 Description of the MC Method

The MC simulations presented here are the extension of the simulations presented in [6,7]. As previously, we have generated equilibrium configurations of the finite-size cubic spin samples of the size $L \times L \times L$ $(L \leq 30)$ for fixed values of the model parameters, using the Metropolis algorithm. Periodic boundary

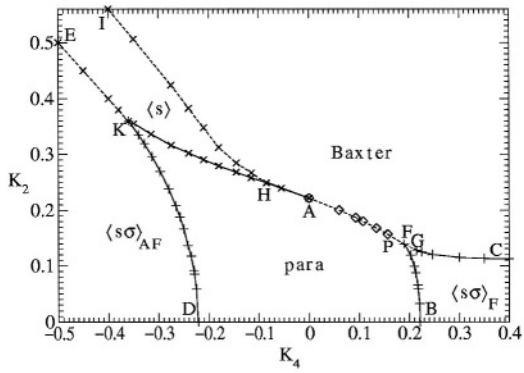


Fig. 1. The phase diagram of the 3D Ashkin-Teller model on a cubic lattice – the present state of knowledge. The broken lines denote the 1st order phase transitions, whereas the solid lines – the 2nd order ones. The precise results of Arnold and Zhang [5] are indicated with diamonds and our earlier results are marked with +'s [6] and with ×'s [8]. The phase labeled Baxter is ferromagnetically ordered with $\langle s \rangle$, $\langle \sigma \rangle$ and $\langle s\sigma \rangle$ non-zero, whereas in the phase labeled para they are all zero. In the phases "$\langle s\sigma \rangle_F$" and "$\langle s\sigma \rangle_{AF}$" $\langle s \rangle = \langle \sigma \rangle = 0$ and only the parameter $\langle s\sigma \rangle$ is ferro and anti-ferromagnetically ordered, respectively. For the phase "$\langle s \rangle$" two equivalent phases exist in which $\langle s\sigma \rangle = 0$ and either $\langle s \rangle$ or $\langle \sigma \rangle$ is ferromagnetically ordered but the other is not

conditions were imposed and thermalization of the initial configurations of the length of 10^5 to 10^6 Monte Carlo steps (MCS) was applied, depending on the size of a sample. The 48-bit random number generator was used. Each MC run was split into k $(6 \leq k \leq 40)$ segments consisting of 10^7 MCS and calculating partial averages. In the calculation of the partial averages only every i-th MC step contributes $(6 \leq i \leq 10)$, to avoid correlations between sampled configurations of spins in the system. CPU time used for one MC run varied from a few hours for the small samples ($L < 16$) up to a couple of weeks for largest ones.

Due to the well known fact that the true phase transition occurs only in the thermodynamic limit $L \to \infty$, in our analysis we use a finite-size scaling (FSS) relation between the magnetization M, temperature T and the size of the system L in the form [9] $L^{\beta/\nu} M(T - T_c, 1/L) = c_1 + c_2 (T - T_c) L^{1/\nu}$, where the amplitudes c_1 and c_2 are temperature- and size-independent. When temperature T is close to its critical value T_c, from this equation one obtains

$$L^{\beta/\nu} M(T \cong T_c) = c_1. \tag{1}$$

The relation (1) means that $L^{\beta/\nu} M$ should approach a horizontal line independently of L, if we have chosen a value for K_2 close enough to K_2^c at a given value of K_4 and we have used the proper values of the critical exponents β and ν.

The phase transition points were determined from the common intersection point of the curves [6,9,10] $U_L^{(\alpha)} = 1 - \langle M_\alpha^4 \rangle_L / (3 \langle M_\alpha^2 \rangle_L^2)$ versus K_2 atfixed K_4, where $\langle M_\alpha^n \rangle_L$ denotes the n-th power of the α spins order parameter, with $\alpha = s$,

σ or $s\sigma$, averaged over an assembly of independent samples of the size $L \times L \times L$. This analysis we call the intersection method.

4 Implementation of the Method

The simplest way to perform such calculations is to run sequential jobs in the Unix system only, but it is not satisfactory for our large-scale simulations. It was necessary to execute over 5000 MC runs to obtain the phase diagram presented in Fig. 1. Moreover, the larger the samples considered, the better the analysis of the results. We have used OpenMosix environment to accelerate the computational process, but obtaining of the results for samples with $L \geq 22$ in a realistic time forced us to parallelize our jobs. For this purpose we have used MPI working both, separately and within the OpenMosix environment (MPICH).

Our simlations are able to parallelize. Different processes of the parallelized job calculate different partial averages of the moments of an order parameter M. However, one has to remember to lead the system to the thermodynamical equilibrium independently on each of the p parallel processes. Speedup of such calculations (defined as t_{ser}/t_{par}, where t_{ser} and t_{par} denote time for computation with 1 process and with p parallel processes, respectively) tends to $1/p$ as the number of partial averages increases [7].

Table 1. Runtimes of 5 MC jobs in days (d) and quality of balancing. L denotes the linear size of simulated cubic samples of spins

	no OpenMosix, no MPI	only OpenMosix, no MPI	only MPI, no OpenMosix	MPICH within OpenMosix
L=8	1.10d	0.75d	0.68d	0.62d
L=16	4.16d	3.12d	2.60d	2.24d
L=24	23.48d	16.50d	14.68d	12.71d
balancing	poor	good	sufficient	very good

Our jobs ran on 4 Red Hat Linux PCs with different processors. In Table 1 we present runtimes for 5 MC jobs and quality of balancing of calculations. The first column contains the linear size L of simulated cubic samples of spins. In the second column is mean execution time of 5 sequential jobs running separately in the Linux system only (no OpenMosix and no MPI). The third column presents mean execution time when these sequential jobs were running simultaneously within OpenMosix environment (no MPI), whereas the fourth column contains execution time when one job is running 5 parallel processes (MPI only, no OpenMosix). The last column shows execution time of one job with 5 parallel processes running within OpenMosix environment (MPICH within OpenMosix). We can conclude that the common use of OpenMosix and parallelization library MPI considerably accelerate the computational process, even by the factor 2. Thus, best runtimes and balancing occur when simultaneously applying the MPI

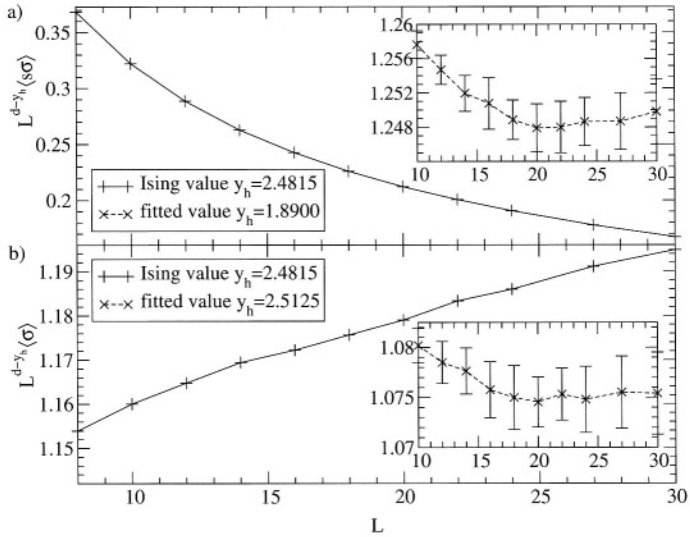


Fig. 2. Size-scaled values of the order parameter $L^{d-y_h}M$ with a) $M = \langle s\sigma \rangle$ and b) $M = \langle \sigma \rangle$ calculated at $K_4 = -0.055$ for different values of linear size L of samples. The values of the critical exponent y_h when calculating our MC data are given in the legend box, and the data obtained with the fitted value of y_h are presented in the insert

and OpenMosix. The biggest differences are noticeable for samples with biggest values of size L (the most time-consuming runs).

Assuming the Ising values of the critical exponents β and ν, and the critical value of the coupling K_2 carefully calculated with the intersection method (at the fixed value of K_4), we compute the dependency $L^{\beta_I/\nu_I}M(L)$. When this dependency for a critical point (K_4, K_2^c) evidently deviates from a horizontal line for $L \geq 16$ (the data for $L < 16$ should be discarded in a FSS analysis of this model [6,7]), one can conclude that this (continuous) phase transition does not belong to the Ising universality class. Then varying the value of the ratio β/ν we demand the dependency $L^{\beta/\nu}M(L)$ to fulfil the condition (1). Thus, fitting the dependency $L^{\beta/\nu}M(L)$ to a horizontal line we can determine the value of the critical exponent y_h by using the well known dependencies $\beta = (d - y_h)/y_t$ and $\nu = 1/y_t$. This way of calculation of y_h is our fitting procedure.

Applying the above mentioned intersection method at a fixed value of $K_4 = -0.055$ we have obtained the ordinate of the critical point $K_2^c = 0.23936(3)$. Moreover, from the analysis presented in the paper [8] follows that the phase transitions on the line AHK are continuous. Thus, assuming the Ising value of the critical exponent $y_h = 2.4815$ [11] and taking $\langle s\sigma \rangle$ and $\langle \sigma \rangle$ as an order parameter M, we have obtained the dependencies $L^{d-y_h}M(L)$ which are marked by +'s in Figs 2 a) and b), respectively. The uncertainties are of the order of the magnitude of the symbols. Their run by no means can be compared to the horizontal line. Applying our fitting procedure, we have calculated the critical exponent values $y_h = 1.890(5)$ and 2.5125(25) for $M = \langle s\sigma \rangle$ and $\langle \sigma \rangle$, respectively. With these

values of y_h, for $L \geq 16$ the curves $L^{d-y_h} M(L)$ evidently oscillate around a horizontal line within the error bars and are presented in the inserts of Figs 2 a) and b) with $\times$'s.

We have also obtained preliminary results for three points, with $K_4 = -0.18$, –0.275 and –0.35 where only the order parameters $\langle s \rangle$ and $\langle \sigma \rangle$ are critical, as explained in Fig. 1. The estimated values of the critical exponent y_h step by step become closer to its Ising value.

5 Concluding Remarks

OpenMosix uses the most of Linux PCs cluster resources. Moreover, migration of processes, ability of running of parallelized jobs and especially simplicity of scalability makes OpenMosix to be an effective, comfortable and relatively cheap tool for large-scale numerical calculations. It would be interesting to compare the efficiency of such MC simulations with MPI working in the OpenMosix environment and in other ones, like Condor eventually within GLOBUS or UNICORE.

We conclude that the line AH is the most complicated region of the phase diagram presented in Fig. 1, as suggested by Ditzian et al. [4]. We observe different behaviour for two components of the order parameter when the phase transitions occur along the same line. We may wander if this can be attributed to the tricritical behaviour, as the line AH is relatively short and is comparable to the scale of changes of the measured magnitudes. On the other hand we might attribute these effects to nonuniversality which could survive in this piece of the phase diagram with the change of dimensionality, as it is observed in 2D [4].

Acknowledgements. The authors wish to thank Professor G. Kamieniarz for valuable discussions, the Poznań Supercomputing and Networking Center for the access to the SGI Power Challenge and SGI Origin 3200C supercomputers, and the State Committee for Scientific Research for partial financial support within the grant 4 T11F 014 24.

References

1. www.openmosix.org
2. J. Ashkin and E. Teller, Phys. Rev. **64,** 178 (1943).
3. C. Fan, Phys. Lett. **39A,** 136 (1972).
4. R.V. Ditzian, J.R. Banavar, G.S. Grest, and L.P. Kadanoff, Phys. Rev. **B22**, 2542 (1980).
5. P. Arnold and Y. Zhang, Nuclear Phys. **B501,** 803 (1997).
6. G. Musiał, L. Dębski, and G. Kamieniarz, Phys. Rev. **B66,** 012407 (2002).
7. G. Musiał and L. Dębski, Lect. Notes in Comp. Scie. **2328,** 535 (2002).
8. G. Musiał, phys. stat. sol. (b) **236**, 486 (2003)
9. K. Binder and D.W. Heerman, Monte Carlo Simulation in Statistical Physics, Springer Series in Solid State Physics, Vol. 80, Springer-Verlag, Berlin 1988.
10. K. Binder and D.P. Landau, Phys. Rev. **B30,** 1877 (1984).
11. H.W.J. Blöte, E. Luijten, and J.R. Heringa, J. Phys. A: Math. Gen. **28**, 6289 (1995); A.L. Talapov and H.W.J. Blöte, J. Phys. A: Math. Gen. **29**, 5727 (1996).

Parallelization of the QC-Lib Quantum Computer Simulator Library

Ian Glendinning and Bernhard Ömer

VCPC
European Centre for Parallel Computing at Vienna
Liechtensteinstraße 22, A-1090 Vienna, Austria
http://www.vcpc.univie.ac.at/qc/

Abstract. We report on work to parallelize QC-lib, a C++ library for the simulation of quantum computers at an abstract functional level. After a brief introduction to quantum computing, we give an outline of QC-lib, then describe its parallelization using MPI, and present performance measurements made on a Beowulf cluster. Using more processors allowed larger problems to be solved, and reasonable speedups were obtained for the Hadamard transform and Grover's quantum search algorithm.

1 Introduction

Quantum computers are devices that process information using physical phenomena unique to quantum mechanics, and which have the potential to be able to solve certain problems such as prime factorization spectacularly faster than any conventional computer [1]. In a classical computer the basic unit of information is the bit, a two-state device that can represent the values 0 and 1. The quantum analogue of the bit is a two-state quantum system, such as an electron's spin or a photon's polarization, which has come to be known as a *qubit.* The difference between a qubit and a bit is that a qubit can exist not only in the states 0 and 1, but also in a mixture of both of them, called a superposition state. Furthermore, whereas a register of n bits can be in any one of 2^n states, storing one of the numbers 0 to $2^n - 1$, a register of n qubits can be in a superposition of all 2^n states, and a function applied to a quantum register in a superposition state acts on all 2^n values at the same time! This is known as quantum parallelism, and it is one of the key ingredients in the power of quantum computers.

Unfortunately, when a quantum register in a superposition state is measured, the result obtained is only *one* of the 2^n possible values, at random. However all is not lost, as the probabilities of measuring the different values can be manipulated by operating on a quantum register with quantum gates, which are the quantum analogue of logic gates. Quantum algorithms consist of sequences of quantum gate operations and optionally measurements, and it turns out that algorithms exist that are able to exploit quantum parallelism, and to leave an output register in a state where the probability of obtaining the value that is the answer to the problem is very close to one, giving an advantage over classical algorithms.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 461–468, 2004.
© Springer-Verlag Berlin Heidelberg 2004

However, building quantum computers is a huge technological challenge, and quantum computing hardware is not currently available outside physics research labs, so simulators present an attractive alternative for experimenting with quantum algorithms. Furthermore, they offer the only way to run programs on more than seven qubits, which is the current state of the art in experimental hardware. Simulators also help debugging of quantum programs, allowing direct examination of the quantum state, which is not possible in physical quantum computers. Simulators suffer from a problem, which is that their execution time and memory requirements increase exponentially with the number of qubits. Parallelization alleviates this problem, allowing more qubits to be simulated in the same time or the same number to be simulated in less time. Many simulators exist, but few for parallel systems. Niwa et al. [2] describe one and review related work.

2 Qubits, Registers, and Gates

The state of a qubit can be represented by a two-dimensional complex vector of length 1. The states that are the quantum analogues of 0 and 1 are called the computational basis vectors, and they are written $|0\rangle$ and $|1\rangle$, in a notation due to Dirac. In terms of vectors, they are conventionally defined to be

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad , \tag{1}$$

and a general qubit state is

$$\alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad , \tag{2}$$

where α and β are complex numbers called *amplitudes*. Measurement of the state always gives either $|0\rangle$, with probability $|\alpha|^2$, or $|1\rangle$, with probability $|\beta|^2$, which is consistent with the *normalization* condition that the vector's length is 1, which is $|\alpha|^2 + |\beta|^2 = 1$.

The state of an n-qubit register can be represented as a 2^n-dimensional complex vector of length 1. If we call the ith basis state $|i\rangle$, where $0 \leq i \leq 2^n - 1$, then a general n-qubit state has the form

$$\sum_{i=0}^{2^n-1} \alpha_i |i\rangle \quad , \tag{3}$$

where α_i is the ith complex component of the vector representing the state, $|\alpha_i|^2$ is the probability that measurement will give the value i, and the normalization condition is $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. The numbers labelling the basis states are often written in binary, to show the value of each component qubit in the register. For example, the computational basis vectors for a two qubit register are

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad . \tag{4}$$

Any n-qubit gate (operator) can be represented as a $2^n \times 2^n$ unitary matrix, i.e. a complex matrix U with the property that $U^\dagger U = I$. The operation of a gate on a quantum register is implemented by matrix multiplication. The only non-trivial classical single-bit gate is the NOT gate, but there are many non-trivial single-qubit gates, for example the *Hadamard* gate:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad . \tag{5}$$

This gate is useful because applying it to either of the basis states produces an equal mixture of both of them: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

The prototypical multi-qubit gate is the *controlled*-NOT or CNOT gate. It has two inputs, known as the *control* and *target* qubits, and two outputs. If the control qubit is set to 0, the target qubit is unchanged, and if the control qubit is set to 1, the target qubit is flipped ($|c,t\rangle$):

$$|00\rangle \to |00\rangle; \;\; |01\rangle \to |01\rangle; \;\; |10\rangle \to |11\rangle; \;\; |11\rangle \to |10\rangle \;\; . \tag{6}$$

3 QC-Lib

QC-lib is a C++ library for the simulation of quantum computers at an abstract functional level [3], and it is used as the back end of an interpreter for the QCL Quantum Computation Language [4]. Its main features are basis vectors of arbitrary length (not limited to word length), efficient representation of quantum states using hash tables, nesting of substates and arbitrary combinations of qubits, composition and tensor product of operators (gates), and easy addition of new operators using class inheritance. The top-level classes in QC-lib are:

bitvec – arbitrary length bit vectors which represent basis states
term – a basis vector with a complex amplitude
termlist – a list of terms: the internal representation of a quantum state
quState – user class for quantum states
opOperator – user class for quantum operators

The data structure of class `termlist` is a linear array in combination with a hash table. Only terms with non-zero amplitudes are stored, and the array and hash table are dynamically doubled in size if the array fills up.

The class `quState` contains two sub-classes, `quBaseState` which contains actual state information, and `quSubState` which represents substates. An object of class `quBaseState` represents the state of the whole quantum memory, and the `quSubState` class can be used to allocate subregisters. A `quBaseState` object contains two `termlist` objects. One contains the terms in the current state, and the other is a term buffer to accumulate the result of an operation on the state.

A sub-class of `opOperator` called `opMatrix` implements the representation of an n-qubit operator as a $2^n \times 2^n$ complex matrix, storing the non-zero elements of each row in an array of lists. However, most operators are simpler, only working

on a few qubits, or substituting one basis vector for another, with or without a phase factor, and `opOperator` has sub-classes for a number of such special cases, such as permutation of qubits, quantum functions, general single-qubit operations, and the CNOT gate.

The following example program uses QC-lib to implement the Hadamard transform, which is the Hadamard gate applied to every qubit in a register:

```
void H(quState& qs) {
  opBit H1(1,1,1,-1,sqrt(0.5));       // define Hadamard gate H
  for(int i=0; i<qs.mapbits(); i++) { // loop over qubits in register
    quBit qbit(i,qs);                 // create one-qubit substate
    H1.apply(qbit);                  } // apply H to the qubit
}
quBaseState qs(20); // allocate 20-qubit register qs
H(qs);              // apply the Hadamard transform to qs
```

4 Parallelization

The parallelization is being carried out in the message-passing style of programming, using the MPI message passing interface. The parallelization strategy is to distribute the representation of the quantum memory. Each processor stores a subset of the terms, and operators are applied only to local terms. The result of a local operation in general includes terms that are not owned by the processor, which must be communicated to their owning processors. The program is SPMD, with each processor running a copy of the same code.

The data distribution scheme for terms is that on 2^n processors, the n least significant qubits of their basis states are interpreted as the number of the processor that owns them. For example, the processor allocation of basis states for a four-qubit register on the four processors P_0–P_3 is:

P_0	$\|0000\rangle$, $\|0100\rangle$, $\|1000\rangle$, $\|1100\rangle$
P_1	$\|0001\rangle$, $\|0101\rangle$, $\|1001\rangle$, $\|1101\rangle$
P_2	$\|0010\rangle$, $\|0110\rangle$, $\|1010\rangle$, $\|1110\rangle$
P_3	$\|0011\rangle$, $\|0111\rangle$, $\|1011\rangle$, $\|1111\rangle$

4.1 Communication Pattern for Single-Qubit Operators

Consider a general single-qubit operator $U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$ operating on a single qubit with state $\alpha|0\rangle + \beta|1\rangle$. For simplicity, assume that there are just two processors, and the qubit in question is the least significant one, and so determines the data distribution. After the operation of U locally on each processor,

terms are created that are not owned by the processor, and so communication is necessary. Specifically, $\beta u_{12}|0\rangle$ has to be sent from $P_1 \rightarrow P_0$ and $\alpha u_{21}|1\rangle$ from $P_0 \rightarrow P_1$:

$$\begin{array}{l} P_0 \\ P_1 \end{array} \boxed{\begin{array}{c}\alpha|0\rangle \\ \beta|1\rangle\end{array}} \xrightarrow{U} \boxed{\begin{array}{c}\alpha U|0\rangle \\ \beta U|1\rangle\end{array}} = \boxed{\begin{array}{c}\alpha u_{11}|0\rangle + \alpha u_{21}|1\rangle \\ \beta u_{12}|0\rangle + \beta u_{22}|1\rangle\end{array}} \xrightarrow{\text{Comm}} \boxed{\begin{array}{c}(\alpha u_{11} + \beta u_{12})|0\rangle \\ (\alpha u_{21} + \beta u_{22})|1\rangle\end{array}}$$

When a single-qubit operator is applied to the ith qubit of a register, it is applied to the ith qubit of every term in the superposition state of the register, leaving the other qubits in each term unchanged. For each term in the initial state, at most two terms are therefore created (if one or more of the u_{ij} are zero, less terms will be created). If the ith qubit is not one of the qubits that determines the data distribution, then no communication is necessary, as both new terms are owned by the original processor. Otherwise, one of the new terms is locally owned and the other one is remotely owned and must be communicated. For each processor, all the remotely owned terms are owned by the same other processor, as a single bit has been flipped in the distribution key. Symmetrically, the remote processor potentially creates terms that are owned by the local one. In general therefore, bi-directional communication is needed between disjoint pairs of processors.

If the basis vectors of an n-qubit register are thought of as the coordinates of the corners of an n-dimensional hypercube, such that the ith qubit represents a coordinate of 0 or 1 in the ith dimension, then the communication pattern generated by an operation on the ith qubit can be visualized as being along parallel edges of the hypercube with data movement in its ith dimension.

4.2 Parallelization of Single-Qubit Operators

The representation of the distributed quantum memory has been encapsulated in class `quBaseState`, without altering the operator invocation mechanism in the sequential version of QC-lib. In the parallel version, each `quBaseState` object has a second term buffer (a `termlist` object), to accumulate terms destined for remote processors. The local and remote terms that result from the action of single-qubit operators are accumulated separately, and when the accumulation is complete, the remote term buffer is sent to the remote process, and reciprocally, the buffer that the remote process sent is received. This exchange of data between pairs of processes is implemented using the MPI function `MPI_Sendrecv()`. The received terms are then merged into the local term buffer. Finally, the term buffer is swapped with the termlist for the current state, so that the new terms become the current state, and the old one is ready to be used to accumulate the result of the next operation.

Currently only a few of the operator sub-classes of `opOperator` have been parallelized, but they include the general single-qubit operator and the CNOT

gate, which together are universal for quantum computation, though they don't necessarily offer the most efficient way of achieving a particular operation.

5 Performance Measurements

Performance measurements were made on a Beowulf cluster with 16 compute nodes, each having a 3,06 GHz Pentium 4 processor and 2 GByte of 266 MHz dual channel DDR-RAM, running Linux 2.4.19. The nodes were connected by two independent networks, Gigabit Ethernet for message-passing communication, and Fast Ethernet for job submission, file access, and system administration.

Figure 1 shows how the run time for the Hadamard transform varies with the problem size (number of qubits) for different numbers of processors. The run time for the sequential code is also shown, and it can be clearly seen from the semi-logarithmic plot that it increases by a constant factor for every extra qubit. Each extra qubit corresponds to a doubling of the problem size, which is consistent with the slope of the line, which corresponds to an increase in run time by a factor of 2.05 for each extra qubit. The parallel version of the program run on one processor has very little overhead compared with the sequential version, which is not surprising, as it does not perform any communication. When larger numbers of processors are used for smaller problem sizes there is more of an overhead, but as the problem size increases, the scaling behaviour of the parallel code becomes similar to that of the sequential version.

Figure 2 shows how the speedup varies with the number of processors, for various problem sizes. The speedup is relative to the run time of the sequential code for the same problem size, but since the sequential code ran out of memory for the larger problem sizes, the sequential run time for those cases had to be estimated, which was done by by linearly extrapolating a least squares fit to the the logarithm of the run time versus the number of qubits. For small problem sizes there is actually a decrease in speedup for larger numbers of processors, which is to be expected for any problem size if enough processors are used. For larger problem sizes reasonable speedups were obtained, up to 9.4 on 16 processors for the largest problem size.

The Hadamard transform is a sufficiently simple operation, that it was possible to make make runs that used all the system's available memory within a reasonable run time. It was found that both the sequential version of the program and the parallel version on one processor could simulate a maximum of 25 qubits, and that each doubling of the number of processors increased the maximum number of qubits by one, up to a maximum of 29 qubits for all 16 processors. This is consistent with the amount of memory needed by the program to store quantum states, which in the sequential version of the code, for quantum states with no more than 2^{32} terms, is 64 bytes per term. The parallel version of the code contains an extra `termlist` to buffer terms that need to be communicated, which means that it potentially needs up to 50% more memory, but in the Hadamard transform very few terms need to be communicated, as only changes to the least significant qubits cause communication, and these are pro-

cessed first, but large numbers of terms are not created until the most significant qubits are processed. Operating on n qubits, the Hadamard transform ultimately produces a state with 2^n terms, so the amount of memory needed to represent it is approximately $64(2^n) = 2^{n+6}$ bytes, and 25 qubits require 2 GByte, which was the amount of memory available. Not all algorithms use such a large number of terms with non-zero amplitudes, and as only these are explicitly represented by QC-lib, some algorithms can be run for still larger numbers of qubits.

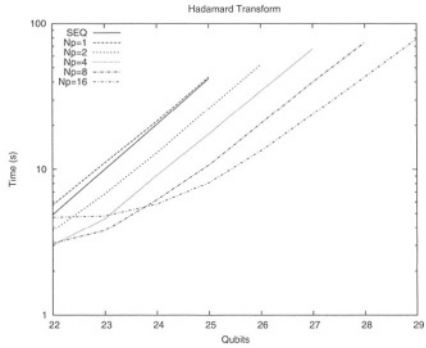


Fig. 1. Run time for Hadamard transform

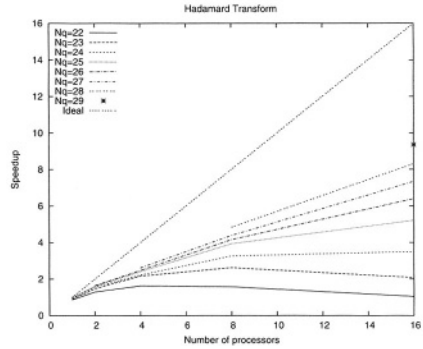


Fig. 2. Speedup for Hadamard transform

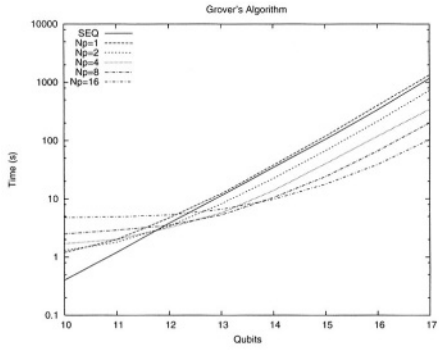


Fig. 3. Run time for Grover's algorithm

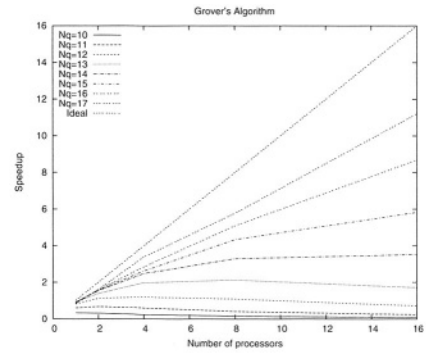


Fig. 4. Speedup for Grover's algorithm

Grover's quantum search algorithm [5] was also implemented using QC-lib, and the run time was measured for the portion of the algorithm up to the point when the first measurement is made. This was done to allow meaningful comparison of run times, as in the full version of the algorithm, there is a finite probability that the search will fail, and that it has to be repeated, so the run time is non-deterministic. Figure 3 shows how the run time for Grover's algorithm varies with the problem size, for different numbers of processors. The behaviour is similar to that in the case of the Hadamard transform, but in this case the

slope of the line for the sequential run time corresponds to an increase in run time by a factor of 3.10 for each extra qubit. This is partly accounted for by the fact that the number of iterations in the main loop of the algorithm increases by a factor of $\sqrt{2}$ for each extra qubit, and taken together with the doubling of the problem size, that would imply an increase in run time by a factor of 2.82.

Figure 4 shows how the speedup varies with the number of processors, for various problem sizes. Significantly better speedups are obtained than for the Hadamard transformation. For just 17 qubits, which is less than the minimum number of qubits considered for the Hadamard transform, a speedup of 11.2 was obtained on 16 processors, and it is to be expected that with more qubits the performance would be even better.

6 Conclusion

Sufficient functionality of QC-lib has been implemented to simulate universal quantum computation. We have implemented the Hadamard transform and Grover's algorithm using our library, and have made performance measurements for these codes. Promising speedups were obtained. Future work will include the implementation of static control of the distribution of qubits by the programmer, more operators, implementation of Shor's prime factorization algorithm [6], and the investigation of its performance. In the longer term, dynamic redistribution of qubits will be implemented and a load-balancing strategy will be developed.

Acknowledgements. This work was partially supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund FWF.

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
2. Niwa, J., Matsumoto, K., Imai, H.: General-purpose parallel simulator for quantum computing (2002) http://arXiv.org/abs/quant-ph/0201042.
3. Ömer, B.: Simulation of quantum computers (1996) http://tph.tuwien.ac.at/~oemer/doc/qcsim.ps.
4. Ömer, B.: Quantum Programming in QCL. Master's thesis, Vienna University of Technology (2000) http://tph.tuwien.ac.at/~oemer/doc/quprog/index.html.
5. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proc. of the 28th annual ACM Symposium on the Theory of Computation (Philadelphia, Pennsylvania), New York, ACM Press (1996) 212–219
6. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comp. **26** (1997) 1484–1509

Parallel Simulation of Czochralski Crystal Growth

Denis Lukanin[1], Vladimir Kalaev[2], and Alexander Zhmakin[3]

[1] Systems Analysis and Control Department, Saint-Petersburg Polytechnical University, 195251 Polytechnicheskaya street 21, Saint-Petersburg, Russia
denlu@softimpact.ru
[2] STR GmbH, Erlangen, Germany
kalaev@semitech.us
[3] A.F.Ioffe Physical Technical Institute, Russian Academy of Sciences, 194021 Polytechnicheskaya street 26, Russia
a@zhmakin.ru

Abstract. We present the results of parallel 3D simulation of Czochralski crystal growth. The description of the program is given and its parallel performance is estimated. The code verification is made by comparing the computed melt-crystal interface shape with the experiment.

1 Introduction

Recently, there have been many publications on the numerical simulation of semiconductor crystal growth. Rapidly growing interest in this field of research is due to the increasing demand on high quality semiconductor wafers which are important parts of microchips and solar cells. The most widely used method to grow bulk crystals is the Czochralski (CZ) technique, which was originally developed by the Polish engineer Jan Czochralski in 1916.

A larger crystal diameter reduces the wafer cost, therefore the general tendency is to grow large crystals. However, a number of problems arise, such as the radial non-uniformity of the physical properties of the crystal, cracks caused by strong thermal stresses , complex melt flows, and so on. Since the experimental optimization of the technology is very expensive, the most effective way to handle the problem is to use computer simulation.

A sketch of a 100mm crystal growth furnace is shown in Fig. 1a. It consists of a crucible, silicon melt, growing crystal which is slowly pulled from the melt upwards, one or more heaters, and several auxiliary parts. The crucible and the crystal revolve around the symmetry axis in opposite directions to ensure the cylindrical shape of the crystal. There is an inert gas flow in the furnace to remove parasitic impurities from the crystallization zone. The external reactor walls are maintained at room temperature.

A predictive computational model for the growth process should account for the following physical phenomena: the heat transport in the whole furnace (both conductive and radiative), the silicon melt turbulent flow, and the inert

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 469–474, 2004.
© Springer-Verlag Berlin Heidelberg 2004

gas flow. A predictive 2D/3D model for the simulation of 100mm and 300mm crystal growth was presented in [4]. The main drawback of the model is that it is very time-consuming: a typical unsteady computation with a grid of 300,000 cells takes about 10 days on a personal computer Intel P-III 1266Mhz. The present contribution is focused on the development of a parallel version of the 3D computational model, both for Linux clusters and SMP workstations running Windows.

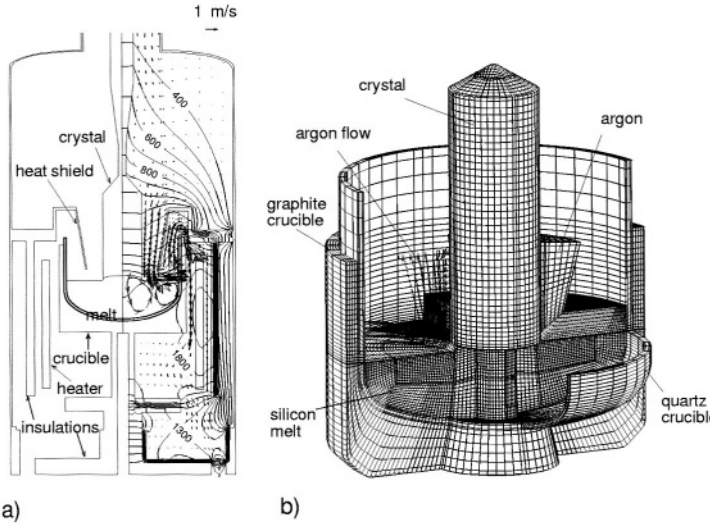


Fig. 1. Cross-section of the 100mm crystal growth furnace, computed temperature distribution and flow velocities (a), 3D computational grid containing crucible, crystal, melt, and inert gas blocks (b)

2 Computational Model

The 3D model was developed in terms of a quasi-steady approach. It considers a domain consisting of a crucible, silicon melt, crystal, and gas, as shown in Fig. 1b. The computation is made of the heat transport, turbulent melt and laminar gas flows, and oxygen impurity transport. Also, we provide a computation of the melt-crystal interface shape.

The governing equations of heat and mass transport are as follows:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \underline{u}) = 0, \tag{1}$$

$$\rho \left(\frac{\partial \underline{u}}{\partial t} + (\underline{u} \cdot \nabla)\underline{u} \right) = -\nabla \left(p + \frac{2}{3}(\mu + \mu_t)\nabla \cdot \underline{u} \right) + \nabla \cdot (2(\mu + \mu_t)\underline{\dot{S}}) + \rho \underline{g}, \tag{2}$$

$$\rho C_p \left(\frac{\partial T}{\partial t} + (\underline{u} \cdot \nabla) T \right) = \nabla \cdot ((\lambda + \lambda_t) \nabla T). \tag{3}$$

Here, $\underline{u}$ is the velocity, ρ is the temperature dependent density, T is the temperature, p is the reduced pressure, $\underline{\underline{S}}$ is the strain rate tensor, μ is the dynamic fluid viscosity, μ_t is the eddy viscosity, $\underline{g}$ is the gravity vector, λ is the conductivity, and λ_t is the turbulent effective conductivity. The eddy viscosity and the turbulent effective conductivity are calculated using the Large Eddy Simulation approach (LES), see [3]. For the material properties, we refer the reader to [2,4].

The set of equations above is supplemented with an oxygen transport equation and appropriate boundary conditions presented in Ref. [1]. The equations are approximated by the finite volume (FV) method. The thermal boundary conditions at the external boundaries are borrowed from the 2D axisymmetric model of global heat transfer in the furnace [2].

It is clear from the previous description that the model considers two fluid flows in one problem, coupled to the computation of the heat transfer in the domain. This approach allows one to provide a self-consistent simulation of the heat transport and to account for the gas flow effect on the silicon melt in a single computation.

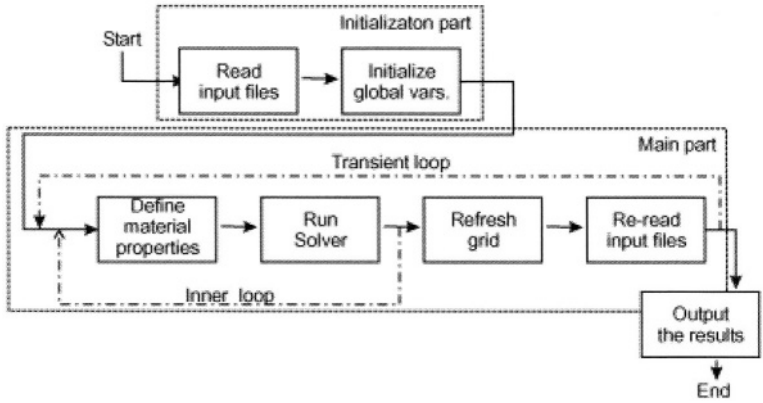


Fig. 2. Sequential program structure

3 Code Parallelization

3.1 Sequential Code

The sequential code structure is shown in Fig. 2. It consists of the initialization part, the main part, and the output of the results. The computation initialization implies reading the grid file containing the computational block structured grid, reading the parameters of the calculation, and calling several procedures which allocate and initialize global variables and arrays. The main part includes a

call to the procedure defining the material properties, a call to the solver for the governing equations, and a call to the melt-crystal interface correction procedure.

The most time-consuming part of the code is the solver which, in turn, involves an assemblage of matrices for the linear sets of equations and the computation of the subsequent approximation for each variable, e.g. temperature, velocities, pressure, etc. Therefore, when parallelizing, the main attention should be given to the solver acceleration.

3.2 Parallel Code

The parallel version of the program should satisfy the following requirements:

- the code should be portable among the variety of Fortran compilers;
- one should be able to run the program under MS Windows as well as under Linux operational systems;
- one should be able to run the program sequentially as well as in parallel.

For the parallel implementation we chose the MPICH message-passing library which fits the requirements well.

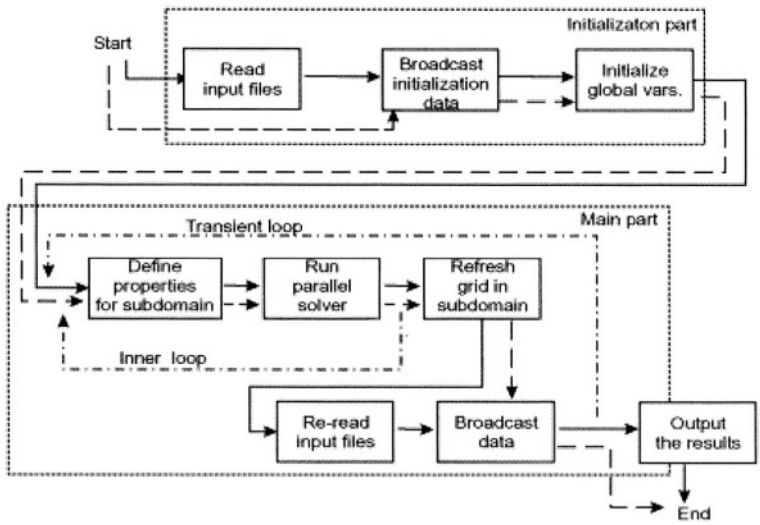


Fig. 3. Parallel program structure

The mapping of the problem onto a number of processors was made by the domain decomposition method. The parallel program structure is presented in Fig. 3. The root process path is marked by a solid line, while the other parallel process pathes are shown by a dashed line. The main part was modified to profit from parallel computations. First, the procedure defining material properties is called in the parallel version for the cells belonging to the subdomain and

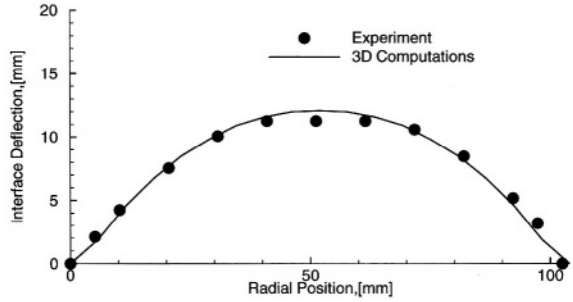


Fig. 4. Computed and experimental melt-crystal interface shape

adjacent to it. Second, the parallel version of the solver is made. This version assembles those linear equations which correspond to the subdomain, so that only part of the matrix is assembled on each processor. After the distributed set of equations is built, the iterative solver is called; we use the SSOR preconditioned BiCGSTAB algorithm (see, e.g. [5]). The parallel version of the algorithm implies two communication calls to refresh some vector elements and four calls providing distributed summation of scalars. So, each sweep of the linear solver contains six calls to the MPI communication procedures. Since every variable requires several iterations of the preconditioned BiCGSTAB to achieve convergence (about 3 for velocities and about 40 for the temperature and pressure), the iterative solver is the most critical program unit to the overall parallel code effectiveness.

Table 1. Parallel code speedup over a single processor one and efficiency

Proc.	Fast Ethernet	Myrinet
1	1 (100%)	1 (100%)
2	1.67 (84%)	1.7 (85%)
6	3.35 (56%)	4.0 (66.7%)
10	–	5.92 (59%)
20	–	9.15 (46%)

4 Results and Discussion

4.1 Parallel Performance

The parallel code performance was tested with a problem containing 154,000 cells on a Linux cluster. The cluster consists of 12 nodes, each of which has two Intel P-III 1266 processors. Two types of interconnection is used, namely the Fast Ethernet and Myrinet. The speedup of the parallel run over a single processor one and its efficiency for various numbers of processors are summarized in Table 1.

One can see that the parallel efficiency decreases relatively fast with the number of processors. This effect is due to the strong influence of the communication time on the total computation time for a large number of processors.

There are two obvious ways to reduce the communication time. First, one can decrease the number of neighboring cells for a particular subdomain by using an optimized domain splitting algorithm. Second, the number of iterative solver sweeps can be reduced by applying an improved preconditioner. However, both approaches require an additional effort to be made and are a subject of further study.

4.2 Code Verification

The verification of the parallel code was made by comparing the experimental melt-crystal interface shape with the computed one as shown in Fig. 4. One can see the coincidence between the computed and experimental interface geometries is fairly good.

5 Conclusion

The parallel version of the computational code was designed for the simulation of Czochralski crystal growth. It allows one to perform a 3D simulation within a period of 1–2 days, in contrast to 10 days on a personal computer. The code can be used under both Linux and Windows operational systems. However, there are still several unsolved problems with the current version of the parallel program, for example, its effectiveness on a large number of processors should be improved.

References

1. V.V. Kalaev, V.A. Zabelin and Yu.N. Makarov : Modeling of Impurity Transport and Point Defect Formation during CZ Si Crystal Growth. Solid State Phenomena 82-84 (2002) 41-46.
2. V.V.Kalaev, I.Yu.Evstratov, Yu.N.Makarov : Gas flow effect on global heat transport and melt convection in Czochralski silicon growth. J. Crystal Growth 249 (2003) 87-99.
3. N.G.Ivanov, et al. : Analysis of magnetic field effect on 3D melt flow in CZ Si growth. J. Crystal Growth 250 (2003) 183.
4. V.V.Kalaev, et al. : Calculation of bulk defects in CZ Si growth: impact of melt turbulent fluctuations. J. Crystal Growth 250 (2003) 203-208.
5. R.Barret and M.Berry and T.F.Chan and J.Demmel and J.Donato and J.Dongarra and V.Eijkhout and R.Pozo and C.Romine, and H.Van der Vorst : Templates for the solution of linear systems: Building Blocks for Iterative Methods, SIAM, 1994, Philadelphia, PA.

Application of Parallel Computing in the Transfer-Matrix Simulations of the Supramolecular Rings

Ryszard Matysiak[1], Monika Haglauer[2], Grzegorz Kamieniarz[2], Alvaro Caramico D'Auria[3], and Filippo Esposito[3]

[1] Institute of Engineering and Computer Education, University of Zielona Góra, ul. Wojska Polskiego 69, 65-625 Zielona Góra, Poland, r.matysiak@eti.uz.zgora.pl
[2] Computational Physics Division, Institute of Physics A. Mickiewicz University, ul. Umultowska 85, 61-614 Poznań, Poland, {monika, gjk}@amu.edu.pl
[3] Dipartimento di Scienze Fisiche, Università di Napoli, Piazzale Tecchio, 80125 Napoli, and INFM Unità di Napoli, Italy

Abstract. Numerical transfer-matrix approach and exact diagonalization technique exploiting the point-group symmetry and the properties of the shift operator are worked out in the framework of quantum statistical mechanics. They are applied, using the Heisenberg spin Hamiltonian and the large-scale simulations based on the parallel algorithm exploiting the Message Parallel Interface (MPI) system library, to the supramolecular rings describing some Mn and Ni clusters ($[Mn(hfac)_2NITPh]_6$ and $Ni_{12}(O_2CMe)_{12}(chp)_{12}(H_2O)_6(THF)_6$, respectively). For Ni complex the energy level structure is analyzed as a function of anisotropy where as for the Mn cluster the in-field magnetization profile is calculated.

1 Introduction

The synthesis of high nuclearity metal complexes has provided a new dimension to the field of nanomagnetism and has resulted in a series of new molecules with unusual geometric symmetries and patterns [1]. Their magnetic properties, associated to a large number of interacting paramagnetic centers in a single aggregate, have significantly stimulated the research effort with the prospect of technological applications [2].

The interest in spin assemblies stems from the fact they set the low-size limit for magnetic nanoparticles. They may display magnetic quantum tunneling and quantum-size effects in the thermodynamical properties [1]. Spin aggregates are embedded as the cores in macromolecules with well-defined size, shape and stoichiometry [2]. They are separated from one another by shells of ligand molecules and can be synthesized in macroscopically large samples in regular structures.

The usual way to characterize molecular magnets is to model them in terms of the spin hamiltonians [2]. However, characterization of the polynuclear magnetic clusters, even in terms of the spin models, remains a challenging task [3] and the role of the numerical simulations is very important in this field. So far, numerical

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 475–480, 2004.
© Springer-Verlag Berlin Heidelberg 2004

diagonalization techniques have been exploited to calculate their thermodynamic properties. Unfortunately, due to the exponential computing time and storage complexity, only in a few fortunate cases, referring to some simplified approaches, quantitative interpretations are available for larger systems [4].

The aim of this paper is to improve the diagonalization technique and to adapt the quantum transfer-matrix technique to the high nuclearity metal complexes with Ni and Mn ions. To achieve that the coarse-grained parallelization of the computer code is exploited.

2 Simulation Techniques

The magnetic finite-temperature features of large metal ion clusters may be quite accurately represented by the Heisenberg model. In the presence of the external magnetic field and with the nearest neighbor interactions, we consider the ring in the framework of the isotropic spin model Hamiltonian with periodic boundary conditions:

$$\mathcal{H} = \sum_{i=1}^{N} \left[J \left(\mathcal{S}_i^x \mathcal{S}_{i+1}^x + \mathcal{S}_i^y \mathcal{S}_{i+1}^y + \mathcal{S}_i^z \mathcal{S}_{i+1}^z \right) + g\mu_B B \mathcal{S}_i^z \right] , \tag{1}$$

where J denotes the nearest neighbor interaction constant (positive for antiferromagnetic coupling), B is the external magnetic field applied along the z direction, g is the corresponding gyromagnetic ratio and N stands for the number of sites in the ring. $\mathcal{S}_i^\nu$ is the ν-component of the spin $\mathbf{S}_i$ located at the i-th site of the ring. The spin values $\mathbf{S}_i$ may be uniform or non-uniform.

The advantages of the quantum transfer matrix simulation method have been demonstrated for the macroscopic magnetic chains [5,6]. The results are not subject to any statistical nor systematic errors and the macroscopic limit can be directly evaluated from the largest eigenvalue of the transfer matrix. For the finite rings, however, all terms in the definition of the partition function bring some contribution, so that the computational complexity of the QTM method increases enormously and the application of parallel computing becomes indispensable.

In the framework of the transfer-matrix method, first we express Hamiltonian (1) in terms of the spin-pair operators $\mathcal{H}_{i,i+1}$:

$$\mathcal{H}_{i,i+1} = J \left(\mathcal{S}_i^x \mathcal{S}_{i+1}^x + \mathcal{S}_i^y \mathcal{S}_{i+1}^y + \mathcal{S}_i^z \mathcal{S}_{i+1}^z \right) + \frac{1}{2} g\mu_B B \left(\mathcal{S}_i^z + \mathcal{S}_{i+1}^z \right) . \tag{2}$$

Then the series of the classical approximants of the quantum thermal values can be found, using the general Suzuki-Trotter formula. The partition function is calculated from the expression

$$\mathcal{Z} = \lim_{m \to \infty} \mathcal{Z}_m = \lim_{m \to \infty} \mathbf{Tr} \left[\prod_{i=1}^{N/2} \mathcal{V}_{2i-1,2i} \prod_{i=1}^{N/2} \mathcal{V}_{2i,2i+1} \right]^m , \tag{3}$$

where $\mathcal{V}_{i,i+1} = e^{-\beta \mathcal{H}_{i,i+1}/m}$, $i = 1, 2, \cdots, N$.

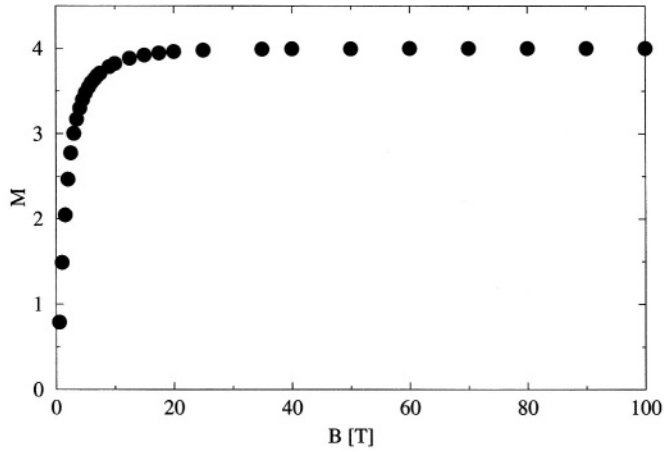


Fig. 1. The magnetization data for Mn cluster as a function external magnetic field.

We define also a unitary shift operator $\mathcal{D}$, acting in the space $\mathcal{H}^N$ which is a direct product of N single-spin spaces $\mathcal{H}_i$

$$\mathcal{D} \equiv \sum_{s_z^1} \cdots \sum_{s_z^N} \mid s_z^3 \ldots s_z^N s_z^1 s_z^2 \rangle \langle s_z^1 s_z^2 s_z^3 \ldots s_z^N \mid , \tag{4}$$

where s_z^i is the eigenvalue of the z component of the spin operator $\mathbf{S}_i$. Using Equation (4), we may express any operator $\mathcal{V}_{i,i+1}$ in terms of the operators $\mathcal{V}_{1,2}$ and $\mathcal{V}_{2,3}$

$$\mathcal{V}_{2i-1,2i} = \left(\mathcal{D}^+\right)^{i-1} \mathcal{V}_{1,2} \mathcal{D}^{i-1} , \quad \mathcal{V}_{2i,2i+1} = \left(\mathcal{D}^+\right)^{i-1} \mathcal{V}_{2,3} \mathcal{D}^{i-1} . \tag{5}$$

The definition (5) lead to the following m-th classical approximant of the partition function (3)

$$\mathcal{Z}_m = \mathbf{Tr} \left(\mathcal{W}_1 \mathcal{W}_2\right)^m , \tag{6}$$

where $\mathcal{W}_i = \left(\mathcal{V}_{i,i+1} \mathcal{D}^+\right)^{N/2}$; $i = 1, 2$. The numerical implementation of (6) is based on two global transfer operators $\mathcal{W}_i$ $(i = 1, 2)$ expressed by the sparse matrices defined by the operators $\mathcal{V}_{i,i+1}$ $(i = 1, 2)$. The latter operators stand for local transfer ones. The explicit form of the operator $\mathcal{V}_{i,i+1}$ and that of $\mathcal{D}$ depends on the choice of the spin operators in Hamiltonian (1).

The quantum transfer matrix technique can be applied for the model (1) with both uniform and non-uniform spin variables. The thermodynamic functions are related to the free energy $\mathcal{F} = -k_B T \ln \mathcal{Z} = -k_B T \ln \mathbf{Tr}\{e^{-\beta \mathcal{H}}\}$. The field-dependent magnetization is then evaluated from the first derivative of the free energy with respect to field.

To accomplish the exact numerical diagonalization in the case of the uniform spin variable $S = 1$ and non-vanishing single-ion anisotropy $A \sum_i \left(\mathcal{S}_i^z\right)^2$ (appropriate for Ni ions), we generalize technique [3] in order to exploit the total spin values as the quantum number. The details will be published elsewhere.

3 Results for the Mn and Ni Rings

The QTM approach is applied to calculate the magnetization of the ring with n=4 pairs of alternating spins $S_A = \frac{1}{2}$, $S_B = \frac{5}{2}$. For the dimensionless temperature $k_B T/J = 0.0286$ we have reached the accuracy up to the $4th$ decimal place pushing the Trotter number up to m=100, taking into account some symmetry properties of the transfer matrix and performing the effective parallelization of the code. The magnetization data for $T = 10$ K is drawn as a function of external magnetic field in Fig. 1. We have used the isotropic parameters: $J/k_B = 350 \pm 10$ K and $g = 2.0$ [3], appropriate for the real Mn compound.

For the first time, we take into account the single-ion anisotropy for modeling of the Ni rings [3]. We present here the energy level structure at the bottom of the spectrum as a function of anisotropy for the antiferromagnetic coupling. In Fig. 2 the interesting level crossing appears usually accompanied by some anomalies of the thermodynamical properties, we have considered the ring with $N = 10$ and the exchange parameter $J = 1$.

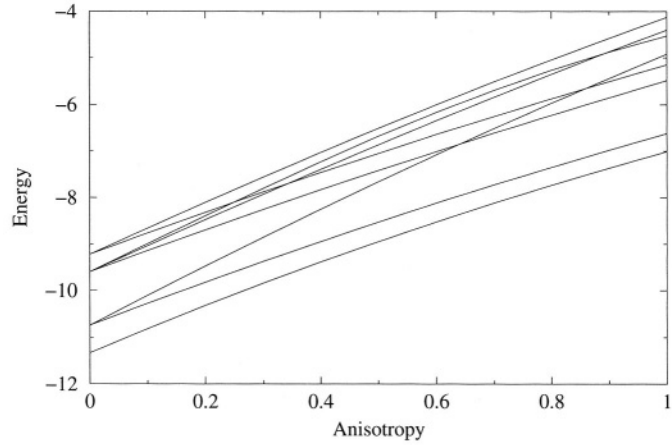


Fig. 2. The energy level structure for Ni cluster as a function of anisotropy parameter.

4 Parallel Implementation of the Transfer Matrix Simulations

The main numerical problem of our simulations is the calculation of the partition function $\mathcal{Z}_m$ which is given as a sum of the corresponding diagonal elements (6). The algorithm was optimised in this way that the global transfer operators $\mathcal{W}_1$ and $\mathcal{W}_2$ act only on the vectors different from the point of view of symmetry. Then for $N = 8$ spins in the ring the global transfer operators act on vector which dimension is equal $12^4 = 20736$ giving only 5226 different values. The

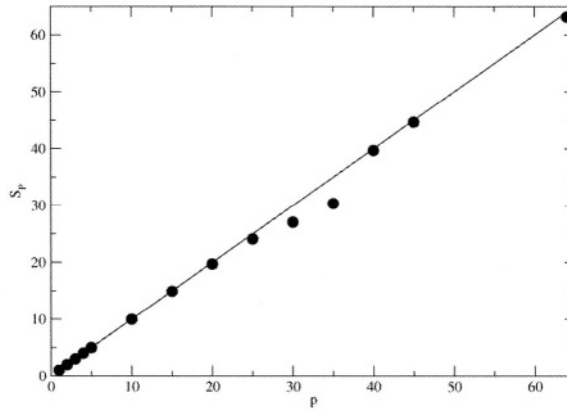


Fig. 3. Variation of the speed-up S_p against a number of processes p.

global transfer operators $\mathcal{W}_1$ and $\mathcal{W}_2$ can be expressed as product sparce matrices with known structure, which facilitates the numerical calculations. In the course of calculations in the memory of computer there are only two vectors and the allocation of the whole matrices $\mathcal{W}_1$, $\mathcal{W}_2$ in the memory is not required. In addition the code was fully vectorised when run on CRAY J916. The algorithm for evaluation of the partition function was also parallelised using Message Parallel Interface (MPI) system on SGI Origin 3800 with 64 processors which can work in parallel. The program was written in Fortran 90 programming language.

The calculation of the subsequent diagonal matrix elements in (6) was distributed among a number of processes. We have used coarse-grained homoparallelism. It means that the work was split on the identical (homogeneous) subtasks and each subtask was executed independently from the other [8]. The whole task was controlled by a single process called master. It sent subtasks to slave processes, which performed only one chunk of work. When the process was ready with its subtask, it sent the results to the controlling master process.

We have tested the parallel version of the algorithm and we have calculated speed-up for different numbers of processes p. Speed-up S_p is defined as the quotient of the sequential CPU time divided by the maximum CPU time used by the slave processes plus the master process CPU time [9,10]. The speed-up versus a number of processes p is presented by the full circles in Fig.3. It is also compared with ideal line $S_p = p$ and shows that the efficiency [9] η_p is very close to 1. We have performed a number of runs. Those ilustrated in Fig.3 represent the computational task which needs about $1.4 \cdot 10^4$ seconds of the sequential CPU time.

5 Conclusions

We have worked out two approaches suitable to characterize the finite temperature magnetic properties of the title high nuclearity cyclic spin clusters with

large and alternating spins. For Mn cluster the quantum transfer-matrix technique provides the numerically exact results. The dimensionality of the multispin space allows us to perform the trace operation in the definition of the partition function so that the quantitative interpretation has been accomplished here. We could follow the same strategy for Ni_{12} system but due to a lower dimensionality of the spin space and our new algebraic group-theoretical approach it was possible to reduce the dimension of the total spin component subspaces to a maximum size of 6166 and to perform numerical diagonalization even for the anisotropic system. We would like also to emphasize that both approaches are effective and can be exploited for other polynuclear clusters. The transfer-matrix simulations can be easily parallelized for larger number of processors, using the master-slave model and run on the supercomputing platforms with parallel architecture.

Acknowledgements. This work was supported in part by the Committee for Scientific Research under the KBN grant No. 4 T11F 014 24. Numerical calculations were mainly carried out on the platforms of the Supercomputing and Networking Center in Poznań.

References

1. Gatteschi, D., Sessoli, R., Cornia, A.: J. Chem. Soc., Chem. Commun. (2000) 725–732.
2. Gatteschi, D., Caneschi, A., Pardi L., Sessoli, R.: Science **265** (1994) 1054–1058.
3. Kamieniarz, G., Matysiak, R., Caramico D'Auria, A., Esposito, F., Benelli, C,: Eur. Phys. J. B **23** (2001) 183–189.
4. Atkinson, I.M., Benelli, C., Murrie, M., Parsons, S., Winpenny, R.E.P.: J. Chem. Soc., Chem. Commun. (1999) 285–286.
5. Matysiak, R., Kamieniarz, G., Gegenwart, P., Aoki, H., Ochiai, A., Steglich F.: phys. stat. sol. (b) **237**, (2003) 549–552.
6. Kamieniarz, G., Matysiak, R., Caramico D'Auria, A., Esposito, F., Benelli, C,: Lecture Notes in Computer Science Vol. 2328. Springer-Verlag, Berlin Heidelberg New York (2002) 502–509.
7. Gatteschi, D., Pardi, L.,: Gazetta Chimica Italiana **123** (1993) 231–240.
8. Bauer, Barr E.: Practical Parallel Programming. Academic Press San Diego, Inc. (1992).
9. Van de Velde, Eric F.: Concurrent Scientific Computing. Springer-Verlag New York, Inc. (1994).
10. Błażewicz, J., Kaczmarek, J., Kasprzak, M., Węglarz, J.: Computational Methods in Science and Technology **1** (1996) 31–40.

Hierarchical Communication for the Parallel Simulations in the Distributed Environment

Rafał Metkowski and Piotr Bała

Faculty of Mathematics and Computer Science,
N. Copernicus University, Chopina 12/8, 87-100 Torun, Poland,
{rafmet, bala}@mat.uni.torun.pl

Abstract. In this paper we present novel approach to the parallel computing in the distributed environment. Described solution utilizes new communication algorithm, which adjusts communication pattern to the grid environment. Presented results show, that this approach reduces significantly communication latency and allows to speedup data transfer up to several times. In result, one can run parallel applications in the distributed environment.

1 Introduction

Grid computing [1] become new approach to the use of computer resources. At the same time, Beowulf type clusters started to play important role as computing platform for the academic and scientific users. Unfortunately there are not many tools available that support distributed computing in grid environment, especially for the parallel jobs run across available computational resources.

In contradiction, parallel computing, limited to the massive parallel systems, or recently to the single, tightly connected clusters is well established. The number of standard tools exists, for example widely used communication libraries such as MPI or PVM or implementations of the shared memory type communication such as *shmem* or OpenMPI. These tools provide currently good performance while used on the dedicated parallel systems as well as on the tightly coupled PC clusters. The communication efficiency significantly decreases at the clusters build with standard Ethernet interconnect, or while distributed resources are used. It is known that the main reason which prohibits efficient parallel simulations comes from the high communication latency, rather than from limited communication bandwidth [2].

This problem has been determined some time ago, but still there is lack of the satisfactory solution. Some progress has been done with reduction of the communication latency on the Ethernet based clusters both on the hardware and software domains. The more efficient Gigabyte Ethernet cards and switches have been available, as well as new card interfaces such as VIA [3] or Score [4] which reduce TCP/IP latency. Unfortunately these solutions cannot be used in the wide area network.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 481–488, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The present work describes new solution which enables execution of the parallel applications in the grid environment, especially on the number of the distributed clusters connected via Internet. The main idea is to reduce communication traffic between nodes by redesigning communication pattern and reducing number of the communication traffic between clusters.

This aim can be achieved with the existing tools only by significant application modification.

Approach presented in this work extends traditional flat communication model used in the standard communication libraries and modifies communication according to the physical communication topology. Similar idea has been investigated by other authors, but existing solutions are not satisfactory and have significant limitations. New communication model presented in this paper removes most of the deficiencies and allows to run parallel jobs on the distributed resources. Our implementation is based on the Pfortran and allows user to run applications with hierarchical communication without code redesign.

2 Related Work

The Message Passing Interface (MPI) is the most widely used tool in the domain of the parallel computing. As stated before, number of efficient implementations exist, but majority of them is design for the uniform, flat communication model which assumes equal communication cost between different processors. Some extension has been developed for the multiprocessor nodes, which can use shared memory for the inter-node traffic.

Recently metacomputing extensions called MPICH-G2 has been developed for the MPICH - popular MPI implementation. This approach is designed to use Globus Toolkit [5] as the main communication library. Because Globus Toolkit introduces significant communication overhead, MPICH-G2 can use native MPI implementations for communication between cluster nodes. It has been demonstrated [6] that this approach can be efficiently used for running parallel applications in the distributed environment. Unfortunately, MPICH-G2 assumes simple, two layer or flat, hierarchy of the connections and cannot be used the complicated communication pattern. Moreover, this tool can be used only with the very particular implementations of the MPI library. From the practical point of view, the most important is lack of support for the firewalls and IP translation. In particular MPICH-G2 cannot be used for the communication between clusters which use private IP addresses for the nodes.

PACX-MPI is another implementation of the MPI, optimized for metacomputing. The major goal of the library is to make MPI applications run on a cluster of MPP's and PVP's without any changes in the sources and by fully exploiting the communication subsystem of each machine. To reach this goal, PACX-MPI makes use of the vendor MPI library on the systems, however number of supported implementations is limited. This solution accommodate also two level hierarchy of the connections. The wide area communication is performed by the dedicated daemons which communicate with cluster nodes for the

local connections and with each other for intra-cluster traffic. As in the previous case, the private IP cannot be used.

The similar model has been realized in the MagPIE [7], system dedicated to perform collective communication in heterogeneous environment. MagPIE improves collective communication by recognition of local- and wide-area traffic and by implementing topology aware algorithm.

This work present new solution open to the more complicated communication hierarchy and capable to perform communication between clusters with private IP addresses.

3 Pfortran

A Fortran implementation of the Planguages, the *Pfortran* compiler extends Fortran with the Planguage operators which are designed for specifying off-process access [8]. In a sequential program the assignment statement specifies a move of a value at the memory location represented by j to the memory location represented by i. Planguages allow the same type of assignment, however, the memory need not be local, as in the following example in a two-process system

$$i@0 = j@1$$

stating the intention to move the value at the memory location represented by j *at process 1* to the memory location represented by i *at process 0.* With the @ operator one can efficiently specify broadcast of the value at memory location a for logical *process 0* to the memory location a on all processes: $a = a@0$

The other Pfortran operator consists of a pair of curly braces with a leading function, $f\{\}$. which represents the common case of a reduction operation where the function f is applied to data across all processes. For example, sum of an array distributed across *nProc* processes, with one element per process, is represented as: $sum = +\{a\}$. Although a is a scalar at each process, it is logically an array across *nProc* processes. With @ and {}, a variety of operations involving off-process data can be concisely formulated.

In the Planguage model, processes interact through the same statement. Programmers have access to the local process identifier called *myProc.* With *myProc,* the programmer distributes data and computational workload.

Pfortran translates parallel extensions to the native communication library which allows to provide different low level communication. This feature was extremely important for us and allowed to modify communication algorithm without modifications of the application source code.

The Pfrotran hs been successuly used for paralleliation of the large scale scientific applications, especially molecular dynamics code used in this work as a test case. This programing paradigm allows also for easy modifications of the underlying communication method with no changes of the application.

4 Communication Model for the Distributed Clusters

The main aim of the present work is to evaluate performance of applications in situation when it is run on clusters connected through the Internet. Currently we have considered homogeneous node architecture (Intel x86), with possible private IP addressing. The homogeneous node architecture does not imply full cluster homogeneity. We have to assume different versions of the operating system on the different clusters, as well as different implementations of the MPI communication library.

Real applications can adopt different communication schemes, however the most important one are broadcast of the data and global reduction of the arrays. In the first case implementation of algorithm is straightforward and requires only modifications of the Pfortran communication library, which calls native, eg. MPI, communication functions. In the second case, additional modification of the Pfortran preprocessor must be performed, since global operations are implemented by modifications of the source code.

As the first and obvious choice we have implemented tree communication pattern which treats all processors as nodes of binomial tree and communication is performed along the edges of that tree. The algorithm is very efficient in low latency interconnections. In the case when latency is high and the time it takes a message to get to the destination is longer than the time it takes the sender to start all messages it is better to use simple algorithm with flat structure.

The concept of our extension to the communication libraries was to introduce a daemon which must be run on host with public IP address. That daemon serves as an intermediary for communication between clusters. Current implementation is limited to only two clusters, but there is no restriction to extend it to serve larger number of nodes, as well as to communicate with other daemons.

Job is divided into separate MPI jobs. Despite of homogeneous environment it must be often compiled separately because of different implementations of the MPI libraries. It has to be started on each cluster separately, and the parallel execution is triggered by the communication daemon.

5 Implementation

The implementation required significant modifications of the communication libraries.

In particular Pfortran initialization subroutine (called pf_com) was modified to perform initial communication with the daemon which includes also information exchange on the total number of processes. During the registration phase the TCP connection to the daemon is opened, and information about number of processes in MPI job (returned by `MPI_COMM_SIZE`) and its rank (returned by `MPI_COMM_RANK`) is exchanged. Initialization procedure modifies local Pfortran variables `nProc` and `myProc` to reflect numeration introduced in daemon. Processes are then renumbered in the daemon based on the cluster they are running on. Daemon is written in C language using TCP/IP sockets and POSIX threads.

Table 1. Testbed configuration

	Cluster 1 N. Copernicus University, Toruń	*Cluster 2* ICM, Warsaw University, Warsaw
Nodes	8 SMP nodes (16 processors)	8 SMP nodes (16 processors)
Processor	Pentium III 450 MHz	AMD Athlon 1800+
Network	Gigabit Ethernet	Gigabit Ethernet
OS	Linux Red Hat 7.2 kernel 2.4.19 SCore - Cluster System Software	Linux Red Hat 7.2 kernel 2.4.18
MPI	MPICH-SCore 1.2.4	MPICH 1.2.4

The Pfortran communication library has been modified to detect whether communication is made with node in the same cluster. If node is located in the other cluster data is send through TCP link opened in the initialization procedure. No conversion is made since we assumed that all nodes have the same architecture. Existing Pfortran implementation is using `MPI_ANY_SOURCE` as rank of source. We had to change it since it would be inefficient to poll TCP/IP socket with every receive call. Performed modifications allowed to run existing parallel codes on the distributed clusters, even with private IP addressing. In the performance analysis this implementation is treated by us as reference one since communication is performed based on the native MPI scheme, with flat communication pattern. As it was expected the performance is far not satisfactory.

In order to improve performance we have modified broadcast and reduce operations. As stated before improvement of the broadcast could be done as modifications of the Pfortran communication subroutine `pf_brdst`, while improvement of reduction had to be performed based on the modifications in the preprocessor. Pfortran is using binomial tree algorithm for collective communication. We have modified algorithm in that way that only the first level communication is performed between clusters and the rest of communication is done inside a cluster. This solution allowed us to minimize number of the connections performed on the wide area networks, and therefore reduce significantly number of connections which take place over slower and high latency link. As it is presented below this solution significantly improved communication performance.

6 Results

Tests have been performed on two clusters located at the geographically separated places connected by the public Internet as described in the Table 1.

Performance of the broadcast operation has been tested for long arrays, as it takes place in practical applications eg. molecular dynamics.

```
1.       do 10 i = 1, 100
2.       c(1:20000) = c(1:20000)@0
3. 10    continue
4.       call psync
```

Similar code has been used to test reduce operation:

```
1.        do 10 i = 1, 50
2.        c(1:20000) = +{c(1:20000)}
3. 10     continue
4.        call psync
```

Tests has been performed in two different configurations:

Case 1. *Cluster 1* has been divided into two parts, each forming separate cluster, connected with Gigabit Ethernet.

Case 2. Test application has been launched on both clusters *(Cluster 1* and *Cluster 2),* connected with wide are network (public Internet). The daemon has been started on the separate node at the location 1.

Obtained results indicate poor performance of no topology aware (flat) broadcast (Table 2) and reduce operations (Table 3). The communication time of the improved solution is significantly smaller than obtained with the flat communication pattern. Moreover, time of global operations depends mainly on the speed of link between clusters and does not increase significantly with number of nodes. This behavior comes from fact, that global operations are performed mostly within clusters, and only small amount of data, which does not dependent on the total number of nodes is sent across the slow link. This fact allows us to predict that presented solution will also scale well with the larger number of nodes and servers.

Table 2. Times of the broadcast and reduction operations for different number of nodes for the **Case 1**. The flat and improved communication models have been used.

No. of nodes	broadcast			reduction		
	flat	improved	ratio	flat	improved	ratio
2	955	987	0.97	1578	1601	0.99
4	2320	994	2.33	4151	1792	2.31
8	4798	1014	4.73	8857	2002	4.42
16	14457	1190	12.1	16019	2570	6.23

Table 3. Times of the broadcast and reduction operations for different number of nodes for the **Case 2**. The flat and improved communication models have been used.

No. of nodes	broadcast			reduction		
	flat	improved	ratio	flat	improved	ratio
2	4049	4113	0.98	5518	6063	0.91
4	7589	4422	1.71	10138	6182	1.64
8	15188	4837	3.13	18957	6411	2.95
16	32225	5299	6.08	36532	6639	5.50

The performed tests of the novel communication method encourages us to use it for the production code. For this purposes we have used parallel version

of molecular dynamics code GROMOS [9]. It is known, that because of the large number of frequent communication molecular dynamics codes cannot be run efficiently on the clusters with slow communication channel, and on the distributed resources connected with wide area Internet.

In the Table 4 we have presented execution time for the parallel GROMOS code run on the two 8 node clusters connected with the Gigabit Ethernet (testbed *Case 1*).

Table 4. Comparison of the molecular dynamics execution time on the single cluster and on the two separate clusters connected with Gigabit Ethernet.

No. of nodes	single cluster		flat communication		improved communication	
	time [s]	speedup	time [s]	speedup	time [s]	speedup
1	298.3	1.0	298.3	1.0	298.3	1.0
2	165.6	1.8	230.9	1.3	225.7	1.3
4	99.8	3.0	189.8	1.6	172.1	1.7
8	72.2	4.1	208.7	1.4	145.9	2.0
16	68.4	4.4	286	1.0	148.7	2.0

As it is clearly seen from the data obtained on the single cluster, MD code requires significant amount of the data transfer between nodes and therefore performance depends on the communication latency. Especially no speedup is observed while flat communication algorithm is used. Our implementation allows for significant improvement of the code performance in the distributed environment. Obtained results, are still far from excellent, but this is limited by the inter-cluster communication, which is in this case Gigabit Ethernet.

7 Conclusions

In this paper we presented novel approach to the parallel computing in the distributed environment. Our solution, based on the hierarchical communication, allows to adopt transfer between nodes to the existing connectivity topology. Presented results show, that this approach reduces significantly communication latency and allows to speedup data transfer up to several times. In result, one can run parallel applications in the distributed environment, without modifications of the application. Our work allows to deal with clusters with private addressing, which cannot be handled by other existing solutions.

Acknowledgments. This work has been performed using resources provided by Interdisciplinary Centre for Mathematical and Computational Modelling at Warsaw University.

References

1. C. Kesselman I. Foster, *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufman Publishers, USA, 1999.
2. R. Metkowski, P. Bała, and T. Clark. The performance of different communication mechanisms and algorithms used for parallelization of molecular dynamics code. In R. Wyrzykowski, J. Dongara, M. Paprzycki, and J. Waśniewski, editors, *PPAM01,* Lecture Notes in Computer Science 2328, pages 151–161. Springer-Verlag Berlin, 2001.
3. A. Bertozzi, M. Panella, and M. Reggiani. idesign of a via based communication protocol for LAM/MPI suite. In K. Klockner, editor, *Procs. 9th Euromicro Workshop on Parallel and Distributed Processing,* pages 27–33, 2001.
4. Y. Ishikawa, A. Hori, H. Tezuka, F. O'Carroll, S. Sumimoto, H. Harada, and T. Takahashi. RWC PC cluster ii and Score cluster system software. In R. Buyya, editor, *High performance cluster computing: Architectures and systems,* volume 1, pages 646–660. Prentice Hall, 1999.
5. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. J. Sci. Appl.,* 11(2):115–128, 1997.
6. I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. Wide-area implementation of the Message Passing Interface. *Parallel Computing,* 24(12–13):1735–1749, 1998.
7. T. Kielmann, H. E. Bal, J. Maasen amd R. van Nieuwpoort, L. Eyraud, R. Hofman, and K. Verstoep. Programming environments for high-performance grid computing: The Albatross project. *Future Generation Computer Systems,* 18(8):1113–1125, 2002.
8. B. Babak. *Parallel programming with guarded objects.* PhD thesis, 1994. Research Report UH/MD.
9. P. Bała, T. W. Clark, and L. R. Scott. Application of PFortran and Co-Array Fortran in the parallelization of the GROMOS96 molecular dynamics module. *Supercomputing Journal,* 9:i61–68, 2001.

Stepwise Development of Distributed Interactive Simulation Systems

Tomasz Orłowski and Bogdan Wiszniewski

Gdańsk University of Technology
Faculty of Electronics, Telecommunications and Informatics
ul. Narutowicza 11/12, 80-952 Gdańsk
{eaglet,bowisz}@eti.pg.gda.pl

Abstract. Objects in distributed interactive simulation systems are unpredictable, i.e., are controlled by their local operators, and are remote, i.e., must rely on some transmission media to visualize dynamic scene from their local perspectives. The paper reports on the ongoing project[1] aimed at a methodology for developing distributed interactive simulation systems for real and simulated objects in a 3D dynamic scene, based on a generic model of a material point, operational characteristics of participating objects, and their trajectories. Suitability of the proposed methodology is briefly discussed using the example of a High Level Architecture (HLA) standard.

1 Introduction

Event-driven simulation systems involve distributed processes or objects that are not predictable and can change their state irregularly, e.g. as a result of a human operator action. Basic notions used for designing such systems include sets of: possible states and transitions of each object, events that can trigger state transitions, and moments of time when these events may occur. When objects can simulate actions of real objects (like humans, machines, physical phenomena, etc.) and can affect behavior of one another, the overall cooperation scheme becomes interactive and can be implemented with distributed object oriented programming technologies; objects send irregular updates on their state changes to other objects and receive updates form other objects. A typical architecture of a *Distributed Interactive Simulation (DIS)* system is shown in Figure 1.

The top layer is responsible for interaction with a local site operator including simulation of the locally operated object and visualization of a dynamic scene from the local site perspective, based on the updates received from other sites. The lower layer performs basic computations regarding filtering "in" updates relevant to the local site and filtering "out" updates relevant to other sites. The basic layer implements *Run-Time Infrastructure (RTI)* functionality, supporting information exchange across the underlying network. RTI functionality has already been established as an international standard, with two dominant lines of products: DMSO RTI [1] and MÄK [5].

[1] Funded by the State Committee for Scientific Research (KBN) grant T-11C-004-22

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 489–496, 2004.
© Springer-Verlag Berlin Heidelberg 2004

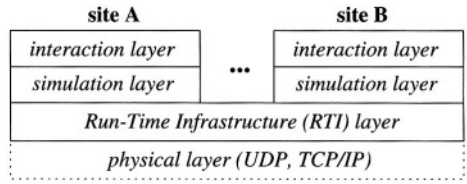


Fig. 1. Architecture of a DIS system

2 Modeling of Dynamic DIS Objects

Movements of material objects controlled by human operators can be predicted to some extent, according to the laws of dynamics. Existing event-driven simulation schemes attempt to address a wide spectrum of object movements that may be reported. Modeling state changes with stochastic functions can be useful in simulation of telecommunication systems, remotely controlled chemical processes, etc. [2]. A more radical reduction of message flow during a simulation experiment requires improvement of object state prediction by introducing the concept of relevance filtering [4].

In the paper we argue for expanding the basis of message filtering from just spatial characteristics to operational characteristics taking into account "semantics" of the object's behavior. Figure 2 outlines the proposed methodology.

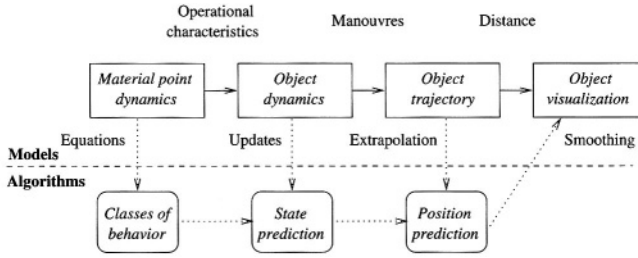


Fig. 2. Steps of modeling dynamic objects

2.1 Material Point Dynamics

Let assume that objects can move on the ground (a car or tank), or can fly (a helicopter or airplane). Each such object has mass, can be characterized with physical (kinesthetic) parameters like velocity, acceleration, etc., and behaves according to the Newtonian laws of dynamics. Classes of behavior that such a material object may exhibit are described by basic equations combining kinesthetic parameters of interest. For example, each class of behavior exhibited by an object (staying still, accelerating, climbing, turning, etc.) can be represented by a specific vector $\boldsymbol{S}$ of just a few related physical parameters, e.g. position

$\boldsymbol{L}$, orientation $\boldsymbol{O}$, linear velocity $\boldsymbol{v}$, linear acceleration $\boldsymbol{a}$, angular velocity $\boldsymbol{\omega}$, angular acceleration $\boldsymbol{\epsilon}$, resultant force $\boldsymbol{F}$, and resultant moment $\boldsymbol{M}$.

Based on inherent relations between various elements of each object's state vector $\boldsymbol{S}$ relevant parameters required for a correct object visualization can be calculated. For example, linear acceleration $\boldsymbol{a}$ of a moving object can be determined based on the resultant force $\boldsymbol{F}$ applied to the object. Similarly, angular acceleration $\boldsymbol{\epsilon}$ can be determined based on the resultant moment $\boldsymbol{M}$ of the object. Linear $\boldsymbol{v}$, and angular $\boldsymbol{\omega}$, velocities can be determined then by integrating the respective acceleration vectors. Integration of the respective time dependent parameters can be computed numerically, based on the rule of trapezium, known also as the modified Euler's method [3]:

$$P_n = P_{n-1} + \left(\frac{\Delta P_{n-1} + \Delta P_n}{2} \right) \cdot \Delta t \tag{1}$$

where P is a parameter of interest, P_n is a new calculated value of P, P_{n-1} is the last value of P, ΔP_{n-1} is the last change of the value of P, ΔP_n is the predicted change of the value of P, and Δt is the integration step.

Given current linear velocity $\boldsymbol{v}$ and the last reported position $\boldsymbol{L}_{n-1}$ of the observed remote object, the observing object may calculate the new position of the former in one step as:

$$\boldsymbol{L}_n = \boldsymbol{L}_{n-1} + \boldsymbol{v} \cdot \Delta t \tag{2}$$

Angular velocity $\boldsymbol{\omega}$ of a moving object is defined with regard to its local coordinate system. In order to visualize a reporting object at some remote display a conversion between the local and global coordinate system must be defined. We refer to this issue in Section 2.3.

2.2 Simulation Object Dynamics

The real meaning of behaviors represented by material point equations is implied by operational characteristics of individual objects. For example, a tank cannot accelerate faster than specified in its respective operations manuals. When combined with operational characteristics of the respective objects, equations describe a real object dynamics – a material point behavior means a distinguishable *maneuver,* with certain predictable limits imposed by the simulation object mechanics. In consequence, each maneuver is characterized by a specific state that has to be reported to remote objects. Figure 3 specifies behaviors of a real object "maneuvering" in a virtual scene, distinguishing just five states.

The initial state of a simulation object is *neutral,* i.e., it either remains still or is in a uniform straight-line motion (vectors $\boldsymbol{a} = \boldsymbol{0}$, $\boldsymbol{\epsilon} = \boldsymbol{0}$, $\boldsymbol{v} = const$). An object may switch (1) from a neutral state to a new maneuver *entering* state and stay in the latter as long as its vectors $\boldsymbol{a}$, $\boldsymbol{\epsilon}$ vary. The state of entering a maneuver may eventually lead to a stable state (2) representing *staying* in the maneuver; in such a case vectors $\boldsymbol{a} = const$, $\boldsymbol{\epsilon} = const$ and at least one of them is non-zero. Any subsequent increase (3) or decrease (4) of $\boldsymbol{a}$ or $\boldsymbol{\epsilon}$ angle or value

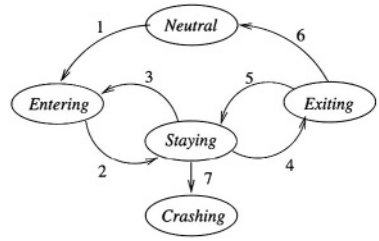


Fig. 3. State-graph of a simulated object

implies further *entering* or *exiting* a maneuver. Exiting a maneuver may end up with entering another maneuver (5) or returning (6) to a neutral state. A trap state, *crashing,* shown in Figure 3 is entered if at least one of the object parameters described before exceeds its operational limits, e.g. exceeding some acceleration limit would lead to disintegration of a real object being simulated.

2.3 Simulation Object Trajectory

Each maneuver can be represented with a specific trajectory that can be visualized with an object image moving in a scene and observed from various distances of other objects belonging to the scene. In order to reduce the volume of data required for calculating a remote object trajectory the following dependencies between parameters describing the state of a moving object may be considered:

1. If a reporting object W sends updates at the rate not less than the framing rate of a graphical system at a visualizing object T side only position $\boldsymbol{L}_w$ and orientation $\boldsymbol{O}_w$ of W are needed by T to correctly visualize W. This approach is straightforward but bandwidth consuming – in many cases it is not acceptable.
2. If at a given moment object T does not have current values of $\boldsymbol{L}_w$ or $\boldsymbol{O}_w$, but has their values sent by W earlier it:
   a) must possess recent values of $\boldsymbol{v}_w$ or $\boldsymbol{\omega}_w$, otherwise it must assume that neither $\boldsymbol{v}_w$ nor $\boldsymbol{\omega}_w$ has changed from the last reported values; in such a case only changes of $\boldsymbol{v}_w$ or $\boldsymbol{\omega}_w$ are needed;
   b) alternatively, T has to extrapolate value of $\boldsymbol{v}_w$ or $\boldsymbol{\omega}_w$ based on at least two recent values of $\boldsymbol{L}_w$ or $\boldsymbol{O}_w$;
3. If at a given moment object T does not have current values of information about $\boldsymbol{v}_w$ or $\boldsymbol{\omega}_w$, but has their values sent by W it:
   a) must possess recent values of $\boldsymbol{a}_w$ or $\boldsymbol{\epsilon}_w$, otherwise it must assume that neither $\boldsymbol{a}_w$ nor $\boldsymbol{\epsilon}_w$ has changed from the last reported values; in such a case only changes of $\boldsymbol{a}_w$ or $\boldsymbol{\epsilon}_w$ will be transmitted;
   b) alternatively, T has to extrapolate value of $\boldsymbol{a}_w$ or $\boldsymbol{\epsilon}_w$, based on at least two recent values of $\boldsymbol{v}_w$ or $\boldsymbol{\omega}_w$;

In a properly initiated simulation system, where each receiver (observer) has once got full information about each participant, for objects associated with decision events (maneuvers initiated by their human operators) only changes of their respective vectors $\boldsymbol{a}$ or $\boldsymbol{\epsilon}$ values are needed.

A system of local coordinates of any simulation object W corresponds to the global cube coordinates with a single rotation ϑ around the (global) axes of a global visualization scene with angles α, β and ζ. These four parameters are sufficient to define transformation of local position $\boldsymbol{L}$ of object W to its global position $\boldsymbol{L}_w$ in a normalized quaternion form [6]:

$$\boldsymbol{L}_w = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \boldsymbol{L} \qquad (3)$$

where $q_0 = \cos 0.5\vartheta$, $q_1 = \cos\alpha \sin 0.5\vartheta$, $q_2 = \cos\beta \sin 0.5\vartheta$, $q_3 = \cos\zeta \sin 0.5\vartheta$. The normalized quaternion form given by Formula 3 can be used for updating location $\boldsymbol{L}$ and orientation $\boldsymbol{O}$. In the case when angles α, β, ζ and ϑ are not initially known, values q_1, q_2, q_3 and q_4 can be calculated from the initial cosine matrix. Updating a quaternion based on angular velocities the following shall be calculated: $\dot{q_0} = 0.5(q_1\omega_x + q_2\omega_y - q_3\omega_z)$, $\dot{q_1} = 0.5(q_0\omega_x + q_2\omega_z - q_3\omega_y) + \lambda q_1$, $\dot{q_2} = 0.5(q_0\omega_y + q_3\omega_x - q_1\omega_z) + \lambda q_2$, $\dot{q_3} = 0.5(q_0\omega_z + q_1\omega_y - q_2\omega_x) + \lambda q_3$, where ω_x, ω_y, ω_z are components of $\boldsymbol{\omega}$, and $\lambda = 1 - (q_0^2 + q_1^2 + q_2^2 + q_3^2)$ is a correction factor of the numerical integration error.

2.4 Simulation Object Visualization

Having defined a model for determining a reporting object position in a global cube, let us now relate it to the local visualization capabilities of the observing object. Let assume first that two visualization parameters are given *a'priori* for the simulation system: maximum resolution $< X_{max}, Y_{max} >$ and refreshing frequency f_o of all visualization devices used by observers. These parameters determine the upper bound of the visualization capability across the system. In many cases frequency of updating state of visualized objects may be less than the refreshing frequency. The lower the quality of a visualizing device, the less frequent the updates of the reporting object's state should be. Similarly, lower velocity of a visualized object, or a greater distance from the observer will also require lower refreshing frequency. A question arises how many updates may be skipped before the visualized object may change its state so much that it can be noticed by the observer? Apparently, the minimum realistic change is one pixel. Therefore, for the reporting object the following information about a remote display is needed: horizontal (X_{max}), and vertical (Y_{max}) screen resolution at the remote site, horizontal (α_x) and vertical (α_y) angles of view at the observer, and distance h between the observed (reporting) and observing (receiving) objects measured in a virtual dynamic scene. With these parameters it is possible to calculate the size of a view base pyramid seen by the observer from distance h as $a = 2h\tan(0.5\alpha_x)$, $b = 2h\tan(0.5\alpha_y)$. Then one can calculate how big

observed object area could be contained in one pixel of the image rendered at the observer's side, as $1p_x = a/X_{max}$ and $1p_y = b/Y_{max}$.

These calculations clearly indicate that if the rendered image of a moving object is smaller than $1p_x$ and $1p_y$, it is pointless to update object's state because it will not be noticed by the observer. The screen refreshing frequency at the observer side f_o can be also used as an additional constraint; so if the frequency of sending updates is f_s we get $f_p \leq f_s \leq f_o$, where time required to move by one pixel ($1p_x$ or $1p_y$) of the visualized object is $T_p = 1/f_p$. If $f_p > f_o$ then the reporting object should send messages with just the f_o frequency. It is particularly important to off-line observers (who may refresh their screens once in a couple of seconds).

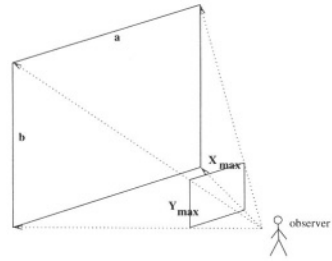


Fig. 4. A view base pyramid

3 Quality of Distributed Simulation

A methodology being developed in the reported project is aimed at achieving a satisfactory level of quality with regard to five quality attributes: *fidelity* defined as a degree of similarity between the training situation and the operational situation, *dependability* aimed at reducing a volume of state update information without compromising on fidelity, *scalability* allowing for expansion of distributed interactive simulation applications both in terms of a number of simulation objects and users operating them, *performance* indicating acceptable time limits for interaction between simulation objects, their graphical displays and human operators, and *stability* assuring acceptable performance regardless of a particular computing platform. Before implementing a pilot application (tanks and helicopters over land) several experiments were carried out to evaluate performance of the available platforms. Below we present briefly results obtained for DMSO RTI 1.3 v3.2 implementation and benchmarks [1], in a standard lab setting with Pentium 3 PCs and Fast Ethernet. One tested characteristic is *throughput,* and another is state update *intensity.* The former will be used for scaling the experimental application, while the latter will be used for controlling the relationship between dependability and fidelity.

Throughput characteristics indicate what are the limits for the number of updates that can be effectively sent over the RTI layer. The number of updates was

measured with regard to the total number of simulation objects participating in the experiment (Figure 5a), and with regard to the size of state update messages (Figure 5b). Two message transport modes provided by RTI were tested: *best effort* mode using datagrams and *reliable* mode using stream connections. It can be seen that if the message size is reasonably low, say 10B, the number of updates fits the range between 15.000 and 25.000 (Figure 5a). On the other hand, a reasonable number (up to 100 for example) of objects can fit that message volume if the message size is reasonably low, say again about 10B (Figure 5b). With a message of that size one or two floating point numbers per message can be sent. Therefore, setting up a distributed interactive simulation application with up to 100 objects which send updates with single elements of the state vector $\boldsymbol{S}$ described in Section 2.1 sholud be feasible.

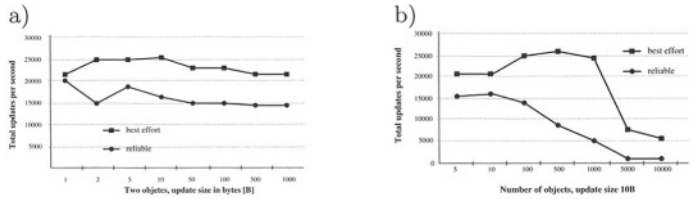


Fig. 5. Throughput for various: (a) state update size, (b) number of simulation objects

Intensity characteristics affect relationships between dependability and fidelity, e.g. by indicating the message loss ratio. A standard (window based) conservative synchronization algorithm provided by tested RTI was used, and its selected parameters (window size *lookahead,* and logical time step *(interval)* were tuned for the best performance. The test (single simulation execution) time in each experiments was set to 60 seconds. While in the throughput test a total number of updates in the system was counted, in the intensity test just two cooperating objects were observed. Results are shown in Figure 6.

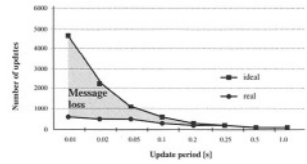


Fig. 6. Message loss with the varying reporting period

It can be seen that if the update period is more than 0.25 seconds, an *ideal* amount of updates (no message loss) and the *real* amount of updates (all messages correctly sent and received) begin to converge. This is a threshold value imposing a limit upon each object dynamism, in order to preserve reality of their

movements in a visualized scene. Intuitively, since most objects in the target DIS application are large and massive bodies, such a quarter of second based time scale for reporting their state changes seems to be quite realistic.

4 Summary

The project mentioned in this paper addresses a much wider range of problems then only modeling interactive phenomena with distributed objects. One important issue includes 3D visualization of a dynamic scene and the moving objects. At the current stage it is assumed that all participating simulation sites share one common source of spatial data about the terrain (a GIS database of a real piece of land in the Gdańsk region). It is planned in the near future to expand the dynamic scene range by incorporating more GIS sites and enable missions over a wider territory, like many regions or even the entire country. Further interesting extensions may involve connecting one or more meteorological servers to add yet more reality to the simulation exercises.

Another important issue is a simulator of a real material object. For the project a generic "component" based simulator has been developed. It provides such components as an engine, a propeller, a rotor, a fixed wing and an undercarriage, which are sufficient to assemble any realistic ground vehicle or aerodyne, i.e. any object that can fly by interacting with the atmospheric air. When assembling a model of a concrete object respective operational parameters have to be defined for each component, while the configured simulator instance can compute automatically relevant elements of the resultant state vector $\boldsymbol{S}$, before sending them out as state updates. In the current experimental DIS application three objects can be simulated: a tank, a light propeller airplane and two kinds of helicopters, with one and two rotors. The above mentioned application can run in a typical lab setting of PCs connected with Fast Ethernet and MS-Windows.

References

1. DoD. High Level Architecture interface specification. Version 1.3, IEEE P1516.1, http://hla.dmso.mil.
2. S.L. Ferenci, R.M. Fujimoto, M.H. Ammar, K. Perumalla, and G.F. Riley. Updateable simulation of communication networks. In *Proc. 6th Workshop on Parallel and Distributed Simulation,* pages 107–114, Washington, D.C., May 2002.
3. B. Goldiez and K. Lin. The orientation representation in the draft military standard for distributed interactive simulation. University of Central Florida, 1991.
4. B.S. Lee, W. Cai, S.J. Tirner, and L. Chen. Adaptive dead reckoning algorithms for distributed interactive simulation. *I. J. of Simulation,* 1(1-2):21–34, Dec. 2000.
5. MÄK. MÄK Technologies, http://www.mak.com.
6. T. Orłowski and B. Wiszniewski. Tracing of dynamic objects in distributed interactive simulation systems. In *Proc. 2nd Int. Conf. Applied Mathematics APLIMAT03,* pages 543–550, Bratislava, Slovakia, Feb. 5-7 2003.

Some Aspects of Parallel Performance of a Seismic Ray Analysis Algorithm*

Marcin Paprzycki[1], Boris Digas[2], and John Kopsky[1]

[1] Computer Science Department,
Oklahoma State University,
Tulsa, OK 74106, USA
[2] Institute of Mathematics and Mechanics,
Ural Branch of the Russian Academy of Sciences,
16 S.Kovalevskaya Str., 620219 Ekaterinburg, Russia

Abstract. We consider the reconstruction of the velocity structure from measurements of travel times of seismic surface waves. The main problem is reduced to finding solution of a system of linear inequalities in infinite-dimensional space. The solution process is based on a combination of parallelized versions of the Backus–Gilbert and the constraint aggregation methods. The experimental results obtained on the IBM SP parallel computer are presented and analyzed.

1 Problem Definition

Consider a domain $\Omega \in \mathbf{R}^2$ containing a large number of sources and receivers of seismic signals. Let $t_1, \ldots, t_n$ be measured travel times corresponding to different pairs "source–receiver". It is assumed that laws of geometrical optics describe the process of propagation of the signal. Each value of travel time is correlated with velocity characteristics of the medium by the following integral relationship:

$$t_i = \int_{l_i} \frac{dl}{v(r)} \quad (i = 1, 2, \ldots, n),$$

where $v(r)$ is the velocity of propagation of seismic waves; l_i is the integration contour corresponding to the i-th seismic ray. The problem of interest is to determine the values of function $v(r)$ at every point of the region under consideration.

This problem is nonlinear, but it can be linearized in a standard manner. For instance, let us consider a non-dimensional value

$$m(r) = \frac{v^{-1}(r) - v_0^{-1}(r)}{v_0^{-1}(r)}$$

* This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The second author was supported in part by RFBR, Project No. 01-07-90210.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 497–504, 2004.
© Springer-Verlag Berlin Heidelberg 2004

as the function subject to estimation. This function represents a relative addition to the inverse value of the initial approximation $v_0(r)$ of the signal propagation velocity. Let us assume that values γ_i $(i = 1, 2, \ldots, n)$ are the observed data points and that they can be represented in as linear functionals of an unknown function:

$$\gamma_i = \iint_{\Omega} G_i(r) m(r)\, dr, \tag{1}$$

where $G_i(r)$ are the kernels, in the form that is determined by the initial data and $m(r)$ is an unknown discrepancy. The starting approximation $v_0(r)$ of the velocity is assumed to be "close enough" to the real distribution $v(r)$. For a problem posed in this way the solution process involves minimization of the discrepancy $m(r)$. In other words, among all solutions satisfying (1) one chooses a function, which minimizes some functional, and in the simplest case, such a functional can be defined as

$$L(m) = \iint_{\Omega} |m(r)|^2\, dr. \tag{2}$$

Thus, the problem of reconstructing the velocity structure becomes a problem of finding function $\hat{m}(r)$ satisfying (1) and minimizing the functional $L(m)$.

In [1,2] Backus and Gilbert first introduced a technique that can be used to solve problems of this type. Their approach was extended to the case of a system of inequalities by Aki and Richards in [3]. Later, in [4,5] these solution techniques have been extended to match the current problem. Finally, in [7,8,9] the initial discussion of parallelization of the proposed approach was presented. Unfortunately, the results presented there leave a few open issues. While [7,8,9] contained some discussion of the efficiency of parallelization they were primarily focused on the development of the parallel algorithm. Furthermore, the results presented there were obtained on a rather exotic parallel computer (an MVS-100) and using a proprietary parallelizing environment developed for that machine. Finally, paper [8] was published in Russian, thus reducing its accessibility to researchers around the world.

The aim of this note is an initial attempt at remedying these problems. We proceed as follows. In the next section we briefly summarize the proposed solution. In Sections 3 and 4 we describe the parallelization of the proposed algorithm as well as the details of the experimental setup. Finally, in Section 5 we discuss results of our experiments on an IBM SP parallel computer.

2 Solution Algorithm

In [6] Kryazhimskij and Maksimov introduced an algorithm for finding a minimal (in some sense) solution of a system of linear inequalities in Hilbert space. This algorithm was then adapted to solve the problem described above. Let us now briefly describe its main points (for more details readers should consult [5]).

Input Data:
Domain $\Omega \subset \mathbf{R}^2$; coordinates of sources and receivers; time measurements t_i, $i = 1, \ldots, n$; errors $\varepsilon^{(1)}$, $\varepsilon^{(2)}$ of measurements of travel times; starting velocity distribution $v_0(r)$, $r \in \Omega$.

Initial Step:
From the starting velocity $v_0(r)$, determine wave paths l_{0i}, and travel times t_{0i}, $i = 1, \ldots, n$; find time discrepancies $\delta t_i^0 = t_i - t_{0i}$, $i = 1, \ldots, n$; from rays l_{0i}, build data kernels G_i^0, $i = 1, \ldots, n$; apply the algorithm developed in [5] to solve the problem of minimization of functional (2) under constraints

$$\delta t_i^0 - \varepsilon^{(2)} \leq \iint\limits_{\Omega} G_i^0(r) m(r)\, dr \leq \delta t_i^0 + \varepsilon^{(1)}, \quad i \in [1:n].$$

Compute M_0 steps of this algorithm, i. e. perform M_0 *inner iterations,* assuming $\nu_j = 0$, $j = 0, \ldots, M_0 - 1$; this results in the distribution of the discrepancy $m^1(r)$, $r \in \Omega$; from the discrepancy, find the velocity distribution $v^1(r)$ using the formula

$$m^1(r) = \frac{1/v^1(r) - 1/v_0(r)}{1/v^1(r)}, \quad r \in \Omega.$$

k-th Step:
(outer iteration) ($k = 1, \ldots, K$): in the ($(k-1)$-st) step, the velocity distribution $v^k(r)$, $r \in \Omega$ is found; it is utilized as the starting distribution for the current step: $v_0(r) = v^k(r)$, $r \in \Omega$. For this distribution, perform ray tracing, i. e. find shape of rays l_i^k, $i = 1, \ldots, n$, which are then treated as the starting approximation: $l_{0i} = l_i^k$, $i = 1, \ldots, n$; tracing may be performed, for instance, by integrating eikonal equations [10]

$$\frac{dx}{ds} = vp_1, \quad \frac{dy}{ds} = vp_2, \quad \frac{dp_1}{ds} = -v^{-2}\frac{\partial v}{\partial x}, \quad \frac{dp_2}{ds} = -v^{-2}\frac{\partial v}{\partial y}$$

using the Euler's method.
From the velocity distribution $v_0(r)$ and rays l_{0i}, find travel times t_{0i}, $i = 1, \ldots, n$ and time discrepancies $\delta t_i^k = t_i - t_{0i}$.
From rays l_{0i}, build data kernels G_i^k, $i = 1, \ldots, n$. Solve the problem of minimization of the functional (2) under constraints

$$\delta t_i^k - \varepsilon^{(2)} \leq \iint\limits_{\Omega} G_i^k(r) m(r)\, dr \leq \delta t_i^k + \varepsilon^{(1)}, \quad i \in [1:n].$$

To solve this step apply, again, the algorithm proposed in [5].
Compute M_k inner iterations for $\nu_j = 0$, $j = 0, \ldots, M_k - 1$; obtain the distribution $m^{k+1}(r)$, $r \in \Omega$; apply it to find the distribution $v^{k+1}(r)$, $r \in \Omega$.

Final Step:
As the final velocity distribution assign: $v(r) = v^{K+1}(r)$, $r \in \Omega$.

Results of numerical experiments reported in [7,8,9] show that satisfactory precision may be reached when performing two outer iterations, consisting of 3–5 inner iterations each. So, in total, algorithm consists of 6–10 inner iterations and one ray tracing.

3 Parallelization of the Solution Method

When the numbers of rays and cells of the discretization is large enough, the most time consuming computation is the ray tracing between the outer iterations. Naturally, most of memory is allocated for matrix *A,* which approximates data kernels G_i for all rays. Observe that solving the direct problem (tracing) is performed independently for each ray. Therefore, the input to the ray-tracing procedure is the (found) velocity distribution determined by vectors v_0 and z; while its output is a row of matrix *A* and a travel time t_0 corresponding to the given ray. This being the case we have selected the "master-slave" approach to parallelization. In particular, the *master process* is responsible for solving the inverse problem (i. e. processing inner iterations), distributing rays to worker processes, collecting results calculated by them and producing the final output. *Worker processes* trace rays, form matrix *A,* find new travel times t_0 for each ray and send them back to the master process.

Time spent by a given worker tracing ray l_i depends on the ray's length, which is not known a'priori. To ensure that rays are evenly distributed among the processes, the distance between the source and the receiver is used to estimate their lengths as follows:

$$|l_i| \approx d_i = \sqrt{(x_i^r - x_i^s)^2 + (y_i^r - y_i^s)^2}.$$

Here, (x_i^s, y_i^s), (x_i^r, y_i^r) are coordinates of the signal source and the receiver for the i-th ray. Total length of rays is approximated by the value

$$\widetilde{d} = \sum_{i=1}^{n} d_i.$$

Let P be the number of worker processes. Then the j-th worker is dedicated to the task of tracing rays numbered $i_{j-1}, \ldots, i_j$, $(j = 1, \ldots, P)$, $i_0 = 1$, $i_P = n$. Number i_j $(j = 1, \ldots, P-1)$ is determined by the relationships:

$$\sum_{i=1}^{i_p - 1} d_i < \frac{j}{N_w} \widetilde{d} \leq \sum_{i=1}^{i_p} d_i.$$

However, this approach has some potential disadvantages. First, data describing the wave velocity distribution is send to all processors. Since each worker does not need to know the complete velocity distribution, some of the transmitted data is not necessary. However, since the problem of extracting data actually required by a given processor is relatively complicated and time consuming we

found it more efficient to send it all and leave the extraction process to each individual processor. In this way all processors can extract pertinent velocity information in parallel. Second, computation time for different rays, even with equal length, may differ considerably. This is a result of inhomogeneities in the structure of the region under consideration. More complex areas of the region will require more computational effort resulting in a load imbalance.

4 Experimental Setup

The experiments were performed on the IBM SP RS/6000 computer (Seaborg) located at the National Energy Research Scientific Computing Center. While the Seaborg consists of 6087 processors we have utilized only one POWER3 SMP wide node consisting of 16 POWER3+ processors running at 375 MHz. Each processor has 32 Kbytes of level 1 instruction cache and 64 Kbytes of Level 1 data cache as well as 8 Mbytes of level 2 cache and is capable of delivering 1.5 Gflops of peak performance.

The code was written in C and parallelized using calls to the MPI library routines. It was compiled with the most aggressive optimization turned on. Experiments have been run in the batch mode assuring that the requested number of processors is dedicated to the job by the runtime environment and therefore no other jobs interact with the executing program. We have run multiple (at least 4) experiments for each data set and report the best time obtained in each case (we have used the MPI-based timer).

Data used in the experiments was prepared in a way analogous to that described in [5,7,8]. The square domain has the background velocity of 3.0 and contains both low and high velocity inhomogeneiety spots of irregular shapes with velocities of 2.5 and 3.7 respectively. Sources are uniformly distributed along the left and bottom sides of the square, while the receivers are located along the right and top sides.

We have selected such an allocation of sources and receivers that the rays cover the domain in a dense and uniform way. Thus, the number of rays was generally equal to $n = (2n_s)^2$, where n_s is the number of sources or receivers located along each side of the square. We did not consider very large sets of rays in order to keep the number of rays "realistic."

5 Experimental Results

We have experimented with $n = 100, 144, \ldots, 400$ rays and a spatial discretization consisting of 20 trough 100 cells in each dimension of the model domain (in other words, when we specify below that a given experiment has been run for 20 cells, this means that the domain has been discretized with 20 × 20 = 400 cells). In Figure 1, we present execution times for $n = 100$ rays and 20, 40, 60, 80, 100 cells, for $P = 1, 2, \ldots, 15$ worker processes.

As expected, as the number of cells increases so does the gain from using multiple processors. More interesting is the behavior of the code for 40 and

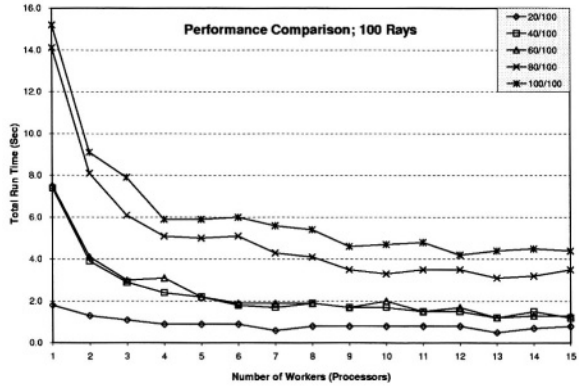


Fig. 1. Performance comparison, $n = 100$ rays, varying the number of cells and the number of processors.

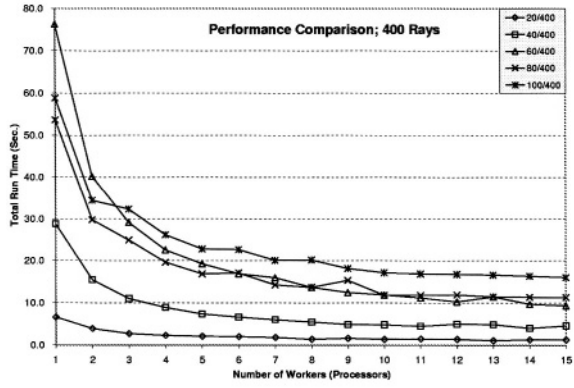


Fig. 2. Performance comparison, $n = 400$ rays, varying the number of cells and the number of processors.

60 cells, where the execution times overlap. As a comparison, on Figure 2, we depict the performance of the code for 400 rays and 20–100 cells and 1–15 worker processes.

Surprising is the performance of the code for 60 cells. It is clearly much slower than that for 80 and 100 cells for both 1 and 2 processors. A similar pattern of behavior was observed for 144, 196, 256 and 324 rays. We plan to investigate this phenomenon in the near future.

Finally, we present the speedup obtained for 100, 144, 196, 256, 324 and 400 rays and 100 cells (the largest considered case for each number of rays). The results are presented in figure 3.

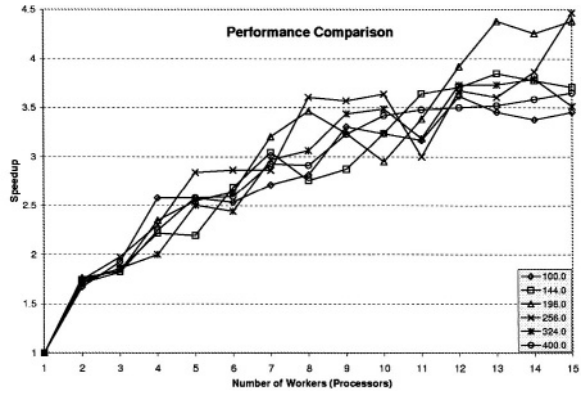


Fig. 3. Speedup comparison, 100 cells, 1–15 workers and varying number of rays.

The results are somewhat disappointing. As the number of rays increases the parallel efficiency does not. The best results are obtained for 256 rays, where an efficiency of approximately 30% is reached.

At the same it can be observed that part of the "problem" is related to the fact that the single-processor realization of the algorithm is very efficient. When single processor execution times in the case of 100 cells and 100 and 400 rays are compared it can be observed that the time in the latter case is 3.87 times longer. This indicates that while the code is expected to perform 4 times more work, this is not exactly the case. The solution of the larger case it is found relatively faster. This pattern is repeated throughout the calculations. The single processor execution time ratios of 144/100 rays, 196/100 Rays etc. are: 1.34, 1.96, 2.46, 3.14 and 3.87.

6 Concluding Remarks

In this note we addressed the parallelization of an algorithm for calculating the velocity structure of seismic waves on the basis of measurements made by the "source–receiver" pairs. This problem is reduced to the problem of minimizing a system of inequalities and solved through application of Backus–Gilbert approach combined with the constraint aggregation method. This problem is parallelized using MPI library functions and experimented with on an IBM SP RS/6000 parallel computer.

In our experiments we have achieved an efficiency of 30% for 15 workers. At the same time we have found a few peculiarities in the behavior of our implementation. First, the unexpected time increase for the case of 60 cells per ray. Second, the fact that as the number of rays increases we do not observe further gains in parallel efficiency. Related to this is the fact that as the number of rays increases the single processor execution time becomes relatively faster. We plan to investigate these facts further and report on our findings in the near future.

References

1. Backus, G., Gilbert, F.: Numerical Applications of a Formalism for Geophysical Inverse Problems. Geophysical Journal of the Royal Astronomical Society, **13** (1967) 247–276
2. Backus, G., Gilbert, F.: The Resolving Power of Gross Earth Data. Geophysical Journal of the Royal Astronomical Society, **16** (1968) 169–205
3. Aki, K., Richards, P.G.: Quantitative Seismology: Theory and Methods. Vol. 2. Freeman and Company, San Francisco (1984)
4. Ditmar, P.G., Yanovskaya, T.B.: Extention of the Backus—Gilbert Method for estimation of lateral variations of surface wave velocity. Izvestia AN SSSR, Fizika Zemli, **6** Moscow (1987) 30–40 (in Russian)
5. Digas, B.V., Maksimov, V.I., Lander, A.V., Bukchin, B.G.: On an Algorithm for Solving the Inverse Problem of Ray Seismics, *Computational Seismology,* **30** (1998) 207–224 (in Russian)
6. Kryazhimskii, A.V., Maksimov, V.I. An Iterative Procedure for Solving a Control Problem with Phase Constraints. Computational Mathematics and Mathematical Physics, Vol. 38, No. 9 (1998) 1423–1428
7. Digas, B., Maksimov, V., Melnikova, L.: Parallel Algorithms for an Inverse Problem of Lithosphere Structure Reconstruction. Notes Numer. Fluid Mech., Vol. 73, Vieweg (2000) 357–362
8. Bukchin, B., Lander, A., Digas, B., Maksimov, I., Melnikova, L.: Parallel Algorithms in Inverse Problem of Ray Seismics. In: Algorithms and software for parallel computations, Ekaterinburg, Vol. 3 (1999) 3–15 (in Russian)
9. Paprzycki, M., Rozenberg, V.,Digas, B.: Medium Structure Modeling on Parallel Computers. In: Maksimov, V., Ermoliev, Yu., Linnerooth-Bayer, J. (eds.): Risk Management: Modeling and Computer Applications, IIASA Interim Report, Laxenburg, Austria (2001) 71–82
10. Psencik, I.: Seismic Ray Method for Inhomogeneous Isotropic and Anisotropic Media. Second Workshop on Tree-dimensional Modelling of Seismic Waves Generation, Propagation and their Inversion, Trieste, Italy (1994)

Fish Schools: PDES Simulation and Real Time 3D Animation*

Remo Suppi, Daniel Fernández, and Emilio Luque

Dept. of Computer Science, Universitat Autònoma de Barcelona
08193, Bellaterra, Spain
{Remo.Suppi,Emilio.Luque}@uab.es, Daniel.Fernandez@campus.uab.es

Abstract. The present work outlines the results of the Parallel Discrete Event Simulation (PDES) and advanced animation techniques for solving an individual based model: Fish Schools. On the one hand, this type of model cannot be solved through analytical methods, thus simulation techniques are necessary. Due to the high computing capacity necessary for solving middle-high size problems, distributed simulation is the most adequate technique. On the other hand, high quality animation from high volume data can only be developed using advanced animation techniques. In the present paper, we present a software infrastructure for high quality animation from PDES simulation data as a useful and low cost tool for the study and analysis of biologically inspired models.

1 Introduction

In the natural world, we find a great number of species organized in groups for various given reasons; generally however, the most important of these is to avoid depredation. To study and analyse problems such as survival, reproduction and migration, biologists have developed models to analyse the behaviour and the survival of a particular species.

One of the species groups that is considered highly important, both for its economic importance and due to its overexploitation, is that of fish colonies (tuna, cod, sardine, etc.), termed fish schools.

Two important aspects in the behaviour simulation of a fish school should be considered when computers are used to solve problems generated by such simulation. These are model and simulation techniques and visualization strategies.

Individual-oriented models have gained acceptance with scientists because they provide better results than do general models or those guided to groups. The main simulation drawback to such models is the high computing capacity necessary to solve the problems of a considerable quantity of individuals. This type of model chooses the individual as the basic element of the system. The ecosystem is described by dynamic and static individual properties. The behaviour of an individual can differ from the behaviour of other individuals of

* This work was supported by the MCyT, under contract TIC2001-2592 and partially supported by the Generalitat de Catalunya-GRC 2001SGR-00218

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 505–512, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the same or other species. This type of model cannot be solved in an analytical form and it is therefore necessary to use simulation techniques in obtaining the ecosystem's dynamical behaviour.

For complex systems (thousands of individuals), there is the need to use advanced simulation techniques and parallel-distributed computing systems to give an efficient response to such problems. PDES (Parallel and Distributed Event Simulation) is a useful tool (and indispensable in some instances) for providing response to complex problems within an acceptable time.

The second problem to solve is the animation of large quantities of independent individuals in a dynamic environment. It is important to consider the animation problem, as we are now accustomed to movie animations (Hollywood provides good examples with movies such as AntZ, Monster S.A., Shrek or Ice Age) which generally involve the use of special effects and require many computing hours. In our case, we will consider animations of hundreds/thousands of individuals in real time using advanced animation techniques in order to represent high-quality 3D animations.

This paper demonstrates the use of PDES and advanced animation techniques in solving a type of individual-oriented model: Fish Schools. The next section is a summary of the characteristics of the individual-oriented model. Section 3 shows the Fish Schools' simulation and animation infrastructure. Section 4 presents the experimental framework and sections 5 and 6 provide the conclusions and references, respectively.

2 Individual-Oriented Models

There are considerable references to individual-oriented models (IoM) in the literature [1,2,4]. The model definition is based on recognizing each individual as autonomous and that acts according to a set of biological rules.

One of the most representative applications of IoM is used to describe the movement of given species (schools, flocks, herds, etc) [1,10,12]. The IoM utilization allows us to determine the movement of a species group by using the movement of each member. The Fish Schools is an IoM application for the movement of fish species. [3,1,5]. From observation, it was discovered that fish can describe very complex figures in their movement, but that these figures are governed by three basic postulates from the point of view of the individual:

1. To avoid collisions
2. Speed coupling
3. To obtain a position in the centre of the group

These rules express both the individual's need for survival and its instinct for protection (the need to escape from predators). Each fish in the model is represented as a point in a three-dimensional space with an associated speed. And each fish changes position and speed simultaneously after a certain period Δt. The actions that the model describes for each fish are:

1. Each fish chooses as many as X neighbour fish ($X = 4$ seems sufficient for most schools), which will be those nearest and in the direct line of vision.
2. Each fish reacts in accordance with the direction and distance of each neighbour. Three influence radios and three possible reactions are established. The final reaction will be the average of the reactions experimented on each neighbour.
   a) If the neighbour is found within the smaller radio, the fish will carry out an "opposed to address" movement -repulsion action- (to avoid collisions).
   b) If the neighbour is within a second radio of influence, the fish will adopt the same direction as the neighbour.
   c) If the neighbour is within a third radio, the fish will move towards it.
3. Each fish calculates its new position according to the new direction.

This generates a very simple model, but one that allows very complex behaviour to be described (an implementation with applets in 2D can be found in [1]). As a counterpart, very high computing power is necessary, since the complexity algorithm is of $O(N^2)$, where N is the number of fish (each fish attempts to find the neighbour fish by inspecting all other fish in the school). A detailed model description is show in [9].

3 Distributed Simulation and Animation Infrastructure

The Fish Schools application has been split into two computing parts (Fig. 1):

1. **Simulation cluster (based on Linux+PVM):** where the distributed simulator is executed and the frame animation data is generated. In this cluster, there is a machine that reorganizes the data and prepares them in order to be animated (Data Collector). This data collector can be connected on line with the animation application through *Tcp/Ip* as well as generating trace files to be visualized in a subsequent animation.
2. **Animation tool:** This tool animates the data sent by the simulator and controls all the animation parameters such as cameras, lights, scenes, point of view, etc.

The user interaction is made through a simple GUI based on a web form where the user indicates the parameters and the characteristics of the simulation. The web server, which is executed in the data collector, generates the initial conditions and runs the distributed simulator that will send data through a Fast Ethernet to the animation tool, or else will generate trace files for post-animation. Figure 1 shows the whole system and interaction.

3.1 PDES Simulation

As a model of the parallel discrete event simulation (PDES), a set of logical processes (LP) managing a distributed event lists was considered. These processes

interact exclusively by exchanging time-stamped messages. The PDES mechanisms can be divided into two categories: *conservative* and *optimistic.* Conservative simulation approaches use synchronization to avoid causality errors [8]. In these algorithms, the events are processed when it is certain that the execution order is correct. On the other hand, in optimistic algorithms, each LP processes the events as soon as they are available and this execution, in some instances, can mean causality errors. Nevertheless, the algorithm has detection mechanisms to avoid these errors and to recover the causality [8,10,6].

The fish-school simulator was built on the base of a PDES simulation kernel developed at the UAB (written in C++). This kernel is designed to be executed in Unix stations and PVM (a PVM daemon is included in each computer in order to communicate the different LPs -see Fig. 1-). The union of the simulation kernel and the model classes is accomplished through the inheritance of the kernel classes and virtual functions implementation. Based on the UAB kernel, two simulator versions were developed: **sequential** (serial) and **distributed.** The distributed version will be used to make performance analysis by using different simulation PDES algorithms. The sequential version will allow speedup and performance analysis to be undertaken with respect to the distributed version. [6,9]

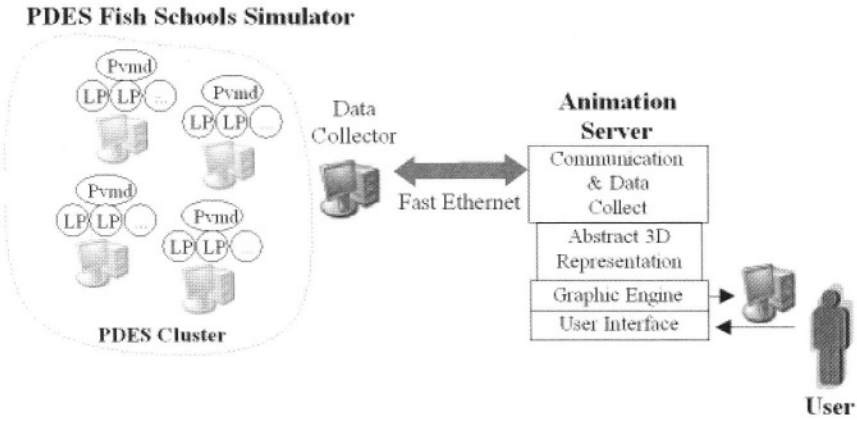


Fig. 1. PDES Simulator and Animation Architecture

3.2 Animation Tool

The animation tool has been designed as a multithreading application communicated by sockets with a PDES simulator process (data collector), in order to obtain the real-time data of the fish movement. OpenGL library was used so as to obtain a high-quality animation in 3D and with the possibility that the user could have interaction with the animation engine changing points of view, light effects, shades, textures etc. [3,7,11,12,13,14].

The application of animation is a critical program whose structure has been carefully designed to represent 3D objects (fish) in a virtual environment in real time. The position and the speed of the fish are produced by the PDES simulator and are send to the animation server (or to store this data in trace files). The animation tool is organized in four modules:

1. **User Interface:** this is a thread that attends to the user during the animation.
2. **Communication and Data Collect:** this is the thread responsible for establishing communication with the simulator to obtain the data and to create and control the hash buffers used to accomplish the animation.
3. **Abstract 3D Representation:** this module manages the virtual world (three-dimensional space and objects) using 3D models of the objects
4. **Graphic Engine:** the main functions of this module are: initialisation of the graphics environment and 3D models subsystems, setup and control of the cameras, lights, textures, scene, etc. and repainting of the screen.

Figures 2 and 3 show two types of animation frames obtained with the PDES simulator for 300 fish.

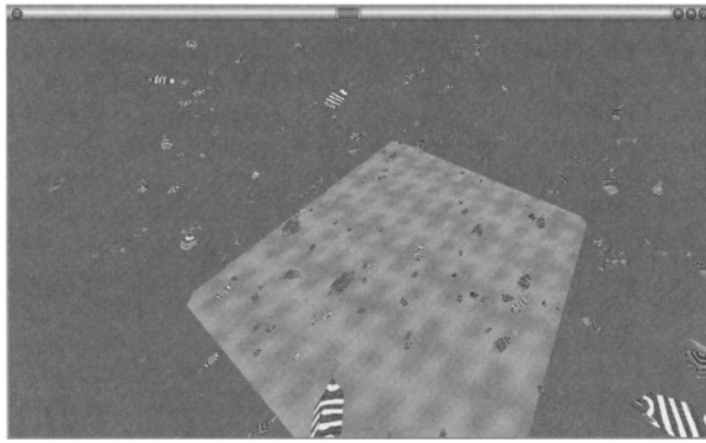


Fig. 2. A Frame of 3-D Fish Animation (300 fish) with shadows and lights

4 Experimental Studies

The experimental framework was developed on a cluster of machines executing Linux SuSE 6.4 and interconnected by a Fast Ethernet. The tools used for the development of the PDES simulator and animation application were: Pvm 3.4, Gcc 2.95, OpenGL 1.2.1, GLX, STL, and Pthreads.

The first step in the development process was the design and development of the sequential-simulator version. This version was analysed with different

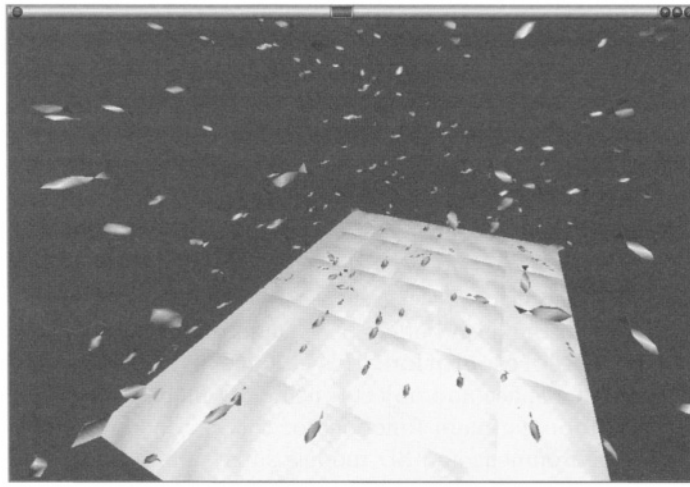


Fig. 3. A frame of 3D fish animation (no textured model for speed optimisation)

simulation parameter values: territory sizes, number of individuals, velocity and neighbour number. As a reference measure, we chose the frame generation time (new position and speed for each individual) for colonies of 100 to 1600 fish. Figure 4 shows that real time animations in the sequential simulator (all LP in the same processor, using an only one event list and executed in a round-robin queue) requires complex treatment, since with groups of 200 individuals, 1 second per frame is needed.

The second step was the utilization of distributed simulation using conservative distributed-simulation algorithms. The optimistic PDES simulation was discarded due to problems with synchronization events in the limits of the fish-school model distribution. A complete description of this problem can be found in [9]. From figure 4, the following conclusions can be extracted:

1. Results are very good and the model scales well: as a rule, an increase in processor number facilitates a reduction in frame time to values below 1 second.
2. Frame generation time cannot be reduced without the limit increasing the processor number. For visualizations in real time, approximately 4 fps for 400 fish and 4 processors are obtained.
3. The model to be simulated must be analysed carefully, there are situations in which adding processors does not result in any benefit. Such is the case in the current model: there is no need to use 8 processors to simulate less than 800 individuals. The model distribution generates high communication granularity for these cases.

This figure confirms that at least 100 fish per processor are required in order to obtain acceptable speedups.

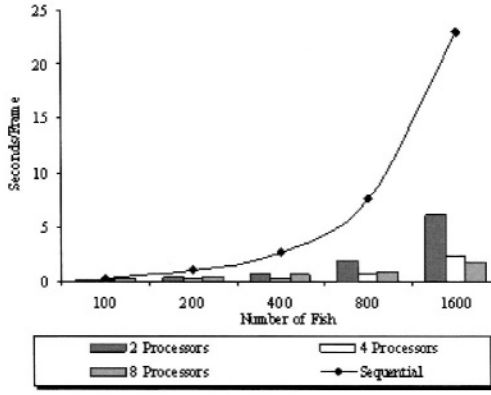


Fig. 4. Sequential and conservative fish school simulation

5 Conclusions

Ecological systems simulation and animation is a field that requires considerable computing power for realistic models. Parallelism is a highly useful tool in solving this type of problem simulation, and advanced techniques for real-time animation are necessary. The present study shows good results for IoM models by using PDES and illustrates the viability of on-line animation using OpenGL. The operative prototypes were built using a PDES kernel developed at the UAB and are executed on a low-cost distributed architecture and servers based on Linux.

Due to the model's characteristics, the IoM Fish Schools model was developed using conservative PDES algorithms. The animation tool was designed and implemented using a separate server, communicated with the simulator by a Fast Ethernet. The main conclusions that can be extracted are:

1. Model scalability is possible and acceptable. The obtained results show a better speedup with respect to the existing parallel versions. [5]
2. Performance is good with respect to large data animations, but there is a limit essentially imposed both by the communications model and the architecture. A reduction in this time would mean changing the communications model (for example using MPI) and modifying the network communications technology (Gigabit Ethernet).

Future work is guided towards:

1. The need to include an increase in individual and processor numbers within the study, in order to verify model scalability.
2. Simulation methods will have to include optimistic protocols such as the STW [8] that allow improvements in results with respect to conventional TW protocols.

3. The IoM implementation code must be improved in two aspects: *1)* to reduce the neighbours localization using advanced searching list algorithms (necessary in high density fish simulation) *2)* to include biological characteristics, in order to make it more realistic (this would involve, for example: speed coupling, behaviour patterns, inertial behaviours, dead angles, etc.)
4. Visualization of the simulation results must be enhanced in order to allow web-based interaction and visualisation through the use of (for example) VRLM. [13,14]

References

1. ECOTOOLS: High level tools for modelling and simulation of individual-oriented ecological models. (1999). *http://www.offis.uni-oldenburg.de/projekte/ecotools*
2. Fishwick, P., Sanderson, J.G., Wolf, W.: A multimodeling basis for across-trophic-level ecosystem modelling. Trans. SCS. **15(2)** (1998) 76–89
3. Huth, A., Wissel, C.: The simulation of movement of fish schools. Journal of Theoretical Biology. **156** (1992) 365–385
4. Kreft, J. Booth, G, Wimpenny, W.: BacSim, a simulator for individual-based modelling of bacterial colony growth. Microbiology. **144** (1998) 3275–3287
5. Lorek, H, Sonnenschein, M.: Using parallel computers to simulate individual oriented models: a case study. European Simulation Multiconference (ESM). (1995) 526–531
6. Munt, P.: Simulació distribuida en PVM: implementació dels algorismes TW i Switch Time Warp. Graduate Thesis (in Catalan). Universitat Autònoma de Barcelona. Spain. (1999)
7. Proctor, G., Winter, C.: Information flocking, data visualisation in Virtual Worlds using emergent behaviours. Virtual Worlds. **1434** Springer-Verlag. (1998) 168–176
8. Suppi, R., Cores, F, Luque, E.: Improving optimistic PDES in PVM environments. Lecture Notes in Computer Science. Springer-Verlag. **1908** (2000). 304–312
9. Suppi, R., Munt, P., Luque, E.: Using PDES to simulate individual-oriented models in ecology: A case study. Lecture Notes in Computer Science. **2329** (2002). 107–116
10. Sloot, P., Kaandorp, J., Hoekstra, A., Overeinder, B.: Distributed Cellular Automata: Large Scale Simulation of Natural Phenomena. Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences. *ISBN: 0-471-35352-3* (2001), 1–46
11. Neider, J., Davis, T., Woo, M.: The OpenGL Programming Guide (The Red Book). Addison-Wesley. Available as Freeware E-Book at *http://fly.cc.fer.hr/~unreal/theredbook/*
12. Reynolds, C.: Flocks, Herds and Schools: A Distributed Behavioral Model. Computer Graphics. **21(4)** (1987) 25–34
13. Carlson, D., Hodgins, J.: Simulation levels of detail for real-time animation. Graphics Interface. (1997) 1–8
14. Multon, F., France, L., Cani-Gascuel, M-P., Debunne, G.: Computer animation of human walking: a survey. The Journal of Visualization and Computer Animation. **10(1)** (1999) 39–54

Consuming Environment with Transportation Network Modelled Using Graph of Cellular Automata

Pawel Topa and Witold Dzwinel

Institute of Computer Sciences, AGH University of Science and Technology,
al. Mickiewicza 30, 30-059 Cracow, Poland
{topa,dzwinel}@agh.edu.pl

Abstract. We discuss the applications of new methods, derived from the cellular automata paradigm, for simulating dynamic feedback systems consisting of transportation network and consuming environment. The general framework of the model is introduced. We describe the application of the methods for modelling the anastomosing river system and root system of plants. The results of simulations are depicted and commented. We show that the methods mimicks well the realistic networking systems.

1 Introduction

Transportation network and consuming environment represent the dynamical system with high level of synergy. The role of transportation network consists in supplying the consuming environment with a certain kind of resources such as nutrients. The environment feedbacks, adapting the network structure to its current requirements.

Resources supplied by the network are distributed along all branches to the surrounding environment. The presence of nutrients results in a growth of the environment, which can be manifested in various ways, usually as a change of a certain set of variables (e.g. altitude of the terrain, amount of some kind of resources etc.). These dynamical changes are accompanied by the the development of the transportation networks, that are driven by many factors. These factors reflect the character of both the network system and the environment. They stimulate the creation of new branches, which supply desirable resources to the starving areas or inhabit the development of existing channels.

Both the phenomenon of nutrients spreading and growth of the environment can be modelled with Cellular Automata (CA) [1], [2]. A regular mesh of cells represents area in which the resources are distributed. The state of each cell is described by at least two parameters: 1) the saturation level of nutrients, and 2) the values reflecting the growth of the modelled environment. Spreading of nutrients is mimicked by using pre-defined rules of local interactions while the growth of the environment is modelled by applying rules of internal transformations. It means that a new state of a cell depends only on its previous state.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 513–520, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The transportation networks can be modelled by a graph with weighting edges. The nodes of such the graph represent the points of merging or forking of the streams of the transported resources. The edges stand for the transportation channels between the nodes. The weights attributed to each of the edges correspond to one or more features of transportation channel, e.g., its throughput or current flow rate.

The transportation network growths creating new branches. The formation of a new channel is governed by a certain number of factors characteristic for modelled phenomena. We must distinguish the factors that a) lead to forking of old branches and b) influence on the route of the newly created branch. All these factors and rules must be introduced to the general framework of the model.

In order to combine the model of nutrients spreading with graph representation of transportation network, we propose a new approach called the Graph of Cellular Automata (GCA). The graph is created by linking some of the cells from the regular mesh with edges (see Figure 1). This idea is similar to graph–based definitions of Cellular Automata (see [3]). Assuming that the Cellular Automata is a "regular" graph with the state automata as its nodes, the graph we create, is a subgraph of the Cellular Automata system.

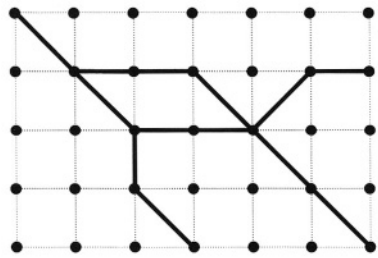


Fig. 1. Graph of Cellular Automata

The cells that belong to the graph are described by a set of parameters. These cells are also the sources, from which the nutrients are spread out to the neighbouring cells of the Cellular Automata environment.

The following section presents the formal definition of the Graph of Cellular Automata method. Next we demonstrate how the GCA model can be adapted for modelling two realistic phenomena: the anastomosing rivers system and the root system. The concluding remarks are collected at the end.

2 The Graph of Cellular Automata

The Graph of Cellular Automata is formerly defined as:
$\boldsymbol{GCA} = \langle Z^n, X_K, A, G_{CA}, Q, \sigma_m, \sigma_t \rangle$,

– Z^n is the set of cells with integer coordinates in 2 or 3-D Euclidean space;

- $A \subset Z^n$ is the nonempty set of cells, which are the root nodes for the graph;
- $X_K \subset Z^n$ defines the neighbourhood for each cell in the regular mesh (we assume the Moore neighbourhood is used):
  $X_K(x_{i,j})=\{x_{i-1,j-1}, x_{i-1,j}, x_{i-1,j+1}, x_{i,j+1}, x_{i+1,j+1}, x_{i+1,j}, x_{i+1,j-1}, x_{i,j-1}\}$
- G_{CA} is a graph (V, E) where $V \subset \mathbb{Z}^n$ is a set of vertices and E is a set of edges $E \subset Z^n \times Z^n$;
- Q is a set of states of elementary automaton: $Q = Q_r \times Q_t \times Q_e$
  - Q_r — corresponds to the amount of resources (nutrients) in the cell,
  - Q_t — corresponds to the parameters of transportation channel,
  - Q_e — avalue describing the growth of the environment in the cell. It can correspond to one or more quantities.
- $\sigma_m : q_m^n \to q_{re}^{n+1}$, $q_m \in Q_r \times Q_e$ is a state transition function for all the cells in Z^n
- $\sigma_t : q_t^n \to q_t^{n+1}$, $q_t \in Q_t$ is a state transition function for the cells in the set of nodes V.

The Cellular Automata cells evolve according to the rules of local and internal transformations described by σ_m. Nutrients are spread out from the cells that belong to the graph. The rule of nutrient spreading has to model the nutrients gradient forming nearby the transportation channel. The presence of the nutrients in the cell results in change of one or more parameters, which correspond to the growth of the environment (Q_e).

The graph is constructed by choosing some cells from the regular mesh and connecting them with the edges according to a predefined procedure. Depending on phenomena modelled the graph can be directed or undirected. The nodes of the graph remain in different states resulting from transition functions σ_m and σ_t. The changes in Q_t states model the processes occurring in the transportation channels, e.g., decreasing throughput, flow rate etc.

The general method for creating a new branch in GCA model consists of two steps:

1. Identifying the nodes of graph in which splitting occur. This can be indicated by a specific values or combination of values of variables describing channel parameters.
2. Building a new branch by choosing the cells iteratively and joining them to the current branch. This procedure implements local optimisation — calculating the path of the steepest descent.

The initial branches of the transportation network are calculated starting from the nodes defined in the A set. They are the entry points through which the resources are supplied to the transportation network. The flow rates in these nodes are initiated arbitrarily.

3 Application of the GCA Model

In this section we present the model of anastomosing river, which is a typical transportation system driven by a consuming environment. We show also that

just by "reversing" the rules we can adapt the model for simulaiting the growth of the root system.

3.1 The Model of Anastomosing River System

The term "anastomosing" corresponds to a network composed of forking and joining branches. The more detailed description of the factors which govern the evolution of the anastomosing river can be found in [4]. The GCA model, which we present in this paper, is a generalisation of the MANGraCA model [5], that has been developed for modelling the anastomosing river networks.

The existence of anastomosing river is a result of interactions between the river carrying nourishing resources (nutrients) and the environment that consumes nutrients converting them into a peat layer. The area of the river valley with growing layer of peat-bog is a typical consuming environment. The nutrients penetrate the soil surrounding the riverbeds and stimulate the vegetation of peat-forming plants. Gradient of nutrient saturation, which appears mainly as a result of suction of root system, decreases the rate of peat accumulation with the distance from the channel. The growing thickness of the peat layer is the principal growth factor of the environment.

Creation of the new channels are the results of jams occurring randomly in the channels. The route of the channel is determined by the local terrain topography. New channels usually merge with others creating a complex network composed of splitting and merging channels.

In the MANGraCA model the environment is defined as a regular mesh of Cellular Automata, The states of the cells are described by the three values which correspond to a) altitude of the terrain, b) thickness of the peat layer and c) the level of nutrient saturation. The CA rule for nutrients spreading is defined as follows:

$$if\ (r^t_{i,j} < max(X_K(r^t_{i,j}))\ then\ r^{t+1}_{i,j} = \delta * max(X_K(r^t_{i,j})) \tag{1}$$

where, $r^t_{i,j}$ is the amount of nutrients in cell $r_{i,j}$ in the timestep t. The value of δ is the gradient coefficient. The value of peat layer thickness is increased according to the current amount of nutrients.

The river network is represented by a directed graph of cellular automata. The nodes in the graph have two additional attributes that correspond to the throughput and current flow rate in this fragment of the riverbed. The nodes are also the sources of nutrients for the cells in the regular mesh.

The throughput values are decreased in each iteration. The rate of decrease is very slow, but in randomly chosen cells a considerable reduction of throughput may occur. This corresponds to the jam in a riverbed. If the flow rate exceeds the throughput value, the new channels must be created by using defined tracing procedure.

The tracing procedure must reflect the character of the network. In the model of anastomosing network it traces the path of the steepest descent, which starts from (or above) the point of jam. The procedure includes iteratively the cells with

the the fastest decrease of altitude (calculated as a sum of initial altitude and the peat thickness) to the newly created branch. The procedure stops if the cell that already belong to the graph is reached or the mesh border is encountered.

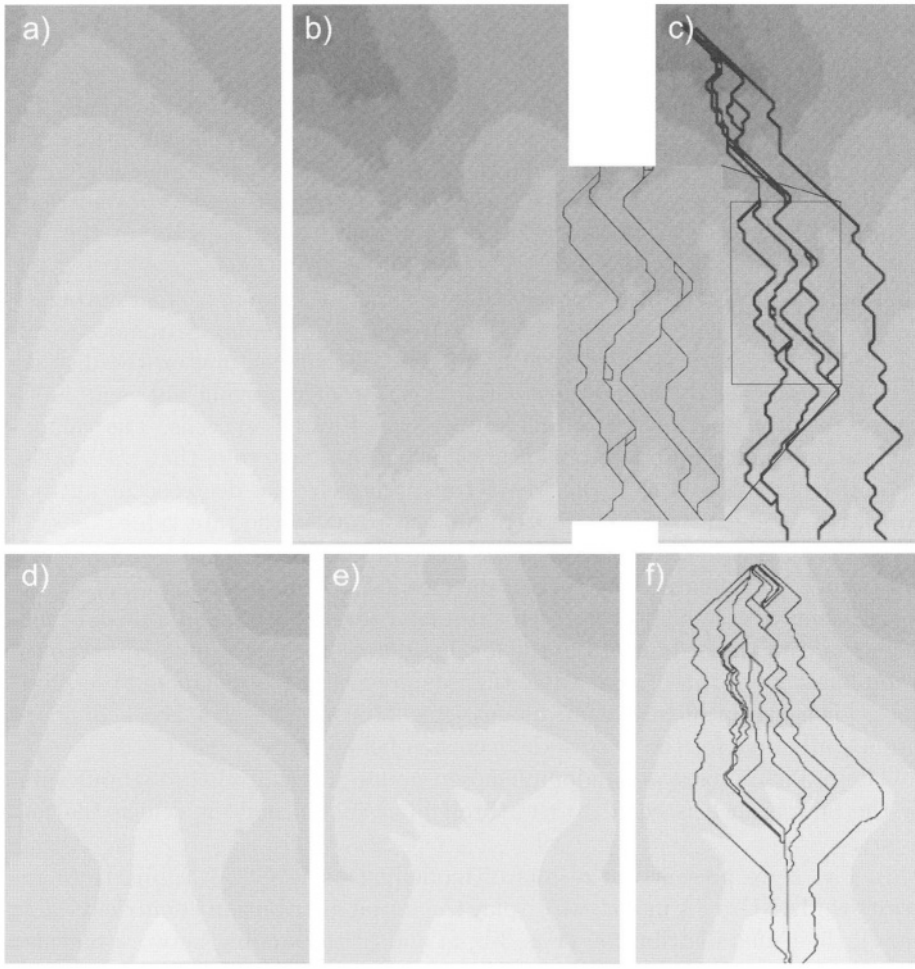


Fig. 2. Anastomosing networks (see text for details)

In Fig. 2 we present the results of simulations obtained by using MANGraCA model. Figures 2a and Figures 2d shows the initial shapes of the terrain — darker shades of grey relate to higher altitude. The final shape of terrain are depicted in Fig2b and Fig2e. Corresponding network patterns are presented in Fig2c and Fig2f. The model has been implemented using rectangular mesh. This results in 45 degree–anisotropy of the branches. To avoid this effect the hexagonal mesh or irregular mesh should be used.

The networks we have obtained have characteristic anastomosing pattern, i.e., splitting and joining branches. Magnification of the network in Fig. 2c shows their fractal structure.

3.2 The Models of Root System

The role of transportation network is not only suppling starving environment with nutrients. It can work in reverse direction as it, is in the root systems. The roots drain off organic substances and water from the soil, collecting them and transporting to the plant cells. The root system expands to supply the growing plant with adequate amount of resources. The decreasing level of nutrients saturation in the soil represents the environmental growth factor of the root system.

The root network is also represented by a directed graph of cellular automata. Contrary to the model of anastomosing river, the nutrients are removed from the system. This process occurs with different rates depending on the distance from the branches of the root network. The rate of removing nutrients is the fastest near the nodes of the graph and decreases in distant cells. The value of the rate parameter is updated in each cell using the same rule, that was applied for nutrient distribution in the MANGraCA model. The decrease in nutrient saturation are calculated in each iteration according to the rate value.

The roots create branching networks. The procedure responsible for creating new branch is calculating the path of the steepest ascent, directing them toward the regions with the highest nutrient saturation level. The throughput of branch decreases with the distance from the forking point. Each new node added to the graph has smaller throughput value than its predecessor. The tracing procedure stops when the local maximum of nutrient saturation level is reached or the throughput value for the next node decreases below zero.

The branches forks at randomly chosen nodes. The initial throughput value for the new branch is equal to the throughput in the node in which the fork occurs.

In Fig. 3 we present the results of modelling using GCA. Similarly to the results of MANGraCA model, we display the initial distribution of nutrients (Fig. 3 a, e), their final distribution (Fig. 3 b, f) and corresponding network patterns (Fig. 3 c, g). The frame with zoom–out of network pattern is also shown (Fig. 3d. Lighter shades of grey corresponds to higher value of the nutrients saturation.

The 45 degree–anisotropy is observed again due to the application of rectangular mesh. The initial distribution of the nutrients (Fig. 3 a)) forms a narrow rectangular area with maximum on the axis of symmetry. The network we obtain in this simulation has a very dense structure. It consist of several very long parallel branches, that has been formed at earlier steps of simulation, and large number of short branches, that were created in further. This pattern is the result of concentrating high saturation of nutrients on small area. Contrary, nutrients distribution in Fig. 3e is forming flat, wide and sloped area. The resulting network covers almost the whole area.

The networks we obtain has a semi-fractal structure, repeating their type of pattern in different scales as it is shown in the zoomed out fragment of network in Fig.3c.

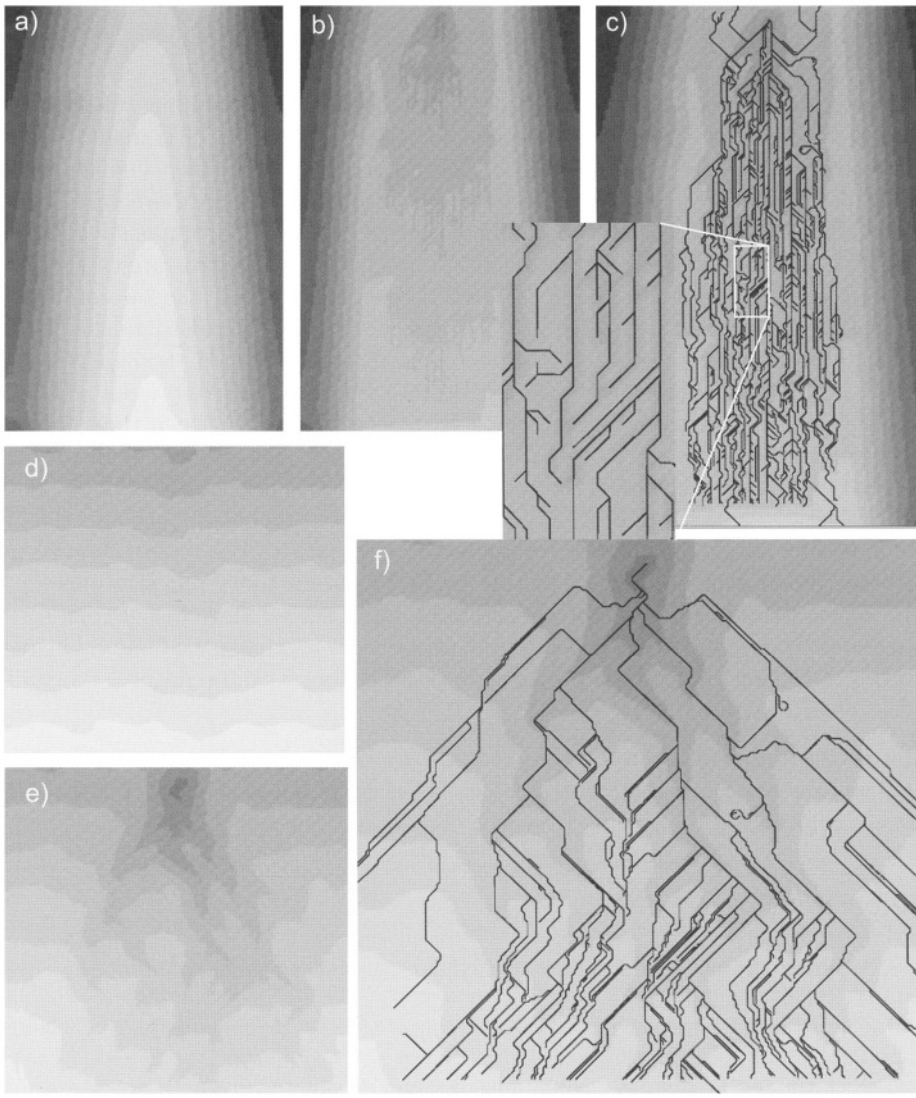


Fig. 3. 2D root structures (see text for details)

The model of root system is significantly simplified. We have concentrated on interaction between the network and the environment. At this moment any biological factors has not been taken into consideration. The process of collecting

and transmitting resources from to soil to the plant tissues has been limited only to reducing nutrient amount in the cells that belong to the graph. The root system develops in 3D space, therefore this model should be also reproduced in 3D.

4 Concluding Remarks

In this paper we have shown that the Graph of Cellular Automata (GCA) can be useful model for simulating dynamical systems consisting of transportation network immersed in a consuming (or producing) environment. The network fulfil the transportation functions supplying or draining off the resources from the environment. The network is a dynamic structure, which evolution is stimulated by internal factors or by surrounding environment. The resources are consumed or produced by the environment. For the systems modelled, the one or more quantities expressing the growth of the environment must be distinguished.

As we show in Section 3, the GCA can be applied for modelling transportation systems with a very diverse character. Despite the differences between anastomosing river system and root systems, the mechanisms that driven their evolution remain the same. We suppose that other systems consisting of transportation networks and consuming environment such as vascular systems, road and railway networks, WAN and MAN computer networks etc. can be also effectively simulated by using the method presented in this paper. We realise, however, that our model can be only a starting point to obtain more realistic patterns representing these complex, multiresolutional structures. Moreover, larger systems have to be simulated to obtain qualitative agreement (verified e.g. by using fractal dimension) with the network structure produced by the nature.

Acknowledgement. Authors are grateful to Dr Mariusz Paszkowski for his contribution to this paper. This project is partially supported by The Polish State Committee for Scientific Research (KBN) under grant 7T11C00521.

References

1. S. Wolfram, Computation theory of Cellular Automata, *Comm. Math. Phys.* 96 (1984) 15-57.
2. B. Chopard and M. Droz, *Cellular Automata Modelling of Physical Systems,* Cambridge Univ. Press, 1998.
3. C. Nichitiu and E. Remila, Simulations of graph automata, *In Th. Worsch and R. Vollmar, eds, MFCS'93 Satellite Workshop on CA,* Karlsruhe, 1998.
4. R. Gradziński *et al.,* Anastomosing System of Upper Narew River, NE Poland, *Annales Societatis Geologorum Poloniae,* 70:219-229, 200.
5. P. Topa and M. Paszkowski, Anastomosing Transportation Networks, *Lecture Notes in Computer Sciences,* Proceedings of PPAM'2001 Conference, 2002.

Parallelizing Flood Model for Linux Clusters with MPI

Viet D. Tran[1], Ladislav Hluchy[1], Dave Froehlich[2], and William Castaings[2]

[1] Institute of Informatics, Slovak Academy of Sciences
Dubravska cesta 9, 845 07 Bratislava, Slovakia
viet.ui@savba.sk
[2] 303 Frenchmans Bluff Drive, Cary, North Carolina 27513-5662, USA
Froehlich@pbworld.com
[3] LMC-IMAG, Domaine Universitaire BP 53
38041 Grenoble Cedex 9, France
william.castaings@inrialpes.fr

Abstract. This paper focuses on parallelization process of DaveF, a new two-dimensional depth-averaged flow and sediment transport model that allows breach development and the resulting flood wave to be simulated simultaneously. Problems encountered during parallelization and techniques used to solve them are described. The experimental results with different input data on different machines are also included.

1 Introduction

Over the past few years, floods have caused widespread damages throughout the world. Most of the continents were heavily threatened. Therefore, modeling and simulation of floods in order to forecast and to make necessary prevention is very important. The kernel of flood simulation is a numerical modeling, which requires an appropriate physical model and robust numerical schemes for a good representation of reality.

Simulating river floods is an extremely computation-intensive undertaking. Several days of CPU-time may be needed to simulate floods along large sections of rivers. For critical situations, e.g. when an advancing flood is simulated in order to predict which areas will be threatened so that necessary prevention measures can be implemented in time, long computation times are unacceptable. Therefore, using HPCN (High Performance Computing and Networking) platforms to reduce the computational time of flood simulation is imperative. The HPCN versions of hydraulic models not only reduce computation times but also allow simulation of large scale problems, and consequently provide more reliable results.

Generally, the process of HPCN implementation consists of the following steps:

– Analyzing computational approaches used in the models: the methods of discretization (finite elements, finite differences, finite volumes), the algorithms (Newton iteration, frontal solution methods)

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 521–527, 2004.
© Springer-Verlag Berlin Heidelberg 2004

– Analyzing the source codes of the models: the program and data structures, data references
– Choosing appropriate methods for parallelization
– Coding, testing, and debugging HPCN versions
– Installing and deploying HPCN versions

2 Computational Approaches of DaveF

DaveF is the second flood model that has been parallelized during ANFAS project [4] (datA fusioN for Flood Analysis and decision Support, European Union 5th Framework Programme, IST-1999-11676). The first model is FESWMS [3] (Finite Element Surface Water Modeling System) and its parallelization has been published in other papers [1]. Although both models are used for simulation of water flow, they use completely different computational approach.

In DaveF, the depth-averaged surface-water flow and sediment transport equations are solved numerically using a two-dimensional, explicit in time, cell-centered, Godunov-type, finite volume scheme. Godunov-type methods for solving first-order hyperbolic systems of equations are based on solutions of initial value problems, known as Riemann problems, involving discontinuous neighboring states. These methods are found to be accurate and robust when used to solve partial differential equations describing fluid motion, being able to capture locations of shocks and contact surfaces. The experiments with DaveF for Vah river in Slovakia and Loire river in France showed clear advantages of DaveF and finite volume approach over finite element approach in FESWMS model, especially for simulation of complete flood hydrograph.

The complete mathematical approach is presented in a separate paper [5]. This paper focuses on problem encountered during its parallelization and solutions for the problems. Therefore, the following description of computational approaches is purely from the view of parallel programming.

In each time step, DaveF computes the solutions (water levels and velocities) of each cell from its current values and the values of its neighbors. At first sight, it seems to be easily parallelized, however, more careful study shows a big parallelization problem of the computation: the fine granularity. DaveF generally uses very small time steps, and a small amount computation is needed in each time step (to compensate for the large number of steps). Although at each time step, the calculations and solution updating of each cell can be done in parallel without communication, in the next time step, calculating the new solution of an element requires the old solutions of its neighboring cells. It means that in distributed-memory systems like clusters, each processor has to send the solutions of all cells on the border with another processor to the processor before starting the next time step.

The source code of DaveF has about 24000 lines of FORTRAN program instructions. The topography data (cells, nodes) are declared as global arrays that are used in nearly all computational routine. The solutions (water levels and

velocities) are stored in the cell and node arrays. Computations are concentrated in the central nested loop: the outer loop *for_each_time_step* and the inner loop *for_each_cell*. Most of computational routines are called from inside the loops and requires direct access to the arrays.

3 Problems Encountered during Parallelization with MPI

Parallelizing an existing program is completely different from writing a parallel program from scratch and there are many problems that programmers do not encounter during writing new parallel programs. The source code is written by experts in hydraulics. Although the mathematical approach (finite volumes) is well-known, there are many details that are not understandable for the experts in parallel programming who parallelize the source. Furthermore, the source code is large and it is not easy to change the sequential algorithms to the parallel ones without modifying too much of the source code.

Profiling tools (e.g. gprof in Linux) are extremely useful for parallelizing sequential programs. By using profiling tools the programmers can easily identify the computation-intensive parts in the source code (computation kernel), see the call graphs and analyze the performance of the program. Programmers then can concentrate to studying the computation kernel that is needed to parallelize/optimize and consider the rests of the source code as *blackbox*.

Paralleling with MPI for Linux clusters adds some more problems. It may be arguable if writing a parallel program from scratch with MPI on distributed-memory architectures like Linux clusters is easier or more difficult than with OpenMP on share-memory systems like supercomputers. However, for parallelizing existing sequential programs, it is much easier to do it with OpenMP because:

- In OpenMP, usually only the computation kernel is executed in parallel and the rests (e.g. input, output) are executed in sequential mode. Therefore, in the parallelization only the computation kernel is modified and the rest are left untouched. In MPI, the programmers need to take care also the rest even parallelizing these parts have negligible performance improvement. Input and output are often not parallelized, however, they have to be modified during parallelization, too. Typical construction

```
if (there is error in input data)
   then print error message and exit
```

may cause some MPI processes hang infinitely because the program does not terminate correctly (MPI_Finalize() routine should be called by all processes before exit).

- OpenMP does not require changing the data structures (e.g. large arrays) like MPI. Note that most of the routine access to the arrays and changing data structures may touch whole program.

The mentioned problems do not apply only for parallelizing DaveF but also for parallelizing FESWMS and can be generalized for parallelizing other applications.

4 Parallelization Approach

As stated in Section 1, in the current computational approaches of DaveF, each processor has to exchange the solutions of all cells on the border with its neighbors in each time step. The problem is that the computation time of computing the new solution of all cells on a processor is too small, in comparison with the time needed for exchanging bordering cells (fine granularity). There are several solutions:

1. To decrease the communication time: to use supercomputers or clusters with Myrinet network that have much lower latencies.

2. To increase computation time: The large number of cells is assigned to a processor, the larger computation time is needed and the better computation-to-communication ratio. Therefore, the larger area is simulated, the better speedup can be achieved.

3. To change the used algorithm: The main idea of this approach is to group cells with similar properties to *supercells.* This approach allows larger lengths of global time steps by using different local time steps for each group of cells, which implies fewer time steps (and larger computation in each step), less communication and better speedup. Using *supercells* can also significantly improve the performance of the sequential version, as cells in less critical area (floodplain) can have much less time steps (and less computation is requires). However, the approach requires reorganization of the mesh, which changes the global arrays of cells and nodes and consequently cause modification of whole program. Such changes are not desirable in current stable version of DaveF, therefore, the approach will be implemented in the next version of DaveF.

4. To overlap communication by computation: this solution does not solve the problem of fine granularity, but can reduce its effect on performance. It exploits the fact that computation and communication can be don in parallel so during the communication, processors can perform other computations. The algorithm can be describe as follows:

```
For each time step
 Compute new solutions for bordering cells
 Send the new solutions to the neighbor processor
 Compute new solutions for internal cells
 Receive new solutions of bordering cells from neighbor
                                                    processor
```

The time chart of the algorithm can be described in Fig.1 . As shown in the chart, although the communication time is long, the processor utilization is still good because during the communication, processors are busy with other work so the wasted CPU time is minimal. This approach is implemented in the current parallel version of DaveF.

The second largest problem of parallelizing is the global arrays of elements and nodes. Distributing these arrays among processors would cause modification of these arrays (e.g. if each processor have m items of the arrays, item with index i became item with index $i-(i/m)*m$ on processor i/m), that implies modification

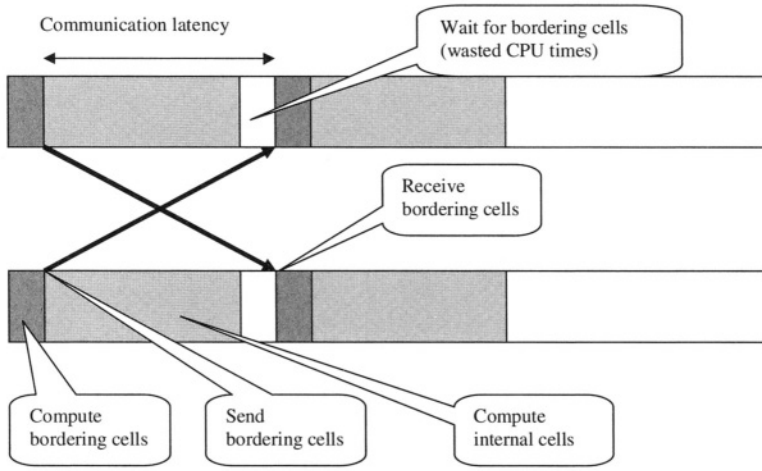


Fig. 1. Computation/communication overlapping

of those routines that use the arrays. As the arrays are global variable and most of routines use them, such changes would cause rewrite nearly whole program.

The solution is not to distribute the arrays among processors but to duplicate the arrays on every processor. Each processor now has whole array although it needs only a part of them. Nothing needs to be modified in routines that use the arrays, and the mount of modified code is minimal. Although this solution cause a little memory waste, the amount memory needed for the arrays is insignificant in comparison with the amount of memory needed to store the global matrix.

Note that DaveF is being continuously improved, and the more code is modified in the parallel version, the more difficult it is to upgrade the parallel version to the new version of DaveF. Minimizing code modification allows synchronizing the code of HPCN version with sequential code much easier.

5 Experimental Results

Experiments have been carried out on two sites: a Linux cluster at the Institute of Informatics (II-SAS) in Slovakia and INRIA icluster in France. The Linux cluster at II-SAS consists of 16 computational nodes, each of which has a Pentium IV 1800 MHz processor and 256 MB RAM. All of the nodes are connected by an Ethernet 100Mb/s switch. The INRIA icluster [6] has 216 HP e-vectra nodes (Pentium III 733 MHz and 256 MB RAM) divided into five segments. Nodes in a segment are connected by 100Mb/s Ethernet and segments are connected by 5 HP procurve 4000 switches.

Input data for the experiments are taken from Vah river in Slovakia and Loire river in France. Fig.2 shows the speedup of DaveF on II-SAS with two different input data from Loire river, one is four time larger than the second one. It is easy to see that the speedup is increased with the size of input data, especially

for larger number of processors. The reason is the fine granularity of DaveF, the more processors are used the larger is effect of the granularity performance.

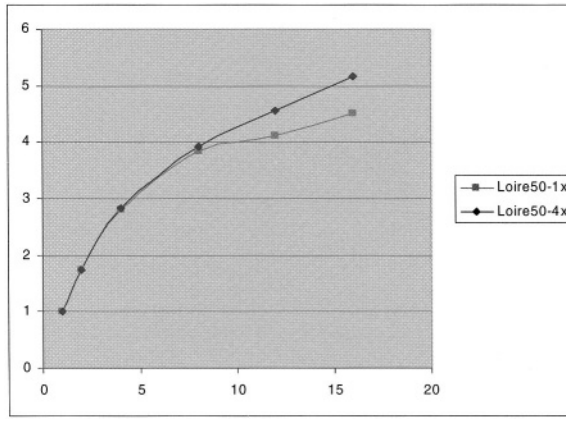


Fig. 2. Speedup of DaveF on II SAS cluster

Fig.3 shows the speedup of DaveF on INRIA icluster. The speedup is smaller than on II-SAS cluster because the network interference with other applications running on the system, especially when the nodes are not on the same segment (nodes are assigned to applications by PBS batch scheduling system). The speedup reaches maximum for 32 processors, for more processors, the speedup begins decrease because communication delays become to large for computations (the number of messages linearly increases with the number of processors while the computation time decrease).

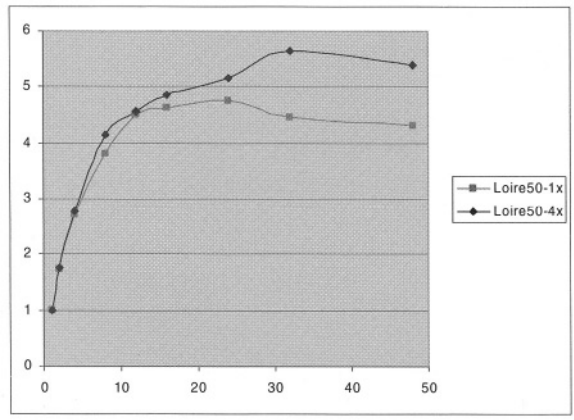


Fig. 3. Speedup of DaveF on INRIA iluster

6 Conclusion and Future Work

In this paper, parallelization process of DaveF flood model has been shown. The problems encountered and their solutions during parallelization process can be applied for parallelizing other applications, too. Despite the fine granularity, the parallel version of DaveF show good performance on Linux clusters.

At the moment, DaveF has been ported to Grid environment in CrossGrid project [7] and is running in CrossGrid testbed [8]. The details of Grid-aware Flood Virtual Orgranization, where DaveF is used, are described in a separate paper [2].

Acknowledgements. This work is supported by EU 5FP CROSSGRID) IST-2001-32243 RTD and the Slovak Scientific Grant Agency within Research Project No. 2/3132/23

References

1. L. Hluchy, V. D. Tran, J. Astalos, M. Dobrucky, G. T. Nguyen, D. Froehlich: Parallel Flood Modeling Systems. International Conference on Computational Science ICCS'2002, pp. 543-551.
2. L. Hluchy, V. D. Tran, O. Habala, J. Astalos, B. Simo, D. Froehlich: Problem Solving Environment for Flood Forecasting. Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting 2002, pp. 105-113.
3. FESWMS – Finite Element Surface Water Modeling. http://www.bossintl.com/html/feswms.html
4. ANFAS Data Fusion for Flood Analysis and Decision Support. http://www.ercim.org/anfas/
5. D. Froehlich: IMPACT Project Field Tests 1 and 2: Blind Simulation by DaveF. 2002.
6. icluster project. http://www-id.imag.fr/Grappes/icluster/materiel.html
7. EU 5FP project CROSSGRID. http://www.crossgrid.org/
8. Marco, R.: Detailed Planning for Testbed Setup. The CrossGrid Project, 2002. http://grid.ifca.unican.es/crossgrid/wp4/deliverables/CG-4-D4.1-001-PLAN.pdf

High Frequency Electromagnetic Fields Analysis with the Use of the Parallel FDTD Method

Wojciech Walendziuk[1], Jaroslaw Forenc[1], and Andrzej Jordan[2]

[1] Bialystok Technical University, Faculty of Electrical Engineering
Wiejska 45D Street, 15-351 Bialystok, Poland
{walenw, jarekf}@pb.bialystok.pl,
http://www.pb.bialystok.pl
[2] Polish-Japanese Institute of Information Technology
Koszykowa 86 Street, 02-008 Warsaw, Poland
jordana@pjwstk.edu.pl
http://www.pjwstk.edu.pl

Abstract. In the paper the parallel algorithms of the Finite-Difference Time-Domain method are presented. Those algorithms are based on the space domain decomposition. In the presented work, communications among computation nodes in a cluster of PCs and the efficiency of the parallel algorithms are also discussed....

1 Introduction

The examining of high frequency electromagnetic waves propagation becomes one of the most important issues of contemporary science. There are two ways of performing the electromagnetic field scattering analysis. One of them is the practical measurement research, which is usually expensive, and the second one is numerical simulations. The numerical analysis enables us to observe important phenomena happening in the examined media such as electromagnetic field distribution, an absorption of radiation or temperature increase. It is also possible to watch particular components of the electromagnetic field in any time moment and place in the computation area. In the numerical analysis of the electromagnetic fields of high frequency the FDTD (The Finite-Difference Time-Domain) method is often used [1,2,3,4]. This method is based on Maxwell time dependent equations which are directly integrated in time and space domains.

$$\nabla \times \boldsymbol{H} = \sigma \boldsymbol{E} + \varepsilon \frac{\delta \boldsymbol{E}}{\delta t} \qquad \nabla \times \boldsymbol{E} = -\mu \frac{\delta \boldsymbol{H}}{\delta t} \tag{1}$$

where:

- $\boldsymbol{E}$ - vector of the electric field intensity [V/m],
- $\boldsymbol{H}$ - vector of the magnetic field intensity [A/m],
- μ - magnetic permeability [H/m],
- σ - electric conductivity [S/m],
- ε - electric permittivity [F/m].

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 528–535, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The Maxwell equations, after the transformation into six scalar equations, are transformed into a differential notation, accordingly to the K.S.Yee algorithm [1]. This algorithm assumes that particular components of the electric $\boldsymbol{E}(E_x, E_y, E_z)$ field intensity and magnetic $\boldsymbol{H}(H_x, H_y, H_z)$ field intensity are situated in the nodes of an elementary Yee cell (Fig.1).Finally, the examined computation area is filled up with these Yee cells, and each node of the Yee cells stands for nodes of the finite differences grid in the numerical algorithms.

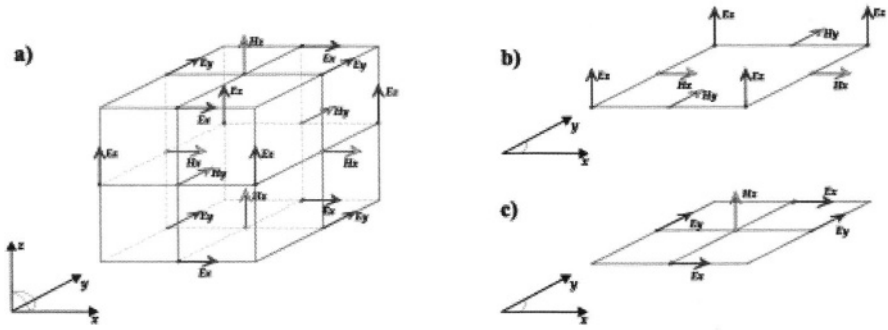


Fig. 1. Location of the components of the electric and magnetic field intensity in an elementary Yee cell for a three-dimensional case (a), for two-dimensional TM (Transverse Magnetic) case (b) and for two-dimensional TE (Transverse Electric) case (c).

To guarantee the numerical stability of the parallel FDTD algorithms, the time step was established for the two-dimensional case as follows:

$$\delta t = \delta x / c\sqrt{2} \tag{2}$$

and for the three-dimensional case,

$$\delta t = \delta x / c\sqrt{3} \tag{3}$$

where:

δt - time step of the numerical algorithms [s],
δx - size of the homogenous Yee cell [m],
c - speed of the light [m/s].

In addition, to limit the computation space, the Mur's absorbing boundary of the first order were used[2,3,4].

2 The Parallel FDTD Algorithms

The research was conducted in the Master-Slave topology. In such a case, the Master computer (e.g. in cluster) initiates the computation process, and the

Slave computers take the role of independent computation nodes equipped with a homogenous algorithm. In the presented algorithms, the communication among computation nodes was established with the use of the blocking message passing MPI instructions.
The general course of the computation process may be presented as follows:

1. Master node: sends data to the computation nodes.
2. Slave nodes: receive data initiating computations.
3. Slave nodes: memorize data needed for computations $\boldsymbol{E}$ components of the Mur's ABC.
4. Slave nodes: compute the electric intensity components $\boldsymbol{E}$.
5. Slave nodes: compute $\boldsymbol{E}$ components of the Mur's absorbing boundary conditions.
6. Slave nodes: compute the magnetic intensity components $\boldsymbol{H}$.
7. Slave nodes: exchange the data between neighbouring computations nodes.
8. Slave nodes: record the obtained results at the end of computations.

It must be mentioned that during the process of computation initiation, the data concerning the size of the analyzed space, places of divisions to sub-areas and boundary conditions are sent to the computation nodes.

3 "Stripes" and "Rectangle" Types of the Communications in a Two-Dimensional Algorithm of the FDTD Method Applied in a Homogenous Cluster of PCs

3.1 Data Exchange between Computation Nodes

A parallel algorithm of the two-dimensional FDTD method was used to divide the space domain into "Stripes" [5,6]. Such a decomposition is based on the division of the examined area into smaller parts in which communication takes place only on the parallel edges of the division. Particular computation nodes conduct parallel computations and then exchange data needed to initiate the next computation step. As it can be observable in Fig.2, in the first step there occurs the magnetic intensity components $\boldsymbol{H_y}$ data exchange (the white arrows), and then - the exchange of the electric intensity components $\boldsymbol{E_z}$ (the grey arrows). During the computations in a homogenous cluster system, the sub-areas computed in particular nodes are of the same size. Therefore, the quantity of the data exchanged among nodes during their communication is equal.

The other way is dividing the sub-areas in such a way that the communication takes place among all the borders of the sub-areas. This type of the communication is called "Rectangle". In such a case the computation nodes exchange also the data considering the vector of the magnetic field intensity $\boldsymbol{H_x}$ (the black arrows). Moreover, in this case, the sub-areas of the homogenous cluster have equal sizes (Fig.3).

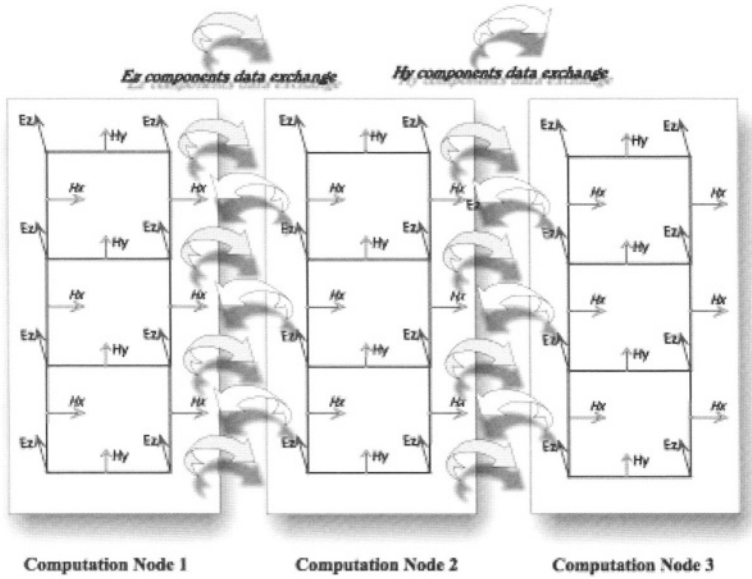


Fig. 2. The "Stripe" decomposition with the communication data exchange.

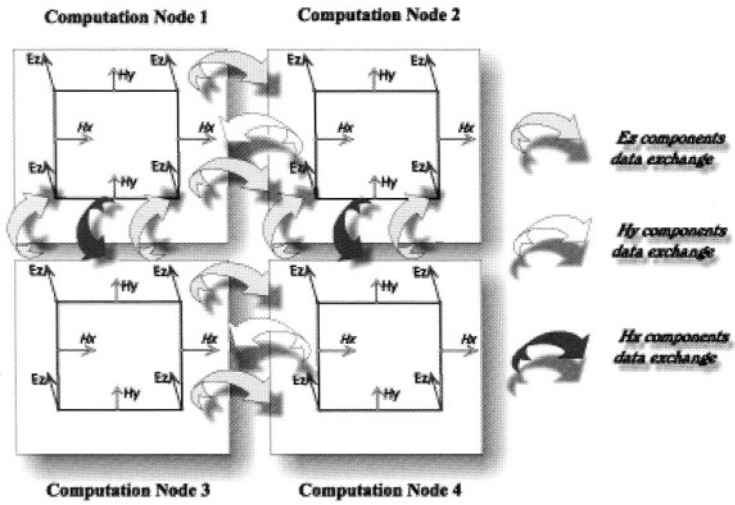


Fig. 3. The "Rectangle" type of the communication between computation nodes.

3.2 Cluster Description and Obtained Results

Seven identical PC units were used for the computations (six computation nodes + one control unit). Each node was equipped with an Intel Pentium II 866 MHz

processor, RAM 128 MB memory, a main board and a 30 GB hard disk of the same trademark. The computations were conducted in the Microsoft Windows ME operational system with the use of the WMPI 1.2 communication library. The computers used for computations were connected via Ethernet 10 Mb/s (the real received transfer was 2 Mb/s). Such a connection significantly aggravates the efficiency of parallel algorithms' work. It is worth stressing that using such a cluster of PCs was determined by the fact that the machines are normally used for everyday work and didactic purposes. During a parallel algorithm efficiency examination it was observable that the speedup, defined as the coefficient of algorithm computation time made on a single sequential machine to the time needed by a multiprocessor system, rises with the increase of the number of the finite differences nodes. With the grid density of *1001*x*1001* elements, the received speedup for the "Stripes" type of communication was 4.0 for six computation nodes and one control node (Fig.4). For the "Rectangle" type of the communication between four computation nodes the speedup 3.0 was obtained.

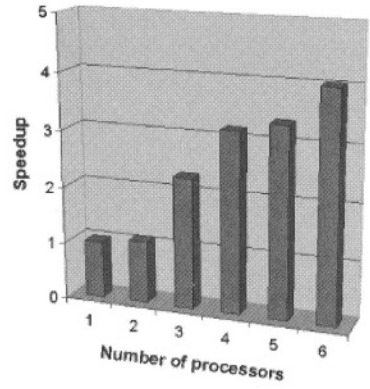


Fig. 4. The "Stripe" algorithm speedup for *1001*x*1001* density of the finite differences grids.

In order to recognize the relation between the calculation times and communication times, a series of analysis on two types of the connection topology was conducted (Fig.5). The relations were examined for different number of computation nodes used in parallel calculations. In both types of algorithms the size of the analyzed area and the number of time steps were not changed. During the survey, it was taken that each computation node receives an equal number of the finite differences grid nodes. In connection with the fact that both the computation and communication times constituted 99% of the parallel total time of the activity programmes (statistics do not take into account the time of saving data on the local hard disks), the diagrams were normalized to 100% form. It is worth

underlining the fact that the calculation times of certain computation nodes in the analyzed topology varied slightly. For this reason they were averaged. Next, a linearly lowering relation of the calculation time and communication time to the number of the computation nodes used has been observed.

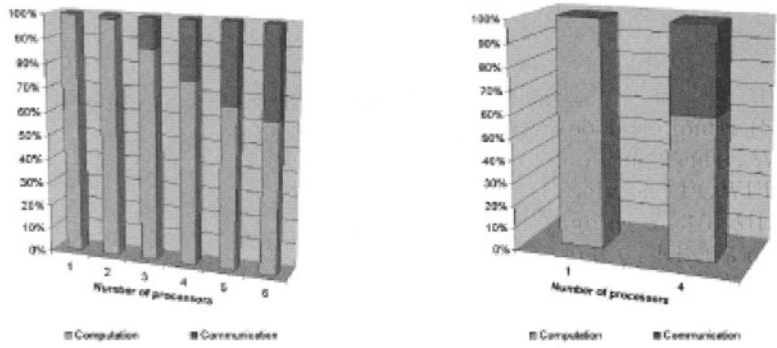


Fig. 5. The relation between the computation times and communication times in the "Stripe" (a) and in the "Rectangle" (b) types of the communication between computation nodes.

4 "Slices" Type of the Communication in a Three-Dimensional FDTD Algorithm in a Heterogeneous Cluster System

4.1 Data Exchange between Computation Nodes

The next phase of the research was the creation of a three-dimensional implementation of the FDTD algorithm shown in Fig.6. In such an algorithm the magnetic field intensity components $\boldsymbol{H_x}$ i $\boldsymbol{H_z}$ are sent to the left "neighbour" [7]. During the next time step activities of the algorithm, the electric field intensity components $\boldsymbol{E_y}$ and $\boldsymbol{E_z}$, which are already calculated, are sent to the right "neighbour". The remaining components - $\boldsymbol{E_x}$ and $\boldsymbol{H_x}$ - do not have to be sent, accordingly to the construction of the K.S.Yee algorithm, used in the FDTD method [1].

4.2 Cluster Description and Obtained Results

The research was conducted in a heterogeneous cluster system consisting of seven PCs (six computation nodes + one control unit), working under the Microsoft Windows 2000 PL operational system. The choice of the operational system was determined mainly by the computers assignment - they are not only used for computations, but also to the everyday work. The computers were connected

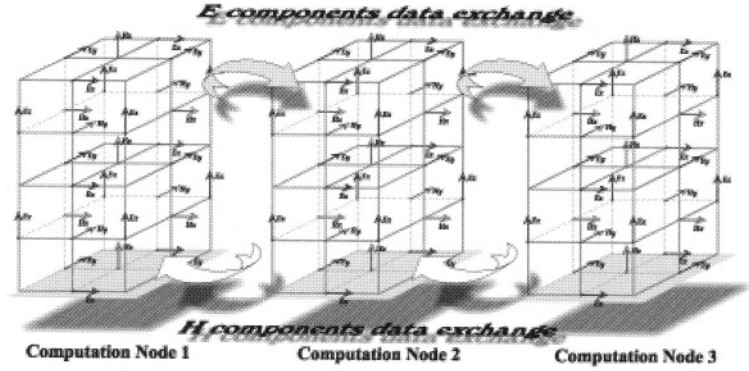


Fig. 6. The "slice" decomposition with the data exchange.

with the use of Fast Ethernet (100 Mb/s) by the Intel 520T switch. The system is fast enough to conduct numerical computations. The maximum bandwidth of the data transmission in this net accounts for about 7.8 Mb/s. Similarly, as in the previous two-dimensional cases, the MPI communication library was used. The presented algorithm was tested on a system consisting of *121*x*121*x*121* finite differences grid density. Obviously, it is not the maximum size of an area that can be examined. A selected model was only used for the algorithm efficiency evaluation. Fig.7 shown the chart of the received speedup. At the six computation processors it accounted for about 5.3.

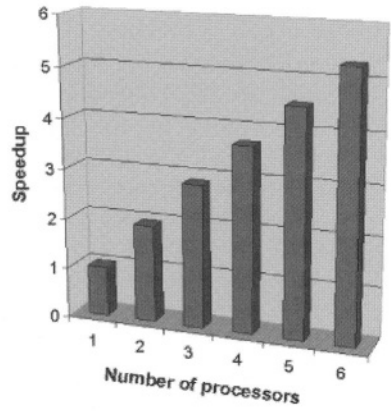


Fig. 7. The computation speedup in the three-dimensional FDTD method.

5 Conclusion

In the presented paper the examples of parallel FDTD algorithms implemented in a cluster system are shown. Using that type of parallel environment brings an opportunity to accelerate the computations of the electromagnetic wave propagation. It is possible in a situation when the communication among the computation nodes has a large bandwidth, and therefore - a great speed of a particular personal computer. Another important advantage is the possibility of enlarging the examined area, even till the size limited by the capacity of the operational memory of all the nodes working in the cluster. During the parallel algorithm efficiency examination, satisfying results of the speedup were obtained. Moreover, it was observed that the speedup was increasing together with the enlargement of the number of the computation nodes. Additionally, the obtained results of the relations of the computation time to the communication time were satisfactory. Those relations can be improved, when the computation time inside the nodes can be increased relatively to the communication time. It is possible to do it when the connection between the computation nodes is realized by a faster network. Sometimes even slow network connections between computation nodes do not cause a big problem, when the size of the examined area plays a greater role than the speed of computations.

Acknowledgements. The work has been sponsored by KBN grant 4T11C 007 22 and an internal PJIIT research grant.

References

1. Yee K.S.: Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, IEEE Transactions on Antennas and Propagation, **AP-14** (1966) 302–307
2. Mur G.: Absorbing boundary conditions for the finite difference approximation of the time-domain electromagnetic-field equations, IEEE Transactions on Electromagnetic Compatibility, **EMC-23/4** (1981) 377–382
3. Taflove A., Hagness S.C.: Computational Electrodynamics, The Finite - Difference Time - Domain Method. Boston, Artech House, Inc. 2000
4. Kunz K.S., Luebbers R.J.: The Finite Difference Time Domain Method for Electromagnetics. Boca Raton, CRC Press, Inc. 1993
5. Walendziuk W.: Forenc J.: Decomposition of the computation area in the parallel implementation of the FDTD algorithm, ISPDC'2002, Informatica **Tomul XI** (2002) 180–192, July 17-20, 2002 ,"Al.I.Cuza" University of Iasi, 2002
6. Walendziuk W.: SAR and thermal computations with the use of parallel FDTD algorithm, PARELEC'2002, Warsaw, 180–192, September 22-25, 2002 ,Polish-Japanese Institute of Information Technology, IEEE. Los Alamitos, 2002
7. Forenc J., Skorek A.: Analysis of high frequency electromagnetic wave propagation using parallel MIMD computer and cluster system, PARELEC'2000, Trois-Riveres, 176–180, August 27-30, 2000, IEEE. Los Alamitos, 2000

Genetic Clustering as a Parallel Algorithm for Approximating Basins of Attraction

Katarzyna Adamska

Instytut Informatyki, Uniwersytet Jagielloński, ul. Nawojki 11, 30072 Kraków,
adamska@ii.uj.edu.pl

Abstract. Genetic clustering consists in performing the analysis genetic optimization results using a clustering technique to get approximations of central parts of attractor of a multimodal objective. This work presents how outputs of Hierarchical Genetic Strategy can be clustered with EM algorithm. The approach gives an opportunity of theoretical analysis aimed on evaluating of approximation accuracy. In considered case genetic clustering can be easily implemented in parallel.

1 Motivation

Global optimization of a multimodal function is a practically important and challenging task. Genetic algorithms can be an operative tool for dealing with that problem. However, it is often very resources-consuming to get the exact solutions using genetic optimization. It seems to be more efficient to use a genetic algorithm just to determine the central parts of basins of attraction (see definition in [4,9,13]) for extremal points of an objective. Then exact solutions can be easily found with a local optimization method started from those areas. Approximations of basins of attraction can also take attention for themselves in some applications - for example in modeling of chemical reactivity.

2 The Idea of Genetic Clustering

Genetic clustering (see [7] for a comprehensive definition) is a strategy aimed on approximating central parts of basins of attraction by analyzing results of some genetic algorithm with a clustering method. The first attempt to genetic clustering has been done in [13] and is also referred in [4]. This work concerns the case when Hierarchical Genetic Strategy (HGS) is used as a genetic algorithm and clustering is performed with finite mixture model and EM algorithm. A simpler instance, based on SGA, has been considered in [6,7,8].

Hierarchical Genetic Strategy (HGS) (see [10] for definition and tests and [9] for detailed theoretical aspects) is a very effective multipopulational parallel global optimization algorithm. Calculation in HGS goes synchronously on several levels. For each level a set of parameters is defined. The higher level is, the lower mutation rate, the smaller population and the longer chromosome is set. As a consequence, low level populations have a sufficient mobility and play a role of

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 536–543, 2004.
© Springer-Verlag Berlin Heidelberg 2004

a control system, which searches a domain and refers populations of a higher level into areas which seem to be interesting. Populations on high levels are responsible for more accurate calculations and are expected to gather close to extrema. After [11,9] a run of HGS can be characterized as follows:
The strategy starts with the process of the lowest level 1 called root. After the fixed number of evolution epochs the best adapted individual is selected. This procedure is called a metaepoch of the fixed period. After every metaepoch a new process of the second order can be activated. This procedure is called sprouting operation. Sprouting can be generalized to higher accuracy levels, up to some fixed value. Metaepoch calculation is continued for all existing populations. Sprouting is performed conditionally, according to the outcome of the branch comparison operation. Details of both operations depend on the HGS implementation. After stopping the algorithm, one gets on the highest accuracy level a set of populations occupying neighborhoods of different extrema in an admissible domain. In compare to other genetic algorithms HGS is noteworthily faster and enables finding many local extrema in a single run. [11]

For the purpose of further theoretical analysis one must assume that evolution in all HGS populations is controlled with the use of Simple Genetic Algorithm (SGA) and a domain is coded binary (however there have been done also successful tests with floating point coding [11]).

Clustering task is performed using *finite mixture models* [3]. The base idea of the approach is that distribution of clustered data can be described with some density function ρ. The main assumption of the finite mixture model is that the density function ρ may be represented by the convex combination of some components (1),

$$\rho(x) = \sum_{k=1}^{s} \gamma_k g_k(x; q_k) \tag{1}$$

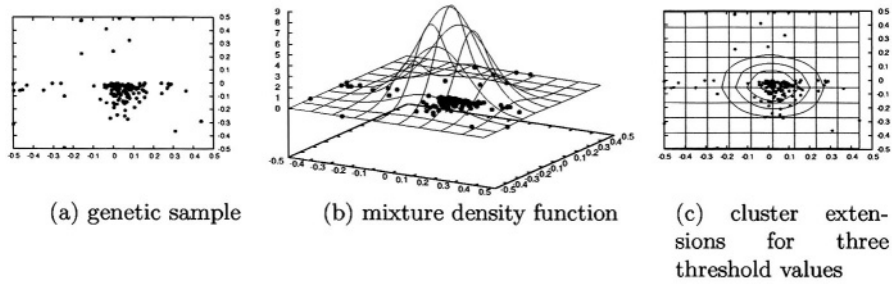
where $g_k(x; q_k)$ stands for a component function, depending on input data x and set of specific parameters q_k. Each component should describe one cluster, which is indexed with k. Coefficients γ_k in (1) are called *mixing proportions* and the function ρ is named a *mixture.* As in the most clustering techniques, the number of clusters s must be predicted. The functional form of components g_k is assumed, but their specific parameters remain unknown, so the mixture depends on the parameter vector $q = (q_1, \ldots, q_s)$. In most practical works components have a form of Gauss distribution, so a covariance matrix C_k and a mean vector m_k are characteristic parameters of a k^{th} component, $q_k = (C_k, m_k)$.

The mixing proportions γ_k must be also evaluated. To perform that a *probability matrix* $\Gamma = [\gamma_{i,j}]_{i=1,\ldots,s}^{j=1,\ldots,n}$ is introduced. An entry γ_{ij} of the matrix Γ stands for the probability that j-th data element belongs to i-th cluster. A mixing proportion γ_k is computed from Γ as a normalized sum over the k-th row; $\gamma_k = \frac{\sum_{i=1}^{n} \gamma_{ki}}{n}$ for all $k = 1, \ldots, s$.

An elementary calculation based on the Bayes rule shows that sets of mixing proportions and component parameters are related with each other. Because of that fact, given one of these values sets, the second can be calculated. It consti-

tutes the basis of the Expectation-Maximization Algorithm (EM) [1], which is the effective method of computing the mixture density function. One execution of the EM algorithm consists of two steps. The E-step is for calculating expected values of Γ entries. In the M-step component parameter vector q is worked out in such a way, that some auxiliary *likelihood function* [2, Sect. 3.2] is maximized. Starting the EM algorithm requires providing some initial evaluation of q or Γ. Then the iteration of EM steps begins.

As already mentioned, applicating a clustering algorithm for genetic optimization output data set is named a genetic clustering. Being more formal one can say that an input data set, which the finite mixture model is constructed for, is constituted by a *genetic sample.* As a genetic sample we understand a multiset of phenotypes included in an admissible set, unambiguously corresponding to a population from a genetic space (for a comprehensive explanation see [7]). If the genetic algorithm is able to find areas close to extremal points of an objective function (i.e an algorithm is *well-tuned,* see [4,7] for a formal definition), a genetic sample will be concentrated in these areas. Thus a mixture function ρ constructed for a genetic sample will have greater values in these parts of admissible domain which are close to extrema. Now, by cutting a mixture function for some threshold, we get level sets of the mixture. Those level sets will be called *cluster extensions.* The described process is illustrated on figure 1. The next section presents an argumentation why cluster extensions can be treated as approximations of basins of attraction.



(a) genetic sample (b) mixture density function (c) cluster extensions for three threshold values

Fig. 1. Steps of genetic clustering

3 Theoretical Analysis of Genetic Clustering Properties

3.1 Measure Density for SGA

Assume that a genetic population is a n-elemented multiset of elements from some r-sized genetic universum. A population can be represented by a frequency vector which i-th coordinate is a normalized contribution of an i-th genotype to a population. According to the theory of SGA introduced by Vose [14], evolution

of a population in SGA can be described with a genetic operator (named also "heuristics") $G : \Lambda^{r-1} \to \Lambda^{r-1}$. $\Lambda^{r-1} \subset \mathbb{R}^r$ denotes $r-1$-dimensional unit simplex, which contains frequency vectors for all populations. Each frequency vector constitutes a discrete measure in a genotype space [14, sect.3]. Since there is one-to-one coding between genotypes and phenotypes, the measure can be transported from a genotype space into a phenotype space D_r (refer to [5,7] for details). It has been shown in [5] that under some mild geometrical assumptions concerning an admissible set D and D_r, a discrete measure μ on a phenotype space D_r can be extended to the particular measure on the whole D, which has a density $\rho_\mu \in L^p(D)$. To distinguish between the measures let us take the following notation: if x stands for a population, let $\Theta(x)$ mean a corresponding discrete measure in a phenotype space, and $\rho_{\Theta(x)}$ a corresponding density on D. Note, that D_r is a discrete set included in D. The idea of constructing $\rho_{\Theta(x)}$ consists in dividing D onto hypercubes so that each point y from D_r is placed in a separate hypercube of the volume V_y. Size of a hypercube is dictated by the accuracy of coding. It is set $\rho_{\Theta(x)}|_{V_y} \equiv \frac{\Theta(x)(y)}{meas(V_y \cap D)}\ \forall y \in D_r$. As a result one gets a step-wise measure density defined on a set of adjacent hypercubes intersected with D (refer for details to [5]).

3.2 Error Estimations in Genetic Clustering

Genetic measure density described in section 3.1 reflects directly properties of genetic algorithm. Level sets of that function separate areas which can be treated as the best approximations of central parts of attractors available while using genetic optimization. Mixture density function (1) defined in section 2 can be also built on the base of genetic sample but its construction is controlled by the concentration of points only. However, the mixture density can be efficiently computed and moreover, the computation has a reasonable time and memory complexity. The question is, what the difference between those two density functions can be and how level sets of the functions can differ.

The construction of measure density mentioned above has been defined for SGA and it can be extended for HGS, if the basic evolutionary mechanism is SGA (see [9]). Let $G_l : \Lambda^{r_l-1} \to \Lambda^{r_l-1}$ denote heuristics for a population on l-th accuracy level in HGS and $1 \le l \le m$, $r_l = 2^{s_l}$ where s_l is accuracy of coding (chromosome length) on l-th level. Assume G_l is well-tuned (see sec. 2on previous page, [4,7]), has a unique fixed point z_l such that $\forall x \in \Lambda^{r_l-1}\ \lim_{t\to\infty} G_l^t(x) = z_l$ Let z_l correspond to the population that represents the best information about basin of attraction of an objective available for a given genetic algorithm (see [4], chapter 4)

Assume that after t_0 metaepochs there are b populations $p^{(1)}_{t_0,n_m}, \ldots, p^{(b)}_{t_0,n_m}$ of the size n_m on the highest level of accuracy. Each of them induces a discrete measure $\Theta\left(p^{(i)}_{n_m}\right)$ on D_{r_m}. After [9], the measure for all populations of m-th level can be defined:

$$\mu^b_{t_0,n_m} = \frac{1}{b}\left(\Theta\left(p^{(1)}_{t_0,n_m}\right) + \ldots + \Theta\left(p^{(b)}_{t_0,n_m}\right)\right) \tag{2}$$

On the base of the thoery developed by Schaefer and Kołodziej in [9] the following theorem can be formulated:

Theorem 1. $\forall \epsilon > 0 \forall \eta > 0$, $\exists N \in \mathbb{N}$, $\exists W(N) > t_0$, *such that* $\forall n_m > N, \forall t > W(N), \forall E \subset D$ *there is:*

$$P\left\{\left\|\rho_{\mu^t_{b,n_m}} - \rho_{\Theta(z_m)}\right\|_{L^p(E)} < \frac{\epsilon}{c}\right\} > 1 - \eta$$

where $\rho_{\mu^t_{b,n_m}}$ *is a density for the measure defined by 2 and* $\rho_{\Theta(z_m)}$ *is a density for the measure induced by a fixed point of* G_m, $p \in [1, \infty]$ *and* $c = \min_{i=1\ldots r_m} m(V_i)$.

Schaefer and Jabłoński in [5] have formulated theorems about probabilistic estimations for SGA measures and measure densities. As proved in [9], with the assumptions of the theorem 1 the analogous estimation for HGS measures holds:

$$P\left\{\left\|\mu^t_{b,n_m} - \Theta(z_m)\right\|_{L^p(E)} < \epsilon\right\} > 1 - \eta \tag{3}$$

Using (3) the proof of the theorem 1 can be done quite analogously as the proof for SGA densities given in [5]

Now let us return to the finite mixture model and the EM algorithm. Each iteration of the EM algorithm results in a set of parameters unambiguously determining a mixture density function. Assume that for a given input data the EM algorithm is convergent. Thus one can construct a contractive function $\mathcal{H}$ leading from a space of mixture parameters onto itself, which describes a single step of the algorithm. According to the theorem about an iterative function (well-known in numerical analysis, see for example [12]), the function $\mathcal{H}$ has a stationary point. A detailed description of construction of $\mathcal{H}$ and adaptation of the theorem for the EM algorithm is presented in [8]. Denote with ρ_i a mixture density which can be obtained in i-th step of the algorithm, and with ρ_S a density corresponding to a stationary point of the EM algorithm. As a conclusion from the theorem about iterative function one gets the following estimation (see [7,8] for details):

$$\|\rho_i - \rho_S\|_{L^p} \leq \frac{a^i}{1-a}\chi \tag{4}$$

where $a \in (0,1)$ is a constant of contraction of $\mathcal{H}$ and χ is a norm-dependent factor. Therefore by putting; together the theorem 1 and (4) one gets the following:

Proposition 1. $\forall i > 0$, $\forall \epsilon > 0 \forall \eta > 0$, $\exists N \in \mathbb{N}$, $\exists W(N) > t_0$, $\exists z \in$ *such that* $\forall n_m > N$, $\forall t > W(N) \forall E \subset D$

$$P\left\{\left\|\rho_i - \rho_{\Theta(z_m)}\right\|_{L^p(E)} < \frac{\epsilon}{c} + \frac{a^i}{1-a}\chi + \sigma_{GC}\right\} > 1 - \eta$$

where i *stands for the number of EM iterations and* $\sigma_{GC} = \left\|\rho_{\mu^t_{b,n_m}} - \rho_S\right\|_{L^p(E)}$ *(all the remaining symbols explained in the above text).*

For the purpose of this consideration it is enough to set $E = D$. The estimated difference $\left\|\rho_i - \rho_{\Theta(z_m)}\right\|_{L^p(D)}$ is important for the theory of genetic clustering, since it expresses the error of genetic clustering for HGS data. It is a difference between the best density available for genetic optimization with HGS and a mixture density calculated after i iterations of the EM algorithm. The first two quantities estimating the difference can be arbitrarily small; in the first component ϵ can be any positive number and the second decreases with the increase of value i. Thus the error of genetic clustering depends on the value of σ_{GC}. The factor σ_{GC} is an unavoidable error of the method. It will be now shown that σ_{GC} takes values of a certain order.

Assume that fitting for $\rho_{\mu^t_{b,n_m}}$ and ρ_S is optimal; it means that $\rho_{\mu^t_{b,n_m}}$ can be treated as approximating spline function for ρ_i, consisting of zero degree polynomials. Thus an approximation theorem can be utilized - complete consideration is presented in [6]. The theorem leads to the conclusion (5):

$$\sigma_{GC} = \left\|\rho_{\mu^t_{b,n_m}} - \rho_S\right\|_{L^p(D)} \leq C\,(measD)^{\frac{1}{p}}\, d^{max}\, |\rho_S|^{max}_{\infty,1} \tag{5}$$

where C is a constant dependent on norm and geometrical properties of division of D, $measD$ means the Lebesque measure of D, $d^{max} = \max_{V \subset D}\{d_V\}$ is a maximal diameter of a ball contained in V and $|\rho_S|^{max}_{\infty,1} = \max_{V \subset D}\left\{|\rho_S|_{\infty,1,V}\right\}$ where $|\rho_S|_{\infty,1,V} = \left(\sum_{|\alpha|=m} \int_V \left|\frac{\partial^\alpha h(x)}{\partial x^\alpha}\right|^p dx\right)^{1/p}$ stands for the seminorm in a Sobolev space.

It is obvious from (5) that if a number of hypercubes V goes to infinity, then the error of the method will decrease to zero. However, one has to remember (5) has been introduced with the assumption of optimal fit between ρ_S and $\rho_{\mu^t_{b,n_m}}$.

Proposition 1 supplemented with (5) gives a probabilistic estimation for difference between density functions. Nevertheless, for the purpose of approximation of basins of attraction, the estimation for level sets of those densities $L^{HGS}_\alpha = \left\{x \in D : \rho_{\Theta(z_m)}(x) \geq \alpha\right\}$, $L^{EM}_\alpha = \{x \in D : \rho_i(x) \geq \alpha\}$ would be more interesting. Unfortunately, it seems that estimations with use of the norms $\|\cdot\|_{L^p(D)}$ for measure densities is not always an adequate tool while trying to estimate level sets of these functions. It is possible that $\|\rho - \widetilde{\rho}\|_{L^p(D)}$ is quite small, but level sets differ significantly. Such a situation is possible for example when $\frac{\epsilon}{2} < \left|\rho(x) - \widetilde{\rho(x)}\right| \leq \epsilon$ for any $\epsilon > 0$ and for all $x \in E$, E simply connected and $measE$ is relatively big. Assume that $\forall x \in E\, \rho(x) < \widetilde{\rho(x)}$. Then, if level sets are defined with $\alpha = \frac{\epsilon}{2} + \max_{x \in E} \rho(x)$, E is included into level set for $\widetilde{\rho(x)}$ but not for $\rho(x)$. The example is illustrated on fig. 2.

4 Notes on Parallel Performance of Genetic Clustering

The proposed approach can be very easily performed using parallel computation. As mentioned at the beginning, HGS has been defined as a parallel algorithm

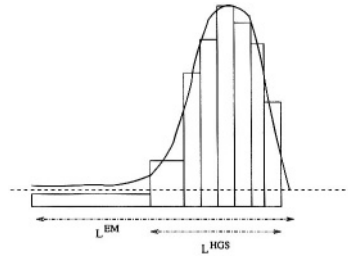


Fig. 2. An example of two density functions close in $\|\cdot\|_{L^p}$ but with significantly different level sets for a specific value of α

(see [10]). The EM algorithm, used as a clustering engine, can be also executed in parallel. Parameters describing a Gaussian component (and also a mixing proportion for it) are independent on properties of other components, thus set of parameters for each Gaussian can be computed in parallel. Analogously, each row of the probability matrix Γ contains probabilities of belonging of data elements to a given cluster, which is also independent from probabilities of classifying them to any other class. So calculating of Γ can be done in parallel, too. According to those remarks, a figure 3 presents a proposition of parallel execution of the EM algorithm.

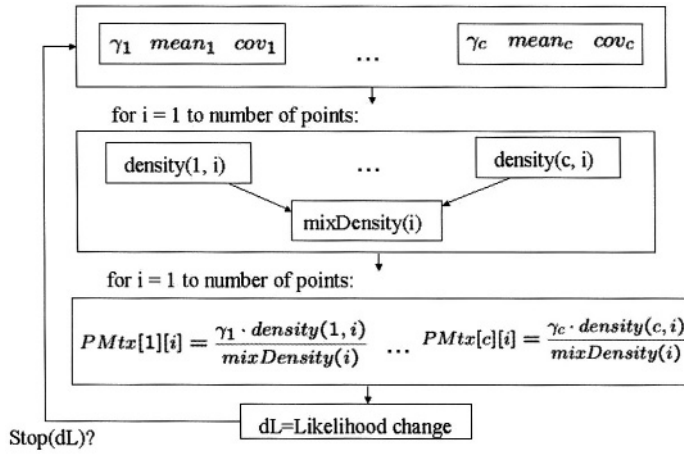


Fig. 3. A scheme of parallel execution for the EM algorithm.

5 Conclusions

- Genetic clustering performed for HGS data using EM algorithm constitutes an efficient approach to approximation of central parts of attractors of a multimodal function.
- The difference between measure density induced by HGS and computed in genetic clustering can be estimated. It depends on a factor σ_{GC} whose value is dictated by geometrical properties of the domain partition. However, the existing theoretical analysis of the method should be extended to get more accurate estimation for level sets of density functions.
- Genetic clustering can be in a straightforward manner implemented in parallel.

References

1. Dempster A.P., Laird N.M., Rubin D.B.; Maximum likelihood from incomplete data via EM algorithm, *J. Royal Statis. Soc., Ser. B,* vol. 39, pp. 1-38, 1977
2. Duda R.O., Hart P.E. *Pattern Classification and Scene Analysis,* Wiley, New York 1973
3. McLachlan G., Peel D., *Finite Mixture Models,* Wiley, 2000
4. Schaefer R., *Introduction to the genetic global optimization* (Podstawy genetycznej optymalizacji globalnej), Jagiellonian University Press, Kraków 2002 (in Polish)
5. Schaefer R., Jabłoński Z.J.: On the convergence of sampling measures in the global genetic search, *Lecture Notes in Comp. Science,* vol. 2328, pp. 593-600, Springer 2002
6. Schaefer R, Adamska K., On the genetic clustering with finite mixture models - error estimation and practical tests, to be appeared in *Proceedings of 6th KAEiOG,* 2003
7. Schaefer R., Adamska K., Jabłoński Z.J., Clustering driven by the genetic sampling measure, *Methods of Artificial Intelligence,* Proceedings of the Symposium on Methods of Artificial Intelligence AI-METH 2002, pp. 361-366, Gliwice, Poland, 2002
8. Schaefer R. Adamska-Piskorz K., Approximation of basins of attraction with mixture resolving method, *Proc. of Conf. on Evolutionary Algorithms and Global Optimization,* 23-25 Sept. 2002, Kraków
9. Schaefer R., Kołodziej J., Genetic search reinforced by the population hierarchy, *Foga Proc. 2002* Morgan, Kaufman, 2002
10. Schaefer R., Kołodziej J., Gwizdała R., Wojtusiak J.; How simpletons can increase the community development - an attempt to hierarchical genetic computation, *Proc. of 4th KAEiOG,* pp.187-197, Ladek Zdrój, 2000
11. Semczuk A., Wierzba B., Kołodziej J., Schaefer R., Hierarchical Genetic Strategy with real number encoding, to be appeared in *Proceedings of 6th KAEiOG,* 2003
12. Stoer J., Bulirsch R., *Introduction to numerical analysis,* Sect. 5.2, New York : Springer-Verlag, 1980.
13. Telega H.: *Parallel algorithms for solving selected inverse problems,* (in Polish). PhD Thesis, Academy of Mining and Metallurgy, Kraków, Poland 1999
14. Vose M. D.: *The Simple Genetic Algorithm,* MIT Press, 1999

Multiple-Deme Parallel Estimation of Distribution Algorithms: Basic Framework and Application

Chang Wook Ahn[1], David E. Goldberg[2], and R.S. Ramakrishna[1]

[1] Department of Information and Communications
Kwang-Ju Institute of Science and Technology, Gwangju 500-712, Korea
{cwan,rsr}@kjist.ac.kr
http://parallel.kjist.ac.kr/~cwan/
[2] Department of General Engieering
University of Illinois, Urbana, IL 61801, USA
deg@illigal.ge.uiuc.edu
http://www-illigal.ge.uiuc.edu/goldberg/d-goldberg.html

Abstract. This paper presents a basic framework that facilitates the development of new *multiple-deme parallel estimation of distribution algorithms* (PEDAs). The aim is to carry over the migration effect that arises in multiple-deme parallel genetic algorithms (PGAs) into probability distribution of EDAs. The idea is to employ two kinds of probability vector (PV): one each for resident and immigrant candidates. The distribution of crossbred individuals (that virtually exist on both kinds of PV) is then utilized by a new type of crossover, the *PV-wise crossover*. A multiple-deme parallel population-based incremental learning (P^2BIL) scheme is proposed as an application. The P^2BIL scheme closely follows the proposed framework that includes a new learning strategy (i.e., PV update rule). Experimental results show that P^2BIL generally exhibits solutions that compare favourably with those computed by an existing PGA with multiple demes, thereby supporting the validity of the proposed framework for designing multiple-deme PEDAs.

1 Introduction

Evolutionary algorithms (EAs) known as *estimation of distribution algorithms* (EDAs) or *probabilistic model-building genetic algorithms* (PMBGAs) have attracted due attention of late [1]. EDAs exploit probabilistic models of promising solutions found so far while efficiently traversing the search space.

EDAs iterate the three steps listed below, until some termination criterion is satisfied:

1. Select good candidates (i.e., solutions) from a (randomly generated) population (of solutions).
2. Estimate the probability distribution from the selected individuals.
3. Generate new candidates (i.e., offspring) from the estimated distribution.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 544–551, 2004.
© Springer-Verlag Berlin Heidelberg 2004

It must be noted that the third step is unique to EDAs. This is in contrast to recombination and mutation operations employed by simple GAs (sGAs). Although sGAs and EDAs deal with solutions (i.e., individuals) in quite different ways, it has been theoretically shown (and empirically observed) that their performances are quite close (to each other) [1,2,3]. Moreover, EDAs ensure an effective mixing and reproduction of building blocks (BBs) due to their ability to accurately capture the BB structure of a given problem, thereby solving GA-hard problems with a linear or sub-quadratic performance with regard to (fitness) function evaluations [1,4].

A large number of serial EDAs have been proposed. *Population based incremental learning* (PBIL) [5], *compact genetic algorithm* (cGA) [3], *mutual information maximization for input clustering* (MIMIC) [6], *extended compact genetic algorithm* (ecGA) [4], and *Bayesian optimization algorithm* (BOA) [7] are widely known in this respect. Most of these algorithms - ecGA and BOA in particular - have been shown to be effective in solving GA-hard problems.

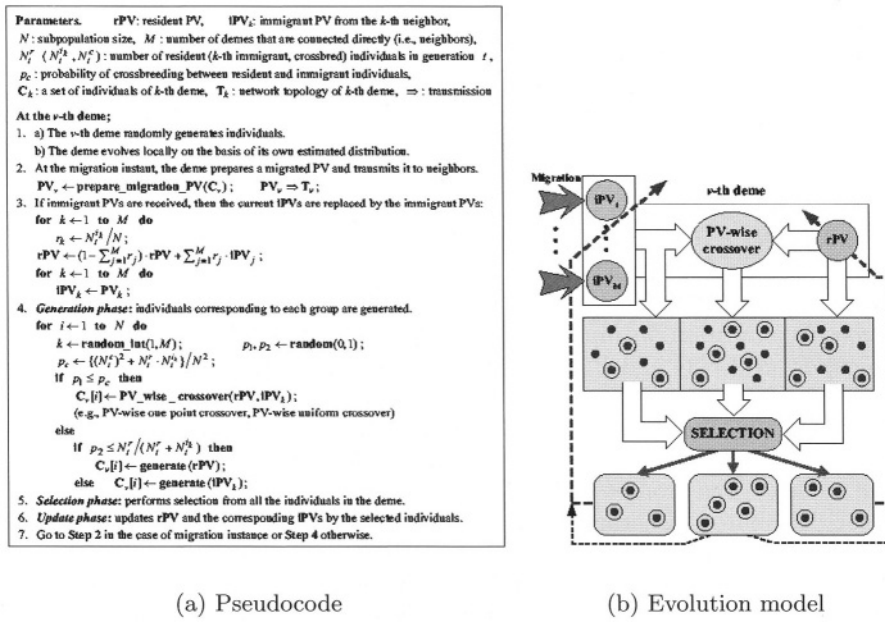
The computationally expensive (BB structure) learning phase (of sophisticated schemes) has prompted the discovery of several parallel EDAs such as parallel or distributed structure learning algorithms [8], PA1BIC and PA2BIC [1], to name a few. The focus is on speeding up structure-learning by decomposing the learning process into concurrently executable, independent tasks. However, the algorithms do not utilize the inherent parallelism in GAs. The parallel nature of GAs has considerable potential for reducing the execution time and finding a better solution. Multiple-deme PGAs make an attempt to take advantage of this parallelism. The algorithms achieve a higher performance with acceptable communication to computation ratio [9]. In this model, the total population is distributed over multiple denies (i.e., subpopulations) that occasionally allow migration or exchange of some individuals (among denies). Hence, they are also known as *coarse-grained, distributed,* or *Island model* GAs [9]. It appears that there is no PEDA that expressly strives to incorporate this migration effect (over probability distribution).

This paper proposes a novel framework for developing multiple-deme PEDAs. It can accurately model the migration effect by employing two kinds of probability (distribution) vector (PV): one of them takes into account the resident individuals, and the other estimates the distribution of immigrant individuals.

Section 2 describes the proposed framework of multiple-deme PEDAs. Section 3 suggests a multiple-deme parallel PBIL (P^2BIL) (as an application of the framework). Experimental results are found in Section 4, and Section 5 concludes the paper.

2 Proposed Framework for Multiple-Deme Parallel EDAs

This section describes a basic framework for designing multiple-deme PEDAs. As explained above, the migration scheme uniquely characterizes multiple-deme PGAs: some individuals in a deme occasionally migrate to other denies, thereby not only reducing the execution time but also improving the quality of the solution. In traditional multiple-deme PGAs, if once migration occurs, the immigrant



(a) Pseudocode (b) Evolution model

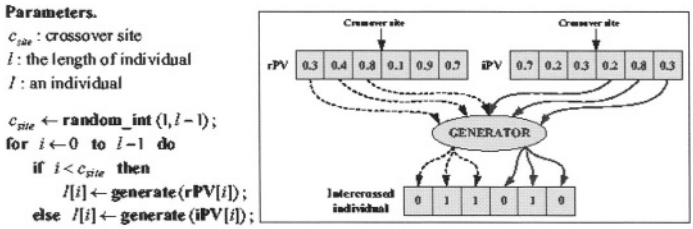
Fig. 1. Proposed framework for multiple-deme PEDAs.

individuals coevolve with resident individuals as long as they are not discarded. Thus, the objective is to realize the effect of the migration scheme with probability distribution.

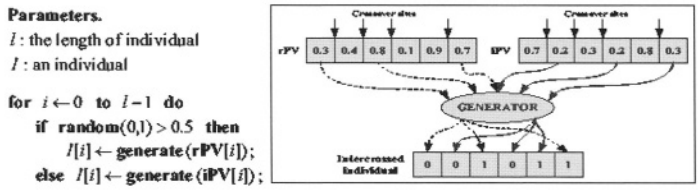
Two types of PV are employed for this purpose: resident PV (rPV) and immigrant PV (iPV). One rPV is used for estimating the distribution of promising solutions found by resident individuals, while the distribution of solutions obtained from the immigrant individuals is modeled by several iPVs (the number of iPV depends on the network topology). The latter feature brings along the ability to create the migration effect of multiple-deme PGAs over probability distribution. This is also a 3-phase procedure consisting of the following phases:

- Generation phase
- Selection phase
- Update (i.e., learning) phase.

The pseudocode and the evolution model of multiple-deme PEDAs are depicted in Fig. 1(a) and (b). Although the pseudocode is described for synchronous (migration) cases, it is easily expandable into asynchronous ones. In traditional multiple-deme PGAs, each deme holds three kinds of individuals: the first and second groups include the individuals created from the resident and immigrant individuals respectively; and the individuals incorporated by both the groups are classified as belonging to the third group. As can be seen in Fig. 1(a) and (b), the proposed scheme accurately maintains such individuals at the same time.



(a) PV-wise one-point crossover



(b) PV-wise uniform crossover

Fig. 2. Proposed PV-wise crossover.

That is, the distributions of the resident and immigrant individuals are modeled by rPV and iPVs, respectively. The distribution of crossbred individuals exists virtually on both kinds of PV.

In the *generation phase,* the offspring of the resident and immigrant individuals can be generated directly from their PVs (as in the traditional EDAs). The (concealed) distribution for generating the crossbred individuals is made available by a new kind of genetic operator, *viz.,* the PV-wise crossover. There are two kinds of PV-wise crossover: *PV-wise one-point crossover* and *PV-wise uniform crossover.* They are described in Fig. 2.

PV-wise one-point crossover generates the alleles of crossbred individuals from rPV and (selected) iPV on the basis of the crossover point. Similarly, PV-wise uniform crossover constructs the crossbred individuals by randomly generating genes from both the PVs. Without employing any additional PV for the mixed distribution, the PV-wise crossover can effectively generate the crossbred individuals (by incorporation of both the PVs). Furthermore, it does not require additional memory or computational resources. Although the PV-wise crossover is suitable to the PEDAs whose variables (i.e., genes) have no interactions (i.e., independent), it is straightforward to expand them for PEDAs that involve interactions. This is because each set of (some of) the variables (that are found by structure learning algorithms) can be dealt with independently.

In the *selection phase,* the number of dominant individuals among resident, immigrant, and mixture groups increases by degrees. The selection is performed

from all the individuals (i.e., total population in the deme) in a proportionate or ordinal-based manner. The number of selected individuals determines the next generation for each group.

In the *update phase,* the selected individuals update their respective PVs. It is depicted in Fig. 1(b). If a selected individual is a member of the resident group, then all of its characteristics are reflected in the rPV. Similarly, the individuals selected from the immigrant group update the iPV. This is clear because those individuals evolved from their species without any crossbreeding. However, the dominant individuals whose genes are created from crossbreeding by the PV-wise crossover must be treated differently. Since the characteristics of the individuals are inherited from both the species, all the PVs (i.e., rPV and iPV) are updated (by the individuals). For example, the traits of intercrossed individuals evenly influences both the PVs (for simple implementation).

It is important to note that, in the long run, the characteristics acquired from dominant individuals would spread across all the groups due to the evolution phases. Since this is exactly what is observed in the traditional multiple-deme PGAs, the objective of realizing migration scheme over probability distribution is successfully accomplished.

3 Multiple-Deme Parallel PBIL

As noted in Section 1, many EDAs with different learning strategies have already been developed. It is quite possible to devise PEDAs in a similar manner. A general framework that includes migration effects within probability distribution was proposed in Section 2. This section attempts to bring the existing EDAs under the purview of the basic framework for developing PEDAs by proposing a *multiple-deme parallel PBIL* (P^2BIL) scheme. PBIL has been selected for the purpose as it has attracted considerable attention from the research community due to the fact that it has been regarded as a premier scheme in the area of EDAs. Although a uni-directional ring topology (i.e., #rPV=1 and #iPV=1) is considered for simplicity, it is straightforward to expand it for complex topologies.

Baluja [5] introduced PBIL for work in a binary search space $\Omega = \{0,1\}^l$. The population of individuals in generation t is represented by a PV,

$$p_t(\mathbf{x}) = (p_t(x_1), \cdots, p_t(x_i), \cdots, p_t(x_l)) \tag{1}$$

where $p_t(x_i)$ is the probability of observing a value of '1' in the i-th allele.

The learning strategy (of PBIL), inspired by Hebbian rule, is defined by

$$p_{t+1}(\mathbf{x}) = (1-\alpha)p_t(\mathbf{x}) + \alpha\frac{1}{M}\sum_{k=1}^{M}\mathbf{x}_{k:N}^t \tag{2}$$

where $\alpha \in (0,1]$ is a relaxation factor that corresponds to the fading of the memory, M is the number of selected individuals and $\mathbf{x}_{k:N}$ denotes the k-th best individual (in a population of size N).

On the other hand, the tournament selection is a common feature as it is perceived to be effective in keeping the selection noise as low as possible. Therefore, the selection mechanism (of the PBIL) that is quite similar to (μ, λ)-selection is replaced by the tournament selection (without replacement). The update rule is slightly altered to

$$p_{t+1}(\mathbf{x}) = (1-\alpha)p_t(\mathbf{x}) + \alpha\frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k^t \tag{3}$$

where, $\mathbf{x}_k^t$ refers to a winner of tournament selection in generation t.

It is interesting to note that the modified PBIL can be viewed as a kind of cGA that is implemented on the population.

At this juncture, we propose the multiple-deme P^{2}BIL that exactly follows the mechanism depicted in Fig. 1(a) and (b). The P^{2}BIL employs (pairwise) tournament selection without replacement, PV-wise uniform crossover, and learning policy expressed by

$$\begin{aligned} p_{t+1}^r(\mathbf{x}) &= (1-\alpha_r)p_t^r(\mathbf{x}) + \frac{\alpha_r}{N_t^r + \lambda_t^r N_t^c}\sum_{k=1}^{N}\left[\mathbf{x}_k^t\left\{\delta(\mathbf{x}_k^t, \Omega_t^r) + \lambda_t^r\delta(\mathbf{x}_k^t, \Omega_t^c)\right\}\right] \\ p_{t+1}^i(\mathbf{x}) &= (1-\alpha_i)p_t^i(\mathbf{x}) + \frac{\alpha_i}{N_t^i + \lambda_t^i N_t^c}\sum_{k=1}^{N}\left[\mathbf{x}_k^t\left\{\delta(\mathbf{x}_k^t, \Omega_t^i) + \lambda_t^i\delta(\mathbf{x}_k^t, \Omega_t^c)\right\}\right] \end{aligned} \tag{4}$$

where $\lambda_t^r = \frac{N_t^r + 0.5N_t^c}{N}$, $\lambda_t^i = \frac{N_t^i + 0.5N_t^c}{N}$ and $\delta(\mathbf{x}_k^t, \Omega) = \begin{cases} 1, & \text{if } \mathbf{x}_k^t \in \Omega, \\ 0, & \text{otherwise.} \end{cases}$ Here, α_r (α_i) is a relaxation factor of rPV (iPV), N_t^r, N_t^i and N_t^c are as in Fig. 1 (a), Ω_t^r (Ω_t^i, Ω_t^c) is a set of resident (immigrant, crossbred) individuals, and λ_t^r (λ_t^i) refers to a contribution factor of the selected crossbred individuals to rPV (iPV). The subscript t stands for an instance of a generation, $N_t^r + N_t^i + N_t^c = N$ and $\lambda_t^r + \lambda_t^i = 1$.

It is clear that the basic principle of updating PVs is the same as in the (modified) PBIL. The contribution factors (λ_t^r and λ_t^i) acknowledge the fact that the crossbred individuals are constructed uniformly from both the PVs under the constraint expressed by $\lambda_t^r + \lambda_t^i = 1$; and their effects on each group are proportional to the number of individuals in the group.

Clearly, other serial EDAs such as ecGA, MIMIC, and BOA can also be parallelized by properly extending the learning strategies (i.e., PV update policies).

4 Experimental Results and Discussion

This section demonstrates the validity of the proposed framework for multiple-deme PEDAs by comparing the performance of a multiple-deme PGA with that of the multiple-deme P^{2}BIL. A (synchronous) uni-directional ring topology with 0.3 migration rate is employed in the experiments. Migration rate δ means that the migrated PV is prepared by the (best or random) δ-portion of individuals.

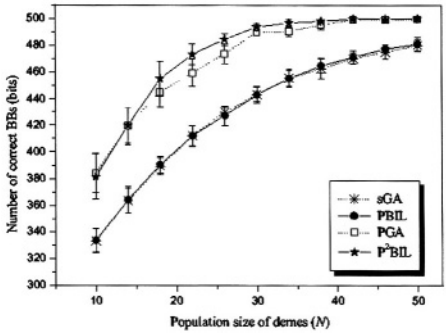


Fig. 3. Performance of the algorithms on 500-bit OneMax problem when migration occurs before convergence of demes.

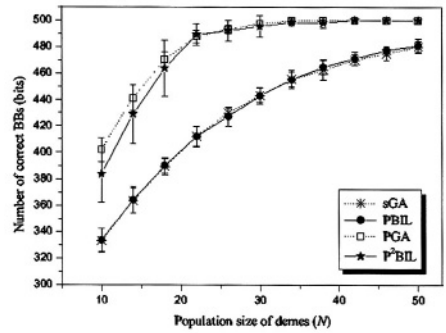


Fig. 4. Performance of the algorithms on 500-bit OneMax problem when migration occurs after convergence of demes.

Since PBIL was designed for solving problems with independent decision variables (i.e., genes), a 500-bit OneMax problem has been selected for tests. To investigate the effect of a migration instance, two kinds of migration were considered: *migration before, convergence* and *migration after convergence.* In these experiments, the relaxation parameter α of the (modified) PBIL was determined by an exhaustive search so as to achieve a performance that is comparable with that of sGA. That sets the stage for fair comparison. Further, α_r and α_i (of $\mathrm{P^2BIL}$) were set equal to α. All the results were averaged over 100 runs.

Fig. 3 compares the number of correct BBs (i.e., bits) returned by each algorithm. The migration of multi-deme PGA and $\mathrm{P^2BIL}$ occurs before the demes converge (i.e., it takes place at about 80% convergence). The migrants are chosen randomly from among the individuals in a deme and the immigrants randomly replace the individuals in the host deme (for the PGA). In Fig. 3, it is seen that the performance of the $\mathrm{P^2BIL}$ is comparable with that of the PGA. Clearly, the $\mathrm{P^2BIL}$ accurately implements the migration scheme of the PGA over probability distributions. It can also be observed that the $\mathrm{P^2BIL}$ slightly outperforms the PGA. The insignificant discrepancy can be attributed to the interactions that occur during the mixing of BBs within the finite population. Note that the $\mathrm{P^2BIL}$ offers an advantage over the PGA with regard to communication overhead: $\mathrm{P^2BIL}$ requires $\lceil l \cdot log_2(\delta \cdot N + 1) \rceil$ bits while PGA consumes $\lceil l \cdot \delta \cdot N \rceil$ bits per migration, because the $\mathrm{P^2BIL}$ prepares $\delta \bullet N$ individuals to be migrated through a probability distribution (i.e. PV) rather than individuals themselves.

Fig. 4 depicts the performance of each of the algorithms. This time around, the migration takes place after every deme has converged to (its own) solution. It is observed that the performance of $\mathrm{P^2BIL}$ is consistent with that of the PGA except when the population size is small. The slight discrepancy clearly comes from the fact that it is relatively hard to change the distribution after convergence because re-convergence time is very short when the population size is small. Note that although the performance discrepancy is not critical, small populations are not regarded as feasible candidates in practice.

As a result of our investigations, we believe that the proposed framework plays a significant role in providing a general guidance for developing multiple-deme PEDAs.

5 Conclusion

This paper has proposed a basic framework for designing multiple-deme parallel EDAs. The aim is to realize the migration effect of PGAs over probability distribution. To this end, two types of PV (i.e., rPV and iPV) have been employed. Resident and immigrant candidates (i.e., individuals) are adequately dealt with in this way. The crossbred individuals are treated with the new PV-wise crossover. The PV-wise crossover can extract (utilize) the concealed distribution of crossbred individuals from rPV and iPV without explicitly managing it. As an application of the suggested framework, a multiple-deme Parallel PBIL (*viz.,* P^2BIL) has also been developed in this paper. The P^2BIL exactly fits into the framework. It also includes a new learning strategy (i.e., PV update rule).

Experimental studies show that the proposed P^2BIL compares quite favourably with the existing multiple-deme PGA. The objective of implementing migration scheme on probability distribution has also been successfully achieved in this way. The proposed framework can be regarded as a promising candidate for designing multiple-deme PEDAs.

References

1. P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation,* Kluwer Academic Publishers, 2002.
2. C. W. Ahn and R. S. Ramakrishna, "Elitism-based Compact Genetic Algorithms," *IEEE Transactions on Evolutionary Computation,* vol.7, no.4, pp.367-385, 2002.
3. G. Harik, F. G. Lobo, and D. E. Goldberg, "The Compact Genetic Algorithm," *IEEE Transactions on Evolutionary Computation,* vol.3, no.4, pp.287-297, 1999.
4. G. Harik, "Linkage Learning via Probabilistic Modeling in the ECGA," *IlliGAL Technical Report No. 99010,* University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999.
5. S. Baluja, "Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," *Technical Report No. CMU-CS-94-163,* Carnegie Mellon University, Pittsburgh, PA, 1994.
6. J. S. De Bonet, C. L. Isbel, and P. Viola, "MIMIC: Finding Optima by Estimating Probability Densities," *Advances in Neural Information Processing Systems,* vol. 9, p. 424, 1997.
7. M. Pelikan, D. E. Goldberg, and E. Cantŭ-Paz, "BOA: The Bayesian Optimization Algorithm," *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'99), pp. 525-532, 1999.
8. W. Lam and A. Segre, "A Parallel Learning Algorithm for Bayesian Inference Networks," *IEEE Transactions on Knowledge Discovery and Data Engineering,* vol. 14, no. 1, pp. 93-105, 2002.
9. E. Cantŭ-Paz, *Efficient and Accurate Parallel Genetic Algorithms,* Kluwer Academic Publishers, 2000.

A Memory-Efficient Elitist Genetic Algorithm

Chang Wook Ahn, Ki Pyo Kim, and R.S. Ramakrishna

Department of Information and Communications
Kwang-Ju Institute of Science and Technology (K-JIST)
Oryong-Dong, Puk-Gu, Gwangju 500-712, Korea
{cwan,kimkp,rsr}@kjist.ac.kr
http://parallel.kjist.ac.kr/~cwan/

Abstract. This paper proposes a *memory-efficient elitist genetic algorithm* ($\mathrm{me}^2\mathrm{GA}$) for solving hard optimization problems quickly and effectively. The idea is to properly reconcile multiple probability (distribution) vectors (PVs) with elitism. Multiple PVs (rather than a single PV as in compact GA (cGA)) provide an effective framework for representing the population as a probability distribution over the set of solutions. A coordinated interplay amongst multiple PVs maintains genetic diversity, thereby recovery from decision errors is possible. On the other hand, reconciling with elitism allows a potentially optimal (elitist) solution to be kept current as long as other (competing) solutions generated from PVs are no better. This is because it exerts a selection pressure that is high enough to offset the disruptive effects of uniform crossover. It also attempts to adaptively alter the selection pressure in accordance with the degree of problem difficulty through pair-wise tournament selection strategy. Experimental results show that the proposed algorithm generally exhibits a superior quality of solution. Moreover, the proposed algorithm deploys memory more efficiently than extant sGA and cGA, especially when the problem is difficult.

1 Introduction

Genetic algorithms (GAs) have been successfully applied to solve many combinatorial optimization problems in business, engineering, and science [1,2]. A major characteristic of GAs is that they work with a population, unlike other classical approaches which do not do so. Hence, they can explore different regions of the solution space concurrently, thereby exhibiting enhanced performance. There is a downside to it, paradoxical though it may seem: it is extravagant on memory and computational effort. Naturally, there have therefore been efforts to have the best of both the worlds: to reap the benefits of a solution population with reduced memory- and computational-costs.

Baluja [3] introduced the population-based incremental learning (PBIL) that employs a probability (distribution) vector (PV) over the individual to represent its population. Each probability is updated independently. The updation rule is akin to that used in learning vector quantization. Inspired by the random walk model (used for investigating the sizing of population), Harik *et al.* [4] proposed

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 552–559, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the compact GA (cGA) that also represents the population as a PV over the set of solutions. Looked from the operational point of view, it mimics the order-one behavior of simple GA (sGA) with uniform crossover using a small amount of memory. Ahn and Ramakrishna [5] proposed a persistent elitist compact GA (pe-cGA) for solving hard optimization problems without compromising on memory- and computational-costs. It is achieved by addressing inherent defects of the cGA such as memory deficiency by employing elitism in an ingenious manner. Moreover, the algorithm is able to automatically alter the selection pressure in tune with the degree of difficulty of the problem (without requiring any problem dependent information).

In this paper, we propose a *memory-efficient elitist genetic algorithm* ($\mathrm{me}^2\mathrm{GA}$). The main objective is to efficiently and speedily solve difficult optimization problems such as fully deceptive problems using a small amount of memory. It can be accomplished by properly reconciling multiple PVs (for effectively representing its population) with elitism. Each PV represents a subpopulation over the set of solutions. A coordinated interplay amongst multiple PVs maintains genetic diversity that enables it to recover from decision errors. Moreover, elitism leads to fast convergence. It also carries partial knowledge about the gene's correlation by mitigating the disruptive effects of uniform crossover (by simply retaining the best solution found so far). A point of practical importance is that the $\mathrm{me}^2\mathrm{GA}$ operates as if the selection pressure is adaptively altered in tune with the degree of difficulty of the problem by employing pair-wise tournament selection alone. It is noteworthy that the $\mathrm{me}^2\mathrm{GA}$, like the pe-cGAs, does not require any problem dependent information that may not be available in practice.

2 Proposed Memory-Efficient Elitist GA

This section describes the proposed algorithm that combines multiple PVs with elitism in an ingenious manner. The objective is to improve the quality of solution to difficult optimization problems by deploying a very small amount of memory without compromising on convergence performance. Fig. 1 presents the pseudocode.

2.1 Representation

The $\mathrm{me}^2\mathrm{GA}$ is defined for work in a binary (genotype) space $\Omega = \{0,1\}^l$. The population of individuals in generation t is presented by multiple PVs,

$$p_t^i(\mathbf{x}) = (p_t^i(x_1), p_t^i(x_2), \cdots, p_t^i(x_l)), \quad \text{where } i \in \{1, 2, \cdots, v\}.$$

Here, $p_t^i(x_j)$ presents the probability (of observing a value of '1') of the jth element in the ith PV. All elements of PVs are initially assigned 0.5 to represent a randomly generated population (see **Step 1**). Since the $\mathrm{me}^2\mathrm{GA}$ manages its population as multiple PVs over the set of solutions (i.e., only models its existence), it can effectively mimic the behavior of sGA with uniform crossover.

Parameters. n : virtual population size, v : number of PVs, l : individual length, $\mathbf{C}$: a set of individuals, $\mathbf{E}_{chrom}$: elite individual, E_{index} : index of elite individual, I_{com} : index of a competitor

Step 1. Initialize probability vectors
$t \leftarrow 0$;
for $i \leftarrow 1$ **to** v **do**
  for $j \leftarrow 1$ **to** l **do**
    $p_t^i(x_j) \leftarrow 0.5$;

Step 2. Generate one individual for each PV
if $t = 0$ **then**
  for $i \leftarrow 1$ **to** v **do**
    $\mathbf{C}[i] \leftarrow$ **generate**$(p_t^i(\mathbf{x}))$;
else
  $\mathbf{C}[E_{index}] \leftarrow \mathbf{E}_{chrom}$;
  for $i \leftarrow 1$ **to** v **do**
    if $i \neq E_{index}$ $\mathbf{C}[i] \leftarrow$ **generate**$(p_t^i(\mathbf{x}))$;

Step 3. Find the best chromosome (i.e., elite chromosome)
$[E_{index}, \mathbf{E}_{chrom}] \leftarrow$ **find_elite_chrom**$(\mathbf{C})$;

Step 4. Update the PVs based on pair-wise tournament selection
for $i \leftarrow 1$ **to** v **do**
  $I_{com} \leftarrow$ **random**$\{1, \cdots, i-1, i+1, \cdots, v\}$;
  $winner, loser \leftarrow$ **compete**$(\mathbf{C}[i], \mathbf{C}[I_{com}])$;
  for $j \leftarrow 1$ **to** l **do**
    if $winner[j] \neq loser[j]$ **then**
      if $winner[j] = 1$ **then**
        $p_t^i(x_j) \leftarrow p_t^i(x_j) + 1/n$;
      else $p_t^i(x_j) \leftarrow p_t^i(x_j) - 1/n$

Step 5. Check if all the PVs have converged (with a tolerance ε)
for $i \leftarrow 1$ **to** v **do**
  for $j \leftarrow 1$ **to** l **do**
    if $p_t^i(x_j) > 0.0 + \varepsilon \wedge p_t^i(x_j) < 1.0 - \varepsilon$ **then**
      $t \leftarrow t + 1$; return to **Step 2;**

Step 6. The PVs represent the final solution

Fig. 1. Pseudocode of the me^{2}GA.

That is, it maintains genetic diversity in a more effective manner than the extant cGA (i.e., a single PV). In addition, it has the potential to recover from decision errors because multiple PVs further enhance the role of population as a buffer against such errors. This representation bears a resemblance to parallel GAs (rather than serial GAs) because each PV can be thought of as modeling each subpopulation (i.e., each deme). It implies that the me^{2}GA is naturally implemented over multiple nodes by allocating one PV to each processor. On the face of it, this approach seems to demand additional memory (as compared with cGA). However, this comparison is unfair. The relevant issue is concretely treated in Section 2.4 and Section 3.2.

2.2 Genetic Operators

Selection focuses on the exploration of promising regions in the solution space by providing high-quality individuals a better chance to get copied into the next generation [1,2]. On the other hand, crossover (i.e., recombination) increases the exploratory power by combining genes and BBs from fit individuals [1,2]. Hence, we employ selection and crossover as genetic operators. The result is a so-called *selectorecombinative* GA that performs global search [1]. Since selection is noisy as stated in Section 2.1, selection of better individuals may drag incorrect schemata into a final solution. Of course, the population being formed by a set of individuals plays a role in relaxing such decision errors [4]. It is common for (small) multiple populations with a certain migration policy (i.e., parallel GAs) to attain a better performance than a single large population with regard to quality of solution and convergence [6]. On the other hand, pair-wise tournament selection is perceived as a mechanism that keeps selection noise as low as possible.

Keeping this in view, we devised the following *seletorecombinative* mechanism. First, we subdivide the population into several equally sized subpopulations and apply uniform crossover to each subpopulation with a view to increase the exploratory power. Second, a number of individuals are chosen randomly

from a single subpopulation and, the best among them is kept as a competitor. This process is repeated on all the subpopulations. Third, each subpopulation is allowed to coevolve on the basis of pair-wise tournament selection among the best individuals.

The first and second processes can be realized by generating a (binary coded) individual on the basis of every current PV (see **Step 2**). It is noted that the generation of individuals (obtained from PVs) simulates the effects of uniform crossover that leads to a decorrelation of the population's genes. It is interesting to note the similarity between this process of generation and the process of observation (in quantum computing) that leads to the collapse of a superposition of states into an eigenstate with a certain probability.

At this juncture, we have the sampled competitors (equal to the number of PVs) which, with pair-wise tournament selection evolve their populations (i.e., the third process). In order to update (i.e., evolve) the first PV (i.e., first subpopulation), for instance, the first individual competes with one that is chosen randomly from the rest (i.e, the second to the last) of the individuals. In a virtual population of size n, the probability $p_t^1(x_j)$ is increased (decreased) by $1/n$ when the jth gene of the winner has an allele of '1' ('0') and the jth gene of the loser has an allele of '0' ('1'). If both the winner and the loser have the same allele in each gene, then the probability remains the same (see **Step 4**).

Note that the above scheme is identical to the extant cGA when it works with two PVs because each element of the two PVs will always have the same values (i.e., for all j, $p_t^1(x_j) = p_t^2(x_j)$). It is also interesting to observe that the above mechanism can be thought of as a combination of the migration scheme in the (fully-connected) parallel GAs, pair-wise tournament selection in the sGA, and an incremental learning policy in the cGA.

2.3 Harmonizing with Elitism

The previous scheme (described in Section 2.1 and Section 2.2) may not be able to solve difficult problems, especially those involving higher order BBs (e.g. deceptive problems) because linkage information may be lost due to independent operations on each gene. More difficult problems require a higher selection pressure for finding a better solution [4,5]. This is because a higher selection pressure offsets the disruptive effects (i.e., decorreation) of uniform crossover, thereby carrying partial knowledge (such as the linkage) about gene's correlation. Hence, it eventually leads to convergence to a better solution.

Although selection pressure can usually be increased by increasing tournament size, it requires additional (by no means significant) memory-costs and problem-dependent information that is not generally available in real-world problems. Even if such information is available, computation of the necessary tournament size that builds an adequate selection pressure is not easy. As a result, our intention is to adaptively alter selection pressure in response to the degree of difficulty of the problems without actually varying the tournament size. Fortunately, this is easily accomplished by putting elitism to work: simply retain the best solution found so far (see **Step 2** and **Step 3**). This is because inheriting

the best solution (i.e., elitism) can increase the selection pressure by preventing the loss of low "salience" genes of individuals due to deficient selection pressure, thereby improving the performance with regard to optimality and convergence in many cases [5].

Note that the pe-cGA is a special case of the proposed algorithm (the number of PVs is 2). This is because the pe-cGA reconciles the extant cGA with elitism in such a manner that keeps the best solution found so far.

2.4 Memory-Efficiency

Let n_s, n, v, and s be the population size (of sGA), virtual population size, the number of PVs and, tournament size, respectively. All the individuals have l genes. Since the sGA needs to store n_s bits for each gene position, it requires $l \cdot n_s$ bits [4]. The cGA and pe-cGA require $\lceil l \cdot log_2(n+1) \rceil$ bits because they only need to treat a finite set of $(n+1)$ numbers (that can be represented with $\lceil log_2(n+1) \rceil$ bits) for keeping the proportion of ones (or zeros) for each gene [4,5]. On the other hand, the proposed me^{2}GA needs $\lceil v \cdot l \cdot log_2(n+1) \rceil$ bits of memory since the proportion for each gene is distributed over all the v PVs. The me^{2}GA appears to be less memory-efficient than cGA and pe-cGA as it needs an additional $\lceil (v-1) \cdot l \cdot log_2(n+1) \rceil$ bits. Similarly, a reverse implication can be made for sGA. For fairness, however, we should compare the quality of solution returned by each algorithm incurring the same memory-cost. Of course, sGA and cGA should be appropriately tuned so as to achieve the best performance (for a given memory size) when they are applied to (especially) hard optimization problems.

Since sGA can realize multiple tournament size without compromising on memory-costs, the entire memory is allotted to its population. In cGA, the memory that remains after setting aside a portion that takes into account the virtual population of me^{2}GA is dedicated to the task of enhancing selection pressure. This is because cGA alone does not bring about such an enhancement by increasing the virtual population size. Therefore, me^{2}GA that needs $\lceil v \cdot l \cdot log_2(n+1) \rceil$ bits (i.e., the number of PVs is v, virtual population size is n, and $s = 2$) must be compared with sGA (with $n_s = \lceil v \cdot log_2(n+1) \rceil$ and $s \leq n_s$), and the cGA (with virtual population size n and $s = \lceil (v-1) \cdot log_2(n+1) \rceil$). This strategy will be adopted in performing a comparative study in Section 3.2.

3 Experimental Results and Discussion

In this section, the performance of me^{2}GA is investigated through computer simulations. All the results were averaged over 100 runs. Each experiment is terminated when all the PVs converge to a solution. Most algorithms achieve high performance on easy problems and their differences, if any, are insignificant [5]. Moreover, most real-world problems such as resource-allocation, multicast routing and (adaptive) equalizer design in fading channels, etc., may not be modeled as easy problems (it may depend on encoding method, though). Therefore, fully

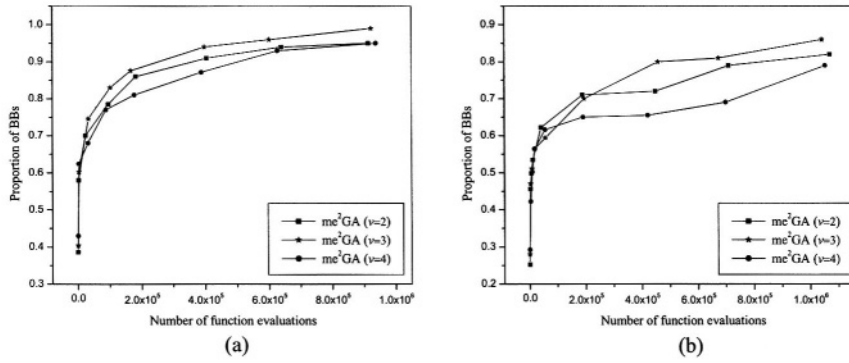


Fig. 2. Solution quality of the me^{2}GA with varying number of PVs. (a) Proportion of BBs versus function evaluations on 3-bit deceptive problem. (b) Proportion of BBs versus function evaluations on 4-bit deceptive problem.

deceptive problems [1,4,5] are considered for putting me^{2}GA to real test. The first (second) deceptive problem is formulated by concatenating 10 copies of the 3-bit (4-bit) trap function for a total individual length of 30 (40) bits. Both 3-bit and 4-bit trap functions have a deceptive-to-optimal ratio of 0.7.

3.1 Performance of Solution Quality

We take the proportion of BBs (i.e., fitness value) versus the number of function evaluations as the performance measure. It emphasizes solution quality (i.e., optimality). In the interest of fair comparison of the algorithms on the basis of optimality and convergence performance, we investigate solution quality obtained by each population size that performs the same number of function evaluations [2,5,6]. Although finding the exact population size for a particular execution for each GA is very difficult in practice, the work with certain constraints is relatively easy. We can determine the population size for each GA by exhaustive search so as to achieve comparable number of function evaluations. From this perspective, Fig. 2(a) and (b) compare proportion of BBs (according to the function evaluations) returned by the proposed algorithm with varying number of PVs as applied to 3-bit and 4-bit deceptive problems, respectively.

The figures show that the performance of me^{2}GA with $v = 3$ is generally higher than its performance with $v = 2$. However, the solution quality of me^{2}GA with $v = 4$ is not an improvement over its performance with $v = 2$ and $v = 3$. The reason why me^{2}GA with $v = 4$ degenerates the solution quality is that a frequent replacement of the reference (i.e., elite individual) prohibits the GA from steadily converging to a solution. It is thus seen that me^{2}GA with $v = 3$ achieves the best performance (i.e., the advisable number of PVs is 3). Note that the me^{2}GA with $v = 2$ falls within the purview of the pe-cGA as described in Section 2.3. In addition, the superiority of the pe-cGA to sGA and cGA

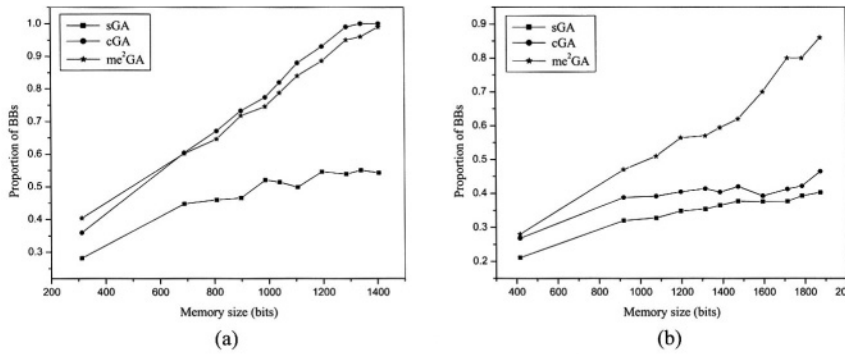


Fig. 3. Memory-efficiency of the algorithms. (a) Proportion of BBs versus memory size on 3-bit deceptive problem. (b) Proportion of BBs versus memory size on 4-bit deceptive problem.

for solving hard optimization problems under fair comparison has already been shown [5]. The conclusion is, therefore, inescapable that me^{2}GA ($v = 3$) is quite a promising candidate for solving especially difficult optimization problems. It is important to note that parallel implementation of the proposed algorithm is advisable as noted in Section 2.1. In the parallel scheme, the function evaluations per node decrease as the number of PVs increases. It gives the impression that the quality of solution improves as the number of PVs (v) increases, though the improvement rate may not be impressive.

3.2 Performance of Memory-Efficiency

Finding a better solution with lesser memory was one of the aims of this investigation. Memory-efficiency of me^{2}GA is examined in this experiment. The proposed algorithm employs three PVs (i.e., $v = 3$). The extant sGA and cGA are chosen as references. The sGA uses tournament selection without replacement and uniform crossover with exchange probability 0.5 [4,5]. The crossover is applied with probability '1.0' and mutation probability is set to '0.0'. On the other hand, sGA and cGA should be properly prepared for comparison purposes. The preparations follow the prescription that is given in Section 2.4. Here, sGA employs a tournament size of 8 (to obtain a high quality of solution) on the basis of exhaustive search. The proportion of BBs returned by each algorithm working with the same memory size is taken to be the performance measure.

Fig. 3(a) and (b) show the results (working with the same memory size) as applied to 3-bit and 4-bit deceptive problems, respectively. The quality of solution found by me^{2}GA is comparable to that exhibited by cGA and is in fact better than that exhibited by sGA on a 3-bit deceptive problem. Moreover, a significant performance difference is seen on a 4-bit deceptive problem. Note that the me^{2}GA performs better than the sGA and cGA as the target problem becomes

harder. It is interesting to note that the quality of solution found by me^{2}GA steadily increases with the size of memory used. Therefore, it is thought that the proposed algorithm is more memory-efficient than sGA and cGA (especially for solving difficult optimization problems).

The proposed algorithm has some strong points as is clear from the results. It seems to adaptively adjust its selection pressure according to the degree of difficulty of the problems so as to provide the selection pressure that is enough to steadily grow the correct BBs as the virtual population size increases. Therefore, it can effectively solve difficult problems (e.g., deceptive problems involving higher-order BBs) without any knowledge about problem dependent information such as the degree of deception (i.e. signal-to-noise ratio), the order of BBs, and the strength of collateral noise, and so on.

4 Conclusion

This paper has proposed a memory-efficient elitist genetic algorithm (me^{2}GA). The aim is to recover from decision errors and mitigate the disruptive effect of uniform crossover by combining multiple PVs with elitism in a proactive manner. It can efficiently and speedily solve many difficult optimization problems using a small amount of memory. It does not need any problem dependent information. The proposed algorithm with two PVs (i.e., $v = 2$) and without elitism is identical to the extant cGA; while, with elitism it is identical to pe-cGA.

Simulation studies show that, under fair comparison, the proposed algorithm operating with $v = 3$ attains the best performance with regard to quality of solution. Furthermore, memory-efficiency of the me^{2}GA is higher than that of sGA and cGA. That is, it returns the best quality of solution under identical memory size. Surprisingly, this memory-efficiency improves as the problem becomes harder and harder.

It is felt that the proposed algorithm will be useful in the memory- and time-constrained applications involving hard optimization problems.

References

1. D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms,* Kluwer Academic Publishers, 2002.
2. C. W. Ahn and R. S. Ramakrishna, "A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations," *IEEE Transactions on Evolutionary Computation,* vol.6, no.6, 2002.
3. S. Baluja, "Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," *Technical Report CMU-CS-94-163,* Carnegie Mellon University, Pittsburgh, PA, 1994.
4. G. Harik, F. G. Lobo, and D. E. Goldberg, "The Compact Genetic Algorithm," *IEEE Transactions on Evolutionary Computation,* vol.3, no.4, pp.287-297, 1999.
5. C. W. Ahn and R. S. Ramakrishna, "Elitism-based Compact Genetic Algorithms," *IEEE Transaction on Evolutionary Computation,* vol.7, no.4, pp.367-385, 2003.
6. E. Cantŭ-Paz, *Efficient and Accurate Parallel Genetic Algorithms,* Kluwer Academic Publishers, 2000.

Augmented Compact Genetic Algorithm

Chang Wook Ahn and R.S. Ramakrishna

Department of Information and Communications
Kwang-Ju Institute of Science and Technology (K-JIST)
Oryong-Dong, Puk-Gu, Gwangju 500-712, Korea
{cwan,rsr}@kjist.ac.kr
http://parallel.kjist.ac.kr/~cwan/

Abstract. An *augmented compact genetic algorithm* (acGA) is presented in this paper. It exhibits all the desirable characteristics of compact genetic algorithm (cGA). While the selection strategy of cGA is similar to (steady-state) tournament selection with replacement (TSR), the proposed algorithm employs a strategy akin to tournament selection without replacement (TS/R). The latter is a common feature of genetic algorithms (GAs) as it is perceived to be effective in keeping the selection noise as low as possible. The proposed algorithm stochastically maintains the progress of convergence even after the probability (distribution) vector (PV) begins transition towards one of the solutions. Experimental results show that the proposed algorithm converges to a similar solution at a faster rate than the cGA.

1 Introduction

Genetic algorithms (GAs) are stochastic search mechanisms[1]. Of all the issues connected with GAs - such as population size, genetic operators (e.g., selection, crossover, and mutation), and encoding methods, etc. - the population size that guarantees an optimal solution quickly enough has been a topic of intense research [2,3]. Goldberg *et al.* [2] developed the population-sizing equation that permits accurate statistical decision making among competing building blocks. Drawing on the decision model in [2], Harik *et al.* [3] exploited the similarity between the gambler's ruin problem and the selection mechanism (i.e., pair-wise tournament selection without replacement (TS/R)) of GAs for determining an adequate population size that guarantees a solution with desired quality. Following the model in [3], they proposed the compact GA (cGA) as a by-product by representing the population as a probability (distribution) vector (PV) over the set of solutions. The algorithm mimics the order-one behavior of simple GA (sGA) with uniform crossover using a small amount of memory [4]. However, the cGA employs a mechanism that is somewhat closer to tournament selection with replacement (TSR) [4] while the sGA uses TS/R. The sGA returns a better performance as TS/R has the potential to keep selection noise as low as possible [2,3]. Furthermore, the analytic model of Harik *et al.* [3] started from the assumption that the fitness values of a pair of individuals can be ordered. This effectively implies that the selection mechanism is TS/R.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 560–565, 2004.
© Springer-Verlag Berlin Heidelberg 2004

This paper proposes an *augmented compact genetic algorithm* (acGA). It mimics the TS/R even more strongly and also converges to a similar solution very quickly.

2 Proposed Algorithm: Augmented Compact GA

Harik *et al.* [4] introduced cGA for work in a binary (genotype) space $\Omega = \{0, 1\}^l$. The population of individuals in generation t is represented by a PV,

$$p_t(\mathbf{x}) = (p_t(x_1), p_t(x_2), \cdots, p_t(x_l)) \tag{1}$$

where $p_t(x_i)$ is the probability of observing a value of '1' in the ith gene.

The selection mechanism of cGA is analogous to (steady-state) TSR. The idea is to improve the convergence performance further by incorporating the TS/R selection mechanism into it. Fig. 1 presents the pseudocode of the acGA. Without **Step 3**, it is clearly seen to be cGA. The acGA is, therefore, found to exhibit all the characteristics of the cGA. In the early stages of PV's evolution, the cGA behaves as if it employed the TS/R mechanism. When the PV shows a tendency to converge to a solution, the probability that identical individuals are generated increases. It means that generations may pass without any evolution (i.e., update) of the PV. It is caused by the TSR-characteristic of cGA. Thus, **Step 3** is being added (by the authors) with a view to mimic TS/R. The process stochastically nominates one from among similar individuals as a competitor even if the PV tends to converge. It plays a role in imposing genetic diversity on the chromosomes. However, the effect is too weak to effectively control PV's march towards convergence. Instead, it usually results in enhanced convergence speed (with possibly a slight improvement in the quality of solutions).

Since the parameter θ (lies between 0.0 and 1.0) determines the degree of change (in non-convergent genes of one of identical individuals), the performance depends on its value.

Three special cases may be considered: 1) $\theta = 1.0$, 2) $\theta = 0.5$ and 3) $\theta = 0.0$. When $\theta = 1.0$, the acGA becomes the cGA because no gene may be flipped over toward its opposite allele (from 0(1) to 1(0)). If $\theta = 0.0$, all the genes corresponding to non-convergent sites in the PV are inverted because all the generated random numbers are always greater than or equal to θ. It assigns another fitness value to one of the two (identical) individuals (by introducing a new individual). But the support (i.e., a decoding value of the individual) of the fitness function may be located far away from that of the current solution. Furthermore, it possibly moves the support in a direction opposite to the direction of evolution of the current solution. Even though it obviously enhances convergence of the PV in the case of simple problems, it carries the potential to induce oscillations of the PV in problems with multiple local optima. If $\theta = 0.5$, it imparts the ability to effectively search the solution space because the genes corresponding to non-convergent sites (of the PV) are flipped over toward their opposite alleles in a totally random fashion. It amounts to saying that the direction and the distance of support (of the new individual) are uniformly distributed between

Parameters. n : virtual population size, l : individual length
Step 1. Initialize probability vector
$t \leftarrow 0$;
for $i \leftarrow 1$ **to** l **do**
$p_t(x_i) \leftarrow 0.5$;
Step 2. Generate two individuals for the probability vector
$\mathbf{a} \leftarrow$ **generate** $(p_t(\mathbf{x}))$; $\mathbf{b} \leftarrow$ **generate** $(p_t(\mathbf{x}))$;
Step 3. Stochastically changes non-convergent genes of one of the identical individuals (under a convergence tolerance of ε)
for $i \leftarrow 1$ **to** l **do**
if $\neg(\mathbf{a}[i] \oplus \mathbf{b}[i])$ **then**
if $p_t(x_i) > 0.0 + \varepsilon \wedge p_t(x_i) < 1.0 - \varepsilon$ **then**
if **random**(0,1) $\geq \theta$ **then**
$\mathbf{a}[i] \leftarrow \neg\mathbf{a}[i]$;
Step 4. Let them compete
$winner, loser \leftarrow$ **compete** $(\mathbf{a}, \mathbf{b})$;
Step 5. Update the probability vector
for $i \leftarrow 0$ **to** $l-1$ **do**
if $winner[i] \neq loser[i]$ **then**
if $winner[i] == 1$ **then**
$p_t(x_i) \leftarrow p_t(x_i) + 1/n$;
else $p_t(x_i) \leftarrow p_t(x_i) - 1/n$;
Step 6. Check if the probability vector has converged (with ε)
for $i \leftarrow 1$ **to** l **do**
if $p_t(x_i) > 0.0 + \varepsilon \wedge p_t(x_i) < 1.0 - \varepsilon$ **then**
$t \leftarrow t + 1$; return to **Step 2**;
Step 7. The probability vector $p_t(\mathbf{x})$ represents the final solution

Fig. 1. Pseudocode of the acGA.

zero and the maximum available value. Therefore, it can improve the quality of solution to difficult problems (especially as elitism is employed). It also improves the convergence speed considerably in every situation.

3 Experimental Results and Discussion

In this section, the performance of acGA is investigated through computer simulations. Binary encoding scheme is employed for numerical optimization. In order to highlight the effects of reinforcement, cGA is employed as a reference. Fitness value (e.g., the number of correct BBs or objective function value) and the number of function evaluations are taken as performance measures. The former considers solution quality (i.e., optimality) and the latter indicates the convergence performance. All the results were averaged over 200 runs.

3.1 Results for Simple Problems

A minimum deceptive problem (MDP) (formed by concatenating 10 copies of minimum deceptive function) [4] is considered for evaluating the proposed al-

gorithm on problems involving distinct order (especially, lower-order) BBs. Fig. 2 presents the number of correct BBs and the number of function evaluations returned by each algorithm as applied to MDP with a deceptive-to-optimal ratio of 0.7. The convergence speed of acGA with $\theta = 0.5$ is far higher than that of cGA (i.e., about 25% improvement) while the algorithm maintains the quality of solution at a level comparable with that of cGA (i.e., acGA with $\theta = 1.0$). The acGA with $\theta = 0.0$ can further improve the speed of convergence and the quality of solution a little. However, acGA may not achieve a better solution than that of sGA because it lacks the memory to retain the knowledge about the non-linearity of the problem. Note that this is an inherent characteristic of compact-type GAs. Although there have been several studies on this inherent disadvantage, the issue is beyond the scope of this paper.

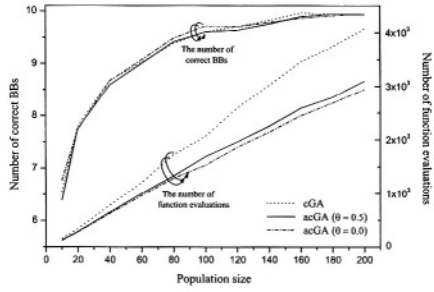


Fig. 2. Performance of the algorithms on the MDP.

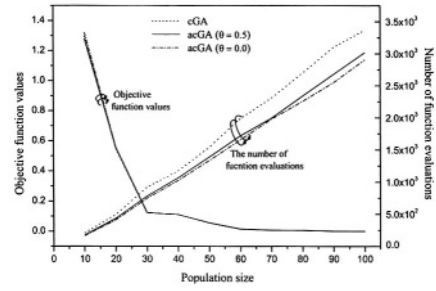


Fig. 3. Performance of the algorithms on the DeJong's test function f_1.

DeJong's test function f_1 [4], presented in (2) below, is employed for studying the performance of acGA on continuous-unimodal problems. The function models simple problems such as the equalization (in flat fading channels) and the noise reduction problems. The problem is to

$$\text{minimize} \quad f_1(x) = \sum_{i=1}^{n} x_i^2, \; x_i \in [-5.12, 5.11], \; n = 3 \tag{2}$$

Fig. 3 compares the objective function value and the number of function evaluations of the algorithms. The acGA ($\theta = 0.5$ and $\theta = 0.0$) converges to a solution that is similar in quality to that of cGA with 10~15% improvement with regard to convergence speed over the cGA.

From Figs. 2 and 3, it is noted that the mechanism with $\theta = 0.0$ does not induce any undesirable effect in the case of simple problems. Instead, it assists the PV in rapidly converging to a better solution.

3.2 Results for Difficult Problems

Most real-world problems may not be modeled by a continuous-unimodal function or by concatenating distinct order-BBs in a simple manner since their solution/search spaces are continuous and multimodal in general. In order to investigate the performance on such problems, DeJong's test function f_5 [4] and Schaffer's binary function f_{S_6} [5] are employed. The functions may be used for modeling various real-world problems, especially those arising in the emerging areas of wireless networks (such as dynamic power allocation and fading channel estimation problems).

DeJong's test function f_5 is investigated first. The problem is to

minimize

$$f_5(x) = \left[0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{n}(x_i - a_{ij})^6}\right]^{-1}, \quad x_i \in [-65.536, 65.535], n = 2,$$

$$[(a_{1j}, a_{2j})]_{j=1}^{25} = \begin{bmatrix} (-32,-32),(-16,-32),(0,-32),(16,-32),(32,-32), \\ (-32,-16),(-16,-16),(0,-16),(16,-16),(32,-16), \\ \cdots,(-32,32),(-16,32),(0,32),(16,32),(32,32) \end{bmatrix} \quad (3)$$

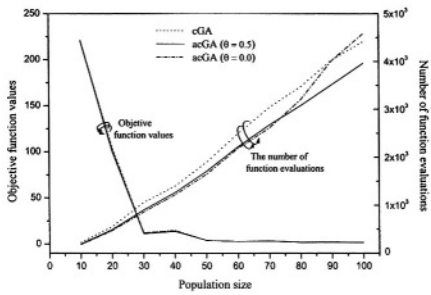


Fig. 4. Performance of the algorithms on the DeJong's test function f_5.

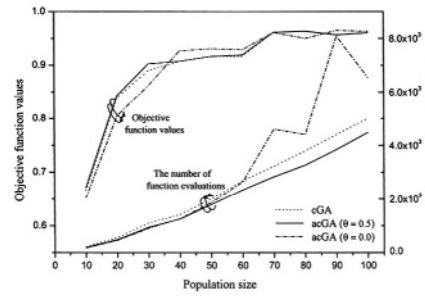


Fig. 5. Performance of the algorithms on the Schaffer's binary function f_{S_6}.

Fig. 4 compares the algorithms as applied to DeJong's test function f_5. The acGA with $\theta = 0.5$ attains about 10% convergence performance improvement over cGA without compromising the quality of solution. The convergence performance of the acGA with $\theta = 0.0$ worsens (after a population size of 70) as compared with acGA with $\theta = 0.5$ as the population size increases. Moreover, the convergence performance is the worst after a population size 90.

Schaffer's binary function f_{S_6}, presented in (4), is considered next. The problem is to

$$\text{maximize} \quad f_{S_6}(x) = \frac{sin^2\left(\sqrt{\sum_{i=1}^{n} x_i^2}\right)}{1.0 + 10^{-3} \cdot \left(\sum_{i=1}^{n} x_i^2\right)^2}, \; x_i \in [-16.383, 16.384], \; n = 2 \tag{4}$$

The function is degenerate in the sense that many points share the same global optimal function value [5]. Fig. 5 compares the performance returned by each algorithm. The acGA with $\theta = 0.5$ generally exhibits the highest convergence speed while the solution itself lies in the vicinity of that of the cGA. The convergence performance improvement amounts to about 12% over cGA. The number of function evaluations of the acGA with $\theta = 0.0$ fluctuates as the population size increases. To achieve a solution with a comparable quality, the acGA with $\theta = 0.0$ increases (i.e., sacrifices) the number of function evaluations. It implies that the acGA with $\theta = 0.0$ imposes genetic diversity at an excessively biased level.

From Figs. 4 and 5, it is seen that the undesirable effect of θ being zero is proportional to the population size and the degree of difficulty of the problem. As a consequence, the acGA with $\theta = 0.5$ is considered to be the most stable and feasible algorithm with regard to both solution quality and convergence speed.

4 Conclusion

The proposed algorithm (acGA) is a modified version of cGA. It carries all the benefits of the cGA and incorporates TS/R by stochastically altering non-convergent genes of one of the (identical) individuals. The result is speedy convergence with a slightly improved quality of solution (for all types of problem) provided that the θ-value is adequate (i.e., $\theta = 0.5$). The algorithm may be useful in memory-constrained applications such as multicast routing and resource allocation problems in the emerging field of wireless networks.

References

1. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.
2. D. E. Goldberg, K. Deb, and J. H. Clark, "Genetic Algorithms, Noise, and the Sizing of Populations," *Complex Systems,* vol.6, pp.333-362, 1992.
3. G. Harik, E. Cantŭ-Paz, D. E. Goldberg, and B. L. Miller, "The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations," *Evolutionary Computation,* vol.7, no.3, pp.231-253, 1999.
4. G. Harik, F. G. Lobo, and D. E. Goldberg, "The Compact Genetic Algorithm," *IEEE Transactions on Evolutionary Computation,* vol.3, no.4, pp.287-297, 1999.
5. J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization," *Proceedings of the 3rd International Conference on Genetic Algorithms,* pp.51-59, CA: Morgan Kaufmann, 1989.

Parallel Genetic Algorithm for the Flow Shop Scheduling Problem

Wojciech Bożejko[1] and Mieczysław Wodecki[2]

[1] Institute of Engineering, Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wbo@ict.pwr.wroc.pl
[2] Institute of Computer Science, University of Wrocław
Przesmyckiego 20, 51-151 Wrocław, Poland
mwd@ii.uni.wroc.pl

Abstract. The permutation flow shop sequencing problem with the objective of minimizing the sum of the job's completion times, in literature known as the $F||C_{sum}$, has been considered. The parallel genetic algorithm based on the island model of migration has been presented. By computer simulations on Taillard benchmarks [10] and the best known results from literature [9] we have obtained not only acceleration of the computation's time but also better quality and stability of the results.[1]

1 Introduction

We take under consideration the permutation flow shop scheduling problem described as follows. A number of jobs are to be processed on a number of machines. Each job must go through all the machines in exactly the same order and the job order is the same on every machine. Each machine can process at most one job at any point of time and each job may be processed on at most one machine at any time. The objective is to find a schedule that minimizes the sum of the job's completion times. The problem is indicated by the $F||C_{sum}$.

There are plenty of good heuristic algorithms for solving the $F||C_{max}$ flow shop problem, with the objective of minimizing the maximal job's completion times. For the sake of special properties (blocks of critical path, [5]) it is recognized as an easier one than the problem with the objective C_{sum}. Unfortunately, there are no similar properties (which can speedup computations) for the $F||C_{sum}$ flow shop problem. Constructive algorithms (LIT and SPD from [11]) have low efficiency and can only be applied in a limited range. There is a hybrid algorithm in [9], consisting of elements of tabu search, simulated annealing and path relinking methods. The results of this algorithm, applied to the Taillard benchmark tests [10], are the best known ones in literature nowadays. The big disadvantage of the algorithm is its time-consumption. Parallel computing is the way to speed it up.

This work is the continuation of the author's research on constructing efficient parallel algorithms to solve hard combinatorial problems ([1,2,12]). Further, we

[1] The work was supported by KBN Poland, within the grant No. T11A01624

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 566–571, 2004.
© Springer-Verlag Berlin Heidelberg 2004

present a parallel algorithm based on the genetic algorithm method which not only speeds up the computations but also improves the quality of the results.

2 Problem Definition and Notation

The flow shop problem can be defined as follows, using the notation of Nowicki, Smutnicki [7] and Grabowki, Pempera [5]. There are a set of n jobs $J=\{1,2,\ldots,n\}$ and a set of m machines $M=\{1,2,\ldots,m\}$. Job $j \in J$ consists of a sequence of m operations $O_{j1}, O_{j2},\ldots,O_{jm}$. Operation O_{jk} corresponds to the processing of job j on machine k during an uninterrupted processing time p_{jk}. We want to find a schedule so that the sum of the job's completion times is minimal.

Let $\pi = (\pi(1), \pi(1),\ldots,\pi(n))$ be a permutation of jobs $\{1,2,\ldots,n\}$ and let Π be the set of all permutations. Each permutation $\pi \in \Pi$ defines a processing order of jobs on each machine. We wish to find a permutation $\pi^* \in \Pi$ that $C_{sum}(\pi^*) = \min_{\pi \in \Pi} C_{sum}(\pi)$, where $C_{sum}(\pi) = \sum_{i=1}^{n} C_{i,m}(\pi)$, and $C_{i,j}(\pi)$ is the time required to complete job i on the machine j in the processing order given by the permutation π. Such a problem belongs to the strongly NP-hard class.

3 Genetic Algorithm

The genetic algorithm is a search procedure, based on the process of natural evolution, following the principles of natural selection, crossover and survival. The method has been proposed and developed by Holland [6]. In the beginning, a population of individuals (solutions of the problem, for example permutations) is created. Each individual is evaluated according to the fitness function (in our problem this is the $C_{sum}(\pi)$ value). Individuals with higher evaluations (more fitted, with a smaller $C_{sum}(\pi)$ value) are selected to generate a new generation of this population. So there are three essential steps of the genetic algorithm: (1) selection – choosing some subset of individuals, so-called parents, (2) crossover – combining parts from pairs of parents to generate new ones, (3) mutation – transformation that creates a new individual by small changes applied to an existing one taken from the population.

New individuals created by crossover or mutation replace all or a part of the old population. The process of evaluating fitness and creating a new population generation is repeated until a termination criterion is achieved.

Let P_0 be an initial population, k – number of iteration of the algorithm, P – population. Let P' be a set of parents – subset of the most fitted individuals of the population P. By the mechanism of crossover, the algorithm generates a set of offsprings P'' from set P'. Next, some of the individuals from the set P'' are mutated. The algorithm stops after a fixed number of iterations. The complexity of the algorithm depends on the number of iterations and the size of the population.

4 Parallel Genetic Algorithm

There are three basic types of parallelization strategies which can be applied to the genetic algorithm: global, diffusion model and island model (migration model).

Algorithms based on the island model divide the population into a few subpopulations. Each of them is assigned to a different processor which performs a sequential genetic algorithm based on its own subpopulation. The crossover involves only individuals within the same population. Occasionally, the processor exchanges individuals through a migration operator. The main determinants of this model are: (1) size of the subpopulations, (2) topology of the connection network, (3) number of individuals to be exchanged, (4) frequency of exchanging.

The island model is characterized by a significant reduction of the communication time, compared to previous models. Shared memory is not required, so this model is more flexible too. Bubak and Sowa [3] developed an implementation of the parallel genetic algorithm for the TSP problem using the island model.

Below, a parallel genetic algorithm is proposed. The algorithm is based on the island model of parallelizm. Additionally, there is the MSXF (Multi – Step Crossover Fusion) operator used to extend the process of researching for better solutions of the problem. MSXF has been described by Reeves and Yamada [9]. Its idea is based on local search, starting from one of the parent solutions, to find a new good solution where the other parent is used as a reference point.

The neighbourhood $N(\pi)$ of the permutation (individual) π is defined as a set of new permutations that can be reached from π by exactly one adjacent pairwise exchange operator which exchanges the positions of two adjacent jobs of a problem's solution connected with permutation π. The distance measure $d(\pi,\sigma)$ is defined as a number of adjacent pairwise exchanges needed to transform permutation π into permutation σ. Such a measure is known as Kendall's τ measure.

Algorithm 1. Multi-Step Crossover Fusion (MSXF), [9]

Let π_1, π_2 be parent solutions. Set $x = q = \pi_1$;
repeat
  For each member $y_i \in N(\pi)$, calculate $d(y_i, \pi_2)$;
  Sort $y_i \in N(\pi)$ in ascending order of $d(y_i, \pi_2)$;
  repeat
    Select y_i from $N(\pi)$ with a probability inversely proportional to the index i; Calculate $C_{sum}(y_i)$;
    Accept $y_i$with probability 1 if $C_{sum}(y_i) \leq C_{sum}(x)$, and with probability $P_T(y_i) = exp((C_{sum}(x) - C_{sum}(y_i)) / T)$ otherwise (T is temperature);
    Change the index of y_i from i to n and the indices of y_k, $k = i+1,...,n$ from k to $k-1$;
  until y_i is accepted;
  $x \leftarrow y_i$; **if** $C_{sum}(x) < C_{sum}(q)$ **then** $q \leftarrow x$;
until *some termination condition is satisfied*;
q is the offspring.

In our implementation, MSXF is an inter-subpopulations crossover operator which constructs a new individual using the best individuals of different subpopulations connected with different processors. The condition of termination consisted in exceeding of 100 iterations by the MSXF function.

Algorithm 2. Parallel genetic algorithm

parfor $j = 1, 2, ..., p$ { *p is number of processors* }
  $i \leftarrow 0$; $P_j \leftarrow$ random subpopulation connected with processor j;
  $p_j \leftarrow$ number of individuals in j subpopulation;
  repeat
    Selection(P_j, P'_j); Crossover(P'_j, P''_j); Mutation(P''_j);
    if (k mod $R = 0$) **then** {*every R iteration*}
      $r := random(1, p)$; MSXF($P'_j(1), P_r(1)$);
    end if;
    $P_j \leftarrow P''_j$; $i \leftarrow i + 1$;
    if *there is no improvement of the average* C_{sum} **then** {*Partial restart*}
      $r := random(1,p)$;
      Remove $\alpha = 90$ percentage of individuals in subpopulation P_j;
      Replenish P_j by random individuals;
    end if;
    if (k mod $S = 0$) **then** {*Migration*}
      $r := random(1,p)$;
      Remove $\beta = 20$ percentage of individuals in subpopulation P_j;
      Replenish P_j by the best individuals from subpopulation P_r
      taken from processor r;
    end if;
  until *Stop_Condition*;
end parfor

The frequency of communication between processors (migration and MSXF operator) is very important for the parallel algorithm performance. It must not be too frequent (long time of communication between processors!). In this implementation the processor gets new individuals quite rarely, every $R = 20$ (MSXF operator) or every $S = 35$ (migration) iterations.

5 Computer Simulations

The algorithm was implemented in the Ada95 language and run on 4-processors Sun Enterprise 4x400 MHz under the Solaris 7 operating system. Tasks of the Ada95 language were executed in parallel as system threads. Tests were based on 50 instances with 100,... ,500 operations ($n \times m$=20×5, 20×10, 20×20, 50×5, 50×10) due to Taillard [10], taken from the OR-Library [8]. The results were compared to the best known, taken from [9]. Every instance of the test problems was executed six times, and the average result was used for comparing. The standard deviation of results was computed too, as a measure of algorithm stability.

Firstly, we made tests of the classical genetic operators efficiency (seek Goldberg [4]) for our flow shop problem on the sequential genetic algorithm. Next, we chose the PMX, CX and SX crossover operator and the I mutation operator (random adjacent pairwise exchange) for further research. After choosing the operators, we implemented the parallel genetic algorithm. The chosen model of parallel computing was the MIMD machine of processors without shared memory – with the time of communication between processors much longer then the time of communication inside the process which is executing on one processor. The implementation was based on the island model of the parallel genetic algorithm with one central processor and slave processors. The central processor mediated in communication and stored data of the best individuals. Slave processors executed their own genetic algorithms based on subpopulations of the main population. Co-operation was based on migration between 'islands' and execution of the MSXF operator with parents taken from the best individuals of different subpopulations (processors).

We tested the efficiency of the parallel algorithm which was activated with combination of three strategies: with the same or different start subpopulations, as independent or cooperative search threads and with the same or different genetic operators. The number of iterations was permanently set to 1000. Results of tests for different start subpopulations for every processor are shown in Table 1. The results of the computations for the same start subpopulations strategy were similar, but slightly worse.

Table 1. Different start subpopulations, various genetic operators

$n \times m$	1 processor	4 processors			
		independent		cooperation	
		the same op.	different op.	the same op.	different op.
20x5	1,00%	0,81%	0,73%	0,66%	0,52%
20x10	1,10%	1,00%	0,97%	0,81%	0,79%
20x20	0,93%	0,75%	0,74%	0,65%	0,64%
50x5	2,96%	3,70%	3,44%	3,43%	3,10%
50x10	4,48%	4,97%	4,70%	4,79%	4,64%
average	**2,13%**	**2,25%**	**2,11%**	**2,07%**	**1,98%**
std.dev.	0,20%	0,15%	0,12%	0,16%	0,12%

As it turned out, the strategy of starting the computation from different subpopulations on every processor with different crossover operators and cooperation, was significantly better than others. The improvement of the distance to reference solutions was at the level of 7%, comparing to the sequential algorithm, with the same number of iterations equal to 1000 for the sequential algorithm and 250 for the 4-processors parallel algorithm. The time of the computing amount of a few seconds up to a few dozen seconds, depends on the size of the problem instance. Moreover, the parallel algorithm has more stability results – standard deviation of the results was on average equal to 0.12% for the best parallel algorithm, compared to 0.20% for the sequential algorithm – so the

improvement of the standard deviation was at the level of 40% with relation to the sequential algorithm.

6 Conclusions

We have discussed a new approach to the permutation flow shop scheduling based on the parallel asynchronous genetic algorithm. The advantage is especially visible for large problems. As compared to the sequential algorithm, parallelization increases the quality of solutions obtained. The idea of the best individual migration and the inter-subpopulation operator was used. Computer experiments show, that the parallel algorithm is considerably more efficient with relation to sequential algorithm. Results of tests (after a small number of iterations) are insignificantly different from the best known. In future work, we wish to add to the algorithm more elements of coevolutionary schemas, e.g. predators (predator-prey model), food, etc., and use environments more suitable for distributed computing (PVM, MPI), which will cause further improvement of the parallel algorithm efficiency.

References

1. Bożejko W., Wodecki M., Solving the flow shop problem by parallel tabu search, IEEE Computer Society, PR01730 ISBN 0-7695-1730-7, (2002), 189-194.
2. Bożejko W., Wodecki M., Parallel algorithm for some single machine scheduling problems, Automatics vol. 134, (2002), 81-90.
3. Bubak M., Sowa M., Object-oriented implementation of parallel genetic algorithms, in High Performance Cluster Computing: Programming and Applications (R. Buyya, ed.), vol. 2, Prentice Hall, (1999), 331-349.
4. Goldberg D., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc., Massachusetts, 1989.
5. Grabowski J., Pempera J., New block properties for the permutation flow-shop problem with application in TS, Jour. of Oper. Res. Soc. 52, (2001), 210-220.
6. Holland J.H., Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, University of Michigan Press, 1975.
7. Nowicki E., Smutnicki C., A fast tabu search algorithm for the permutation flow shop problem, EJOR 91 (1996), 160-175.
8. OR-Library: http://mscmga.ms.ic.ac.uk/info.html
9. Reeves C. R., Yamada T., Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search, IEEE International Conference on Evolutionary Computation, (1998), 230-234.
10. Taillard E., Benchmarks for basic scheduling problems, EJOR 64, (1993), 278-285.
11. Wang C., Chu C., Proth J., Heuristic approaches for $n/m/F/\Sigma C_i$ scheduling problems, EJOR (1997), 636-644.
12. Wodecki M., Bożejko W., Solving the flow shop problem by parallel simulated annealing, LNCS No. 2328, Springer Verlag, (2002), 236-247.

Optimization of Structures Using Distributed and Parallel Evolutionary Algorithms

Tadeusz Burczynski[1,2] and Waclaw Kus[1]

[1] Department for Strength of Materials and Computational Mechanics, Silesian University of Technology, Konarskiego 18a, 44-100 Glwice, Poland
{burczyns, wacok}@polsl.gliwice.pl,
[2] Institute for Computer Modelling, Cracow University of Technology Cracow, Poland

Abstract. This paper is devoted to applications of evolutionary algorithms into optimal design of nonlinear structures and identification of holes. The parallel and the distributed evolutionary algorithms are considered. The optimum criterion is to minimize the plastic strain areas and stress values or an identification functional. The fitness functions are computed using the finite element method or the coupled finite and boundary element method.

1 Introduction

The shape optimization problem of elasto-plastic structures can be solved using methods based on sensitivity analysis information or non gradient methods based on genetic algorithms [4,5,7,8]. This paper is devoted to method based on parallel and distributed evolutionary algorithms. Applications of evolutionary algorithms in optimization need only information about values of an objective (fitness) function. The fitness function is calculated for each chromosome in each generation by solving the boundary - value problem of elasto-plasticity by means of the finite element method (FEM) or the coupled finite and boundary element method (FEM-BEM). This approach does not need information about the gradient of the fitness function and gives the great probability of finding the global optimum. The main drawback of this approach is the long time of calculations. The applications of the parallel and the distributed evolutionary algorithms can shorten the time of calculations but additional requirements are needed: a multiprocessor computer or cluster of computers are necessary.

2 Formulation of the Evolutionary Optimization

A body which occupies the domain Ω bounded by the boundary $\Gamma = \Gamma_u \cap \Gamma_p$ is considered (Fig.1).

The body is made from an elasto-plastic material with hardening. Boundary conditions in the form of displacements u^0 and tractions p^0 are prescribed and body forces b are given. One should find the optimal shape of the body to

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 572–579, 2004.
© Springer-Verlag Berlin Heidelberg 2004

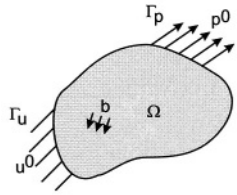


Fig. 1. Considered body

minimize areas of the plastic strains in the domain Ω. This optimization criterion can be achieved by minimizing a fitness function:

$$F = \int_{\Omega} \left(\frac{\sigma_a}{\sigma_0} \right) d\Omega \quad where \quad \sigma_a = \begin{cases} \sigma_{eq} & \text{when} \quad \sigma_{eq} \geq \sigma_{\mathrm{p}} \\ 0 & \text{when} \quad \sigma_{eq} < \sigma_{\mathrm{p}} \end{cases} \tag{1}$$

where σ_{eq} means the Huber - von Mises equivalent stress, σ_p is the yield stress and σ_0 is the reference stress. The identification problem is also considered. The identification of a circular hole inside the structure can be performed as a optimization problem using a fitness function:

$$F = \sum_{i=1}^{n} |u_i - \hat{u}_i| \tag{2}$$

where: $\hat{u}_i$ – are measured displacements in sensor points of the structure, u_i– are computed displacements, n – is the number of sensor points. The chromosome defines radius and position of the centre of the circular hole. Constrains on design variables are imposed. The structure was meshed using triangle finite elements (FEM) or finite elements and boundary elements (coupled FEM-BEM). The Triangle code [10] is used for body meshing.

3 Distributed and Parallel Evolutionary Algorithms

Sequential genetic and evolutionary algorithms are well known and applied in many areas of optimization problems [2,9]. The main disadvantage of these algorithms is the long time needed for computation. The parallel evolutionary algorithms [6] perform an evolutionary process in the same manner as the sequential evolutionary algorithm. The difference is in a fitness function evaluation. The parallel evolutionary algorithm evaluates fitness function values in the parallel way. Theoretically, maximum reduction of time needed to solve the optimization problem using parallel evolutionary algorithms is equal to number of used processing units. The maximum number of processing units which can be used is constrained by a number of chromosomes in the population. The flowchart of the parallel evolutionary algorithm is shown in Fig. 2. The starting population of chromosomes is created randomly. The evolutionary operators change chromosomes and the fitness function value for each chromosome is computed. The

server/master transfers chromosomes to clients/workers. The workers compute the fitness function and send it to server. The workers operate on different processing units. The selection is performed after computing the fitness function value for each chromosome. The selection decides which chromosomes will be in the new population. The selection is done randomly, but the fitter chromosomes have bigger probability to be in the new population. The next iteration is performed if the stop condition is not fulfilled. The stop condition can be expressed as a maximum number of iterations.

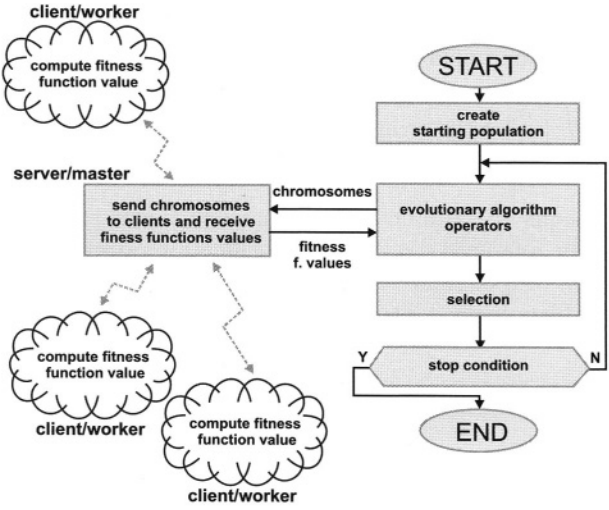


Fig. 2. Parallel evolutionary algorithm

The distributed genetic algorithms [1,11] and the distributed evolutionary algorithms (DEA) work similarly to many evolutionary algorithms operating on subpopulations. The evolutionary algorithms exchange chromosomes during a migration phase between subpopulations. When DEA is used the number of fitness function evaluations can be lower in comparison with sequential and parallel evolutionary algorithms. DEA works in the parallel manner, usually. Each of the evolutionary algorithms in DEA work on a different processing unit. The theoretical reduction of time could be bigger then the number of processing units. The starting subpopulation of chromosomes is created randomly. The evolutionary operators change chromosomes and the fitness function value for each chromosome is computed. The migration exchanges a part of chromosomes between subpopulations. The selection decides which chromosomes will be in the new population. The selection is done randomly, but the fitter chromosomes have bigger probability to be in the new population. The selection is performed on chromosomes changed by operators and immigrants. The next iteration is performed if the stop condition is not fulfilled. The stop condition can be expressed as a maximum number of iterations.

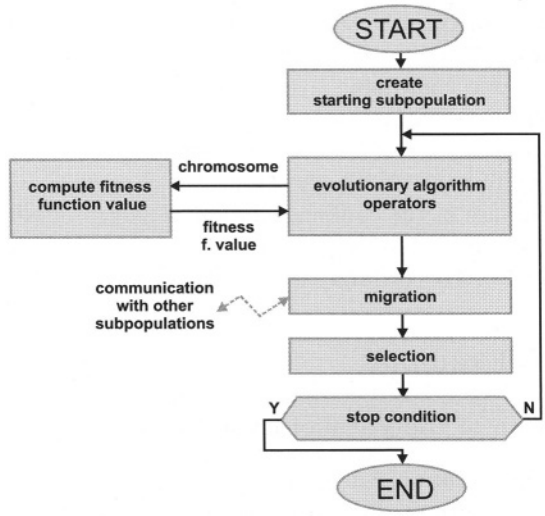


Fig. 3. The distributed evolutionary algorithm (one subpopulation)

To improve scalability of the distributed evolutionary algorithm, mechanisms from the parallel evolutionary algorithm can be used. The simplest improvement is computing fitness function values in a parallel way. The maximum number of processing units which can be used is equal to a sum of chromosomes in subpopulations instead of the number of subpopulations. The flowchart of the modified distributed evolutionary algorithm is presented in Fig. 4.

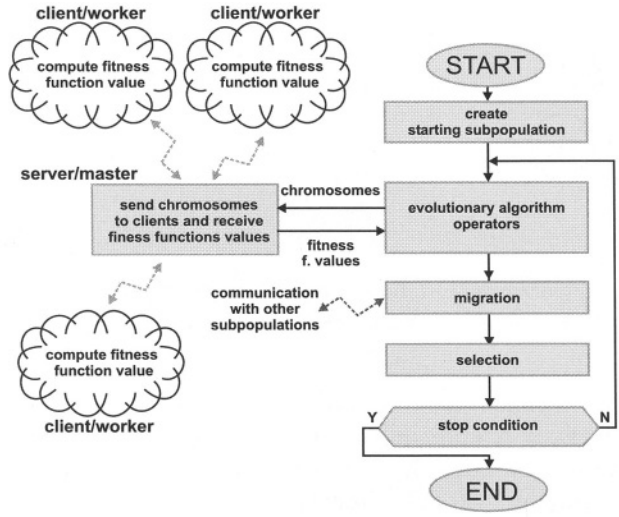


Fig. 4. Improved distributed evolutionary algorithm

The evolutionary operators used in presented algorithms are a crossover and a Gaussian mutation. The crossover chooses randomly two parent chromosomes and creates a new one containing a part of genes form first and a part from second parent. The Gaussian mutation creates chromosome based on randomly chosen one. Values of a part of the genes in the new chromosome are changing by adding random numbers with the Gaussian distribution. The selection is performed with the use of the ranking method. The probability of being in the new population does not depend on the fitness function value, but on the number of chromosomes ordered accordingly to the fitness function values.

4 Numerical Examples

4.1 The Speedup of the Parallel Evolutionary Algorithm

The speedup of computation can be expressed as a time needed to solve the problem on 1 processing unit divided by a time on n-processing units:

$$s = \frac{t_1}{t_n} \tag{3}$$

The optimization problem for an infinite body with holes is considered (Fig 5a). The number of processing units vary from 1 to 4. Two computers with two SMP (Symmetric MultiProcessing) units are used. The problem is to minimize plastic strain areas using the fitness function (1). The body is discretized using boundary elements and finite elements (Fig 5b).

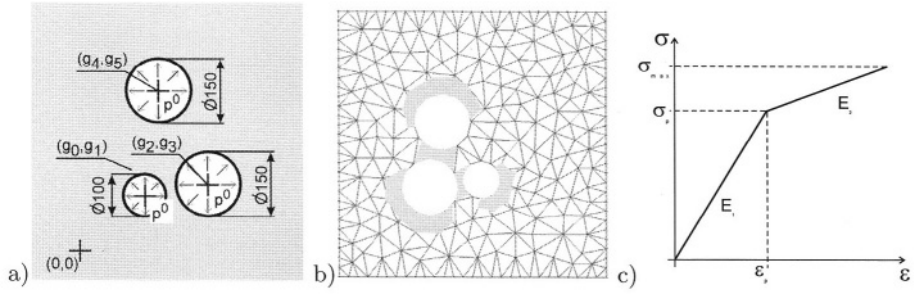


Fig. 5. a) geometry of an infinite body, b) best result after 50 generations, c)uniaxial stress-strain curve for material used in tests

The material of the body is elasto-plastic with hardening (Fig 5c.). Material parameters are: $E_1 = 2 * 10^5$, $\sigma_{p^\cdot} = 300MPa$, $E_2 = 1 * 10^5$, $\nu = 0.3$. Parameters of the parallel evolutionary algorithm are: number of chromosomes - 20, number of generations - 50. The best result after 50 generations is presented in Fig. 5b. Actual and linear speedups of the parallel evolutionary algorithm is show in Fig. 6. The fitness function evaluation of one chromosome takes about

2s. The speedup depends on a fitness function evaluation time and speedup is restricted by the ratio between the fitness function evaluation time and a time of communication between processors.

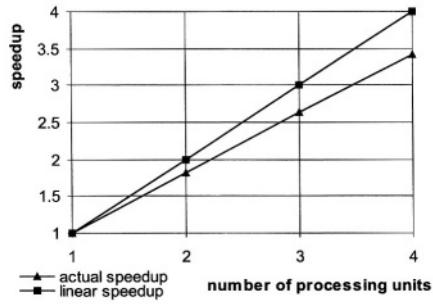


Fig. 6. Speedup of the parallel evolutionary algorithm

4.2 The Influence of Subpopulations Number on the Distributed Evolutionary Algorithm

The influence of a number of subpopulations on the distributed evolutionary algorithm for the identification problem is considered. The plate with a circular hole is considered. The position of the center of the hole and radius are searched. The plate is made from the elastic-plastic material with hardening (Fig. 5c). Material parameters are: $E_1 = 2 * 10^5$, $\sigma_p = 300MPa$, $E_2 = 1 * 10^5$, $\nu = 0.3$. The geometry is presented in Fig. 7a. The plate is meshed using finite elements. The identification is performed using information about displacements in 12 sensor points (Fig. 7b). The distributed evolutionary algorithm with 1 to 10 subpopulations was tested. The starting population in every test is the same and contains chromosomes with genes (35,35,10). The number of chromosomes in every subpopulation were 10.

The 10 tests were performed for 1,2,6,10 subpopulations. The minimum, maximum and average number of fitness function evaluations in tests are presented in Fig. 8. It is seen that optimal number of subpopulations is 2.

4.3 The Test of Speedup for the Improved Distributed Evolutionary Algorithm

The test problem is identification of the circular hole in the plate presented in 4.2. The results from the previous test show that the optimal number of the subpopulations is 2. The speedup was evaluated for different number of processors (1-6). The hardware for tests was the same as for tests 4.1. The actual and linear speedup are shown in Fig. 9. The fitness function evaluation of one chromosome

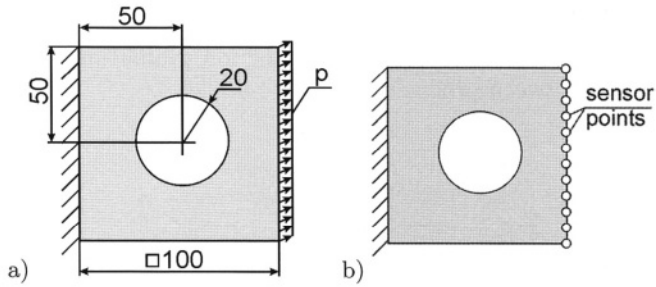


Fig. 7. a) Geometry of plate with a hole, b) sensor points

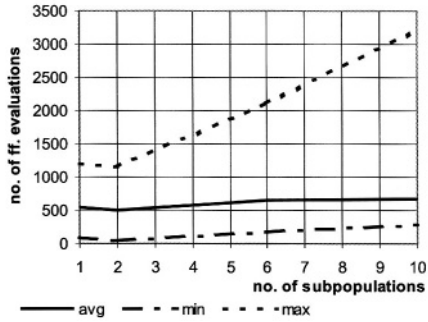


Fig. 8. Number of fitness function evaluations with respect to subpopulations number.

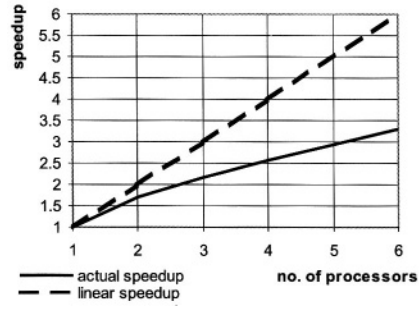


Fig. 9. Speedup of the improved distributed evolutionary algorithm with 2 subpopulations

takes about 0.4s. The speedup depends on the fitness function evaluation time. The speedup is restricted by the ratio between the fitness function evaluation time and time needed for communication between processors.

5 Conclusions

The parallel evolutionary algorithm speeds up computations. The optimal number of subpopulations in the distributed evolutionary algorithm depends on the problem. In the considered problem the optimal number of subpopulations was 2. The improved distributed evolutionary algorithm can be used instead of the classical distributed evolutionary algorithm when number of subpopulations differs from the number of available processing units.

Acknowledgement. This research was carried out in the framework of the KBN grant no. 4T11F00822.

References

1. J.T. Aleander: An Indexed Bibliography of Distributed Genetic Algorithms, University of Vaasa, Report 94-1-PARA, Vaasa, Finland, (2000).
2. J. Arabas: Lectures in evolutionary algorithms, WNT, Warszawa, (2001). (in Polish)
3. Brebbia C.A., Telles J.C.F., Wrobel L.C: Boundary Element Techniques. Springer-Verlag, Berlin, (1984).
4. T. Burczynski , W. Kus: Shape optimization of elasto-plastic structures using distributed evolutionary algorithms. Proc. European Conference on Computational Mechanics ECCM 2001, Cracow (2001.
5. Burczynski T., Kus W.: Evolutionary methods in shape optimisation of elastoplastic structures. Proc. 33rd Solid Mechanics Conference. Zakopane, (2000).
6. Cantu-Paz E.: A Survey of Parallel Genetic Algorithms, Calculateurs Paralleles, Reseaux et Systems Repartis. Vol. 10, No. 2, pp. 141-171, Paris, (1998).
7. W. Kus: Coupled boundary and finite element method in optimization of mechanical structures, Ph.D. Thesis, Gliwice, (2002). (in Polish)
8. W. Kus, T. Burczynski: Evolutionary optimization of elastoplastic solids. Proc. AI-MECH 2000 Symposium on Methods of Artificial Intelligence in Mechanics and Mechanical Engineering (eds. T.Burczynski and W.Cholewa), Gliwice, (2000).
9. Michalewicz Z.: Genetic algorithms + data structures = evolutionary algorithms. Springer-Verlag, Berlin, (1996).
10. Shewchuk R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, First Workshop on Applied Computational Geometry, Association for Computing Machinery, pp. 124-133, Philadelphia, Pennsylvania, USA, (1996).
11. Tanese R.: Distributed Genetic Algorithms. Proc. 3rd ICGA, pp.434-439, Ed. J.D. Schaffer. San Mateo, USA, (1989).

A Parallel Evolutionary Algorithm for Discovery of Decision Rules

Wojciech Kwedlo

Faculty of Computer Science
Białystok Technical University
Wiejska 45a, 15-351 Białystok, Poland
wkwedlo@ii.pb.bialystok.pl

Abstract. In the paper a new parallel method for learning decision rules is proposed. The method uses evolutionary algorithm to discover decision rules from datasets. We describe a parallelization of the algorithm based on master-slave model. In our approach the dataset is distributed among slave processors of a parallel system. The slave procesors compute fitness function of chromosomes in parallel. The remainder of evolutionary algorithm i.e. selection and genetic search operators is executed by the master processor. Our method was implemented on a cluster of SMP machines connected by Fast Ethernet. The experimental results show, that for large datasets it is possible to obtain a significant speedup.

1 Introduction

Discovery of decision rules is a well-known classification technique, which is commonly used in data mining. The advantages of the rule-based approach include natural representation and ease of integration of learned rules with background knowledge.

Evolutionary algorithms (EAs) [5] are stochastic optimization techniques, which have been inspired by the process of biological evolution. Their advantage over greedy search methods is the ability to avoid local optima. Several EA-based systems, which learn decision rules were proposed [1,3]. The solutions obtained by those systems are often better than the solutions obtained by traditional methods. However, the main disadvantage of EAs is their high computational complexity. In many real-life applications of data mining the size of analyzed dataset is very large. Is such cases the big computational complexity of EAs makes their use extremely difficult. A possible solution of this problem is a parallel implementation of the given algorithm.

In the paper we describe a parallel implementation of the system EDRL-MD (Evolutionary Decision Rule Learner with Multivariate Discretization) [3]. The main advantage of EDRL-MD in comparison with other EA-based systems is the capability of direct extraction of rules from datasets with continuous-valued attributes. The other systems require prior *discretization* of such attributes.

The reminder of the paper is organized as follows. The next section presents EDRL-MD system. The parallel formulation of the system is described in Section

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 580–585, 2004.
© Springer-Verlag Berlin Heidelberg 2004

3. Section 4 is devoted to presentation of the results of computational experiments investigating scalability of our approach. The last section contains the conclusions.

2 Description of the System EDRL-MD

In this section we present two main topics, i.e. representation of solutions by chromosomes and the fitness function, which are most important for parallel formulation of the algorithm. Description of the remaining components, e.g. genetic operators, can be found in [3] or [4].

2.1 Basic Notions

We assume that a learning set $E = \{e_1, e_2, \ldots, e_M\}$ consists of M examples. Each example $e \in E$ is described by N attributes (features) $A_1, A_2, \ldots, A_N$ and labelled by a class $c(e) \in C$. The domain of a nominal (discrete-valued) attribute A_i is a finite set $V(A_i)$, while the domain of a continuous-valued attribute A_j is an interval $V(A_j) = [l_j, u_j]$. For each class $c_k \in C$ by $E^+(c_k) = \{e \in E : c(e) = c_k\}$ we denote the set of *positive examples* and by $E^-(c_k) = E - E^+(c_k)$ the set of *negative examples.* A *decision rule R* takes the form IF $t_1 \wedge t_2 \wedge \ldots \wedge t_r$ THEN c_k, where $c_k \in C$ and the left-hand side (LHS) is a conjunction of $r (r \leq N)$ conditions $t_1, t_2, \ldots, t_r$; each of them concerns one attribute. The right-hand side (RHS) of the rule determines class membership of an example. A *ruleset RS* is a disjunctive set of decision rules with the same RHS. By $c_{RS} \in C$ we denote the class on the right-hand side of the ruleset *RS.*

In our approach the EA is called once for each class $c_k \in C$ to find the ruleset separating the set of positive examples $E^+(c_k)$ from the set of negative examples $E^-(c_k)$. The search criterion, in terminology of EAs called the *fitness function* prefers rulesets consisting of few conditions, which cover many positive examples and very few negative ones.

2.2 Representation

The EA processes a population of candidate solutions to a search problem called *chromosomes.* In our case a single chromosome encodes a ruleset *RS.* Since the number of rules in the optimal ruleset for a given class is not known, we use variable-length chromosomes and provide the search operators, which change the number of rules. The chromosome representing the ruleset is a concatenation of *strings.* Each fixed-length string represents the LHS of one decision rule. Because the EA is called to find a ruleset for the given class c_{RS} there is no need for encoding the RHS.

The string is composed (Fig. 1) of *N substrings.* Each substring encodes a condition related to one attribute. The LHS is the conjunction of these conditions. In case of a continuous-valued attribute A_i the substring encodes the lower

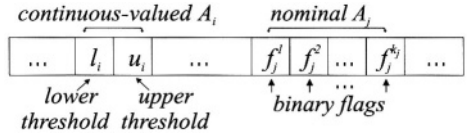


Fig. 1. The string encoding the LHS of a decision rule ($k_j = |V(A_j)|$). The chromosome representing the ruleset is the concatenation of strings. The number of strings in a chromosome can be adjusted by some search operators.

l_i and the upper u_i threshold of the condition $l_i < A_i \leq u_i$. It is possible that $l_i = -\infty$ or $u_i = +\infty$.

For a nominal attribute A_j the substring consists of binary flags. Each of the flags corresponds to one value of the attribute.

2.3 The Fitness Function

Consider a ruleset *RS,* which covers *pos* positive examples and *neg* negative ones. The number of positive and negative examples in the learning set is denoted by *POS* and *NEG* respectively. The ruleset *RS* classifies correctly *pos* positive examples and *NEG – neg* negative ones. Hence the probability of classifying correctly an example from the learning set is given by:

$$Pr(RS) = \frac{pos + NEG - neg}{POS + NEG}. \tag{1}$$

The fitness function is defined as:

$$f(RS) = Pr(RS) - \alpha Compl(RS), \tag{2}$$

where *Compl(RS)* is the complexity of the ruleset and α is a user supplied parameter. As a measure of complexity we take:

$$Compl(RS) = L/N + 1, \tag{3}$$

where *L* is total the number of conditions in the ruleset *RS* and *N* is the number of attributes.

3 Implementation in a Parallel System

The main loop of evolutionary algorithm begins with computation of fitness function of all *S* individuals in the population. In the next step, called *selection* [5], a new population is created by multiple random choice of chromosomes with high fitness from the old population. After selection, some genetic operators like *mutation* and *crossover* are applied. The algorithm iterates these three steps until a termination condition is met.

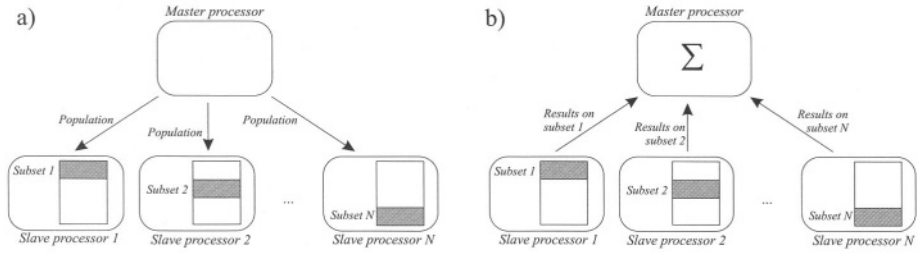


Fig. 2. Computation of the fitness function.

As equation (1) shows to determine the fitness of a chromosome it is necessary to calculate the counts of positive and negative examples denoted by *pos* and *neg* respectively. To obtain *pos* and *neg* the algorithm has to iterate through all the examples in the learning set. For each example $e_i \in E$ the algorithm checks if e_i is covered by the ruleset *RS*. If the example matches a premise of at least one rule from the *RS* it is regarded as covered. Then, depending on the type of the example either the counter of positive examples or the counter of negative examples is incremented. In many practical applications *M,* i.e. the size of the learning set is very large. Moreover, the CPU time required by remaining components of the EA i.e. genetic operators and selection does not depend on the size of the learning set. (it depends on size of the population *S* instead). In almost all cases $S << M$. For instance in our experiments we always set $S = 50$, whereas for the smallest dataset the size $M = 2310$. Consequently the the computational complexity of the algorithm is dominated by the calculation of the fitness.

For these reasons in our parallel implementation we decided to focus on computation of the fitness. Selection and genetic operators are executed sequentially. The algorithm runs in a *master–slave* model. The dataset is divided evenly into subsets; each subset is placed on a single slave processor. Each slave processor is responsible for the evaluation of the rules on the corresponding subset. At the begin of an iteration of EA the master processor broadcasts (Fig. 2a) the population i.e., the set of *S* chromosomes to all slave processors. For each ruleset in the population a slave processor counts the number of covered positive and negative examples from its subset. The master processor gathers (Fig. 2b) results from each slave, sums up the counts for each ruleset and computes the fitness.

When the fitness of all chromosomes is computed the remaining part of the iteration of EA (i.e. the selection and genetic operators) is executed solely on the master processor.

4 Experimental Results

In this section experimental results are presented. We have tested the parallel version of EDRL-MD on six datasets of different size. The datasets were taken

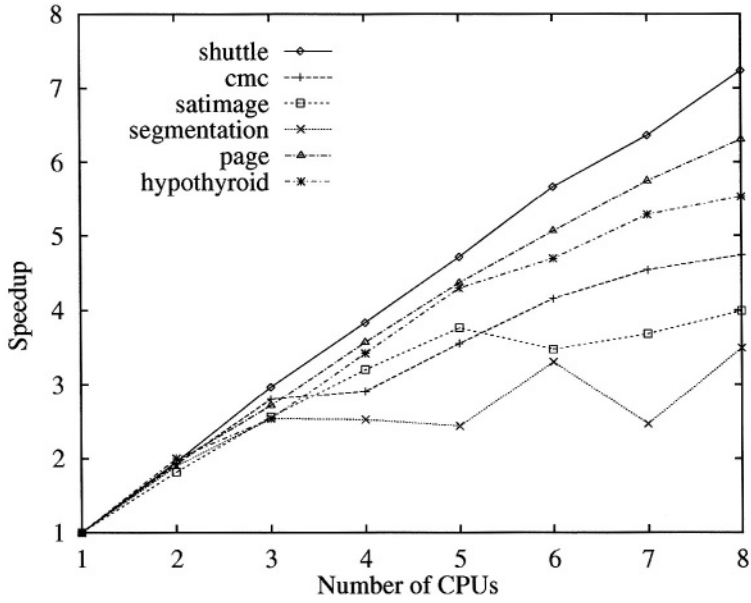


Fig. 3. Speedup obtained for different datasets.

Table 1. The datasets used in the experiments

Dataset name	No. of records	No. of attributes	No. of classes
page	5473	10	5
segmentation	2310	19	7
hypothyroid	3773	21	3
cmc	1473	9	3
shuttle	43500	9	7
satimage	4435	36	6

from the repository of publicly available data at University of California, Irvine [6]. The description of the datasets is shown in Table 1.

To evaluate the proposed approach we performed an experiment on a 8 CPU cluster consisting of one four-processor and two two-processor machines (4 * Pentium III Xeon 700 + 2 * 2 * Pentium III 750) running Linux 2.4.18. Machines in the cluster were connected via a Intel 440T Fast Ethernet switch. For message passing MPICH [2] MPI implementation was used. We used the default configuration of MPICH.

Usually, to compute a speedup or efficiency of an parallel algorithm the total processing time is used. In our case this time is proportional to the number of iterations, which in turn depends on the termination condition. The algorithm terminates when the fitness of the best chromosome does not improve during

consecutive N_{TERM} generations. Because the EA is a probabilistic algorithm, number of iterations and the processing time can significantly vary with different runs on the same data. However the time of a single iteration is approximately constant for the given learning set. Therefore an average time of a single iteration was used to compute speedup of our method. To compute the average time of an iteration we divided total processing time by the number of iterations.

Figure 3 shows the speedup obtained by our implementation for all datasets. For the largest data set (shuttle) we were able to achieve speedup greater than 7.2 for eight processors.

5 Conclusions

In this paper we have shown, that the computational efficiency of evolutionary algorithms for data mining applications can be significantly improved by the use of parallel machines. We proposed a parallel version of EDRL-MD based on data distribution approach. The experimental results suggest that for large datasets near-linear speedup is possible.

References

1. Janikow, C.: A knowledge intensive genetic algorithm for supervised learning. *Machine Learning* 13 (1993) 192–228.
2. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing standard. *Parallel Computing,* 22 (1996) 789–828.
3. Kwedlo, W. Kretowski, M.: An evolutionary algorithm using multivariate discretization for decision rule induction. *3rd European Conference on Principles and Practice of Knowledge Discovery in Databases. Lecture Notes in Artificial Intelligence 1704,* (1999) 392–397, Springer Verlag.
4. Kwedlo, W. Kretowski, M.: Learning decision rules using a distributed evolutionary algorithm. *TASK Quarterly,* 6 (2002) 483–492.
5. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs.* 3^{rd} edn. Springer Verlag (1996).
6. Murphy, P., Aha, D.: *UCI repository of machine-learning databases,* available online: http://www.ics.uci.edu/pub/machine-learning-databases.

An Evolutionary Programming Algorithm for Automatic Engineering Design

Andrew Lewis[1], David Abramson[2], and Tom Peachey[2]

[1] Div. of Information Services, Griffith University, Brisbane, Queensland, Australia,
a.lewis@griffith.edu.au
[2] Dept. of Computer Science and Software Engineering, Monash University,
Melbourne, Victoria, Australia
{davida,tcp}@csse.monash.edu.au

Abstract. This paper describes a new Evolutionary Programming algorithm based on Self-Organised Criticality. When tested on a range of problems drawn from real-world applications in science and engineering, it performed better than a variety of gradient descent, direct search and genetic algorithms. It proved capable of delivering high quality results faster, and is simple, robust and highly parallel.

1 Introduction

In the engineering design process there is a demand for the capability to perform automatic optimisation, minimising or maximising some derived quantity, a measure of "fitness" of the design. For real-world problems these objective function values are often computed using sophisticated and realistic numerical simulations of physical phenomena. This is an extremely computationally intensive process when models must be run tens, or maybe hundreds, of times to effectively search design parameter space. Current High Performance Computing systems mostly derive their capacity from parallel architectures so for optimisation methods to be practical and effective the algorithms used must preferably have a large degree of concurrency. Figure 1 offers a simple taxonomy of methods that can be used for this automatic optimisation.

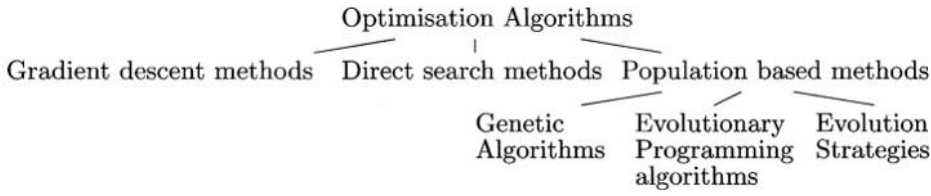


Fig. 1. A taxonomy of optimisation algorithms

Gradient descent algorithms, based on classical, Newtonian methods of analysis, generally exhibit rapid local convergence [1] but due to the iterative steps

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 586–594, 2004.
© Springer-Verlag Berlin Heidelberg 2004

of gradient determination and some form of line search most employ, they are inherently sequential in nature. In many problems arising in real-world design processes there are limits imposed on problem parameters by their nature; it does not make sense, for example, for a physical component to have negative size. The authors have previously exploited this simple bounding of engineering problems to enhance the parallelism of line searching to some extent [2].

Of greater potential concurrency are **direct search methods,** such as the Nelder-Mead Simplex algorithm[3]. These methods do not explicitly use approximate gradient information, but instead take objective function values from a set of sample points, and use that information to continue sampling. Use of the Nelder-Mead simplex algorithm remains current, largely because on a range of practical engineering problems it is capable of returning a very good result [4].

The early direct search methods, however, exhibit very limited concurrency. The Multidirectional Search (MDS) method of Dennis & Torczon[5] has greatly enhanced parallelism. The authors have explored hybrids and variants of Nelder-Mead Simplex and MDS in order to try and increase the amount of concurrency.

Offering the greatest concurrency are **population-based methods,** in which large numbers of objective function evaluations are typically performed in parallel, evolving solutions over several "generations". It is only over the past decade that the computational capacity available to scientists and engineers has increased to the point where population-based methods of optimisation have become practical for the solution of real-world problems. Moreover, in engineering design, the evaluation of the objective function is so much slower than the rest of the algorithm, that such codes demonstrate excellent speedup in spite of the need for global communication on each iteration. **Genetic Algorithms** (GA) now may be frequently encountered in application to engineering problems (for examples, see Alander[6]). These have inherent, easily exploitable parallelism, and attractive global convergence probabilities, but have been considered generally slow to converge [7].

In the general case, evolutionary computation seeks to use insight into natural processes to inform population-based computational methods. **Evolutionary Programming** refers to that class of methods in evolutionary computation that apply a random mutation to each member of a population, generating a single offspring. However, unlike Genetic Algorithms, no recombination operators are applied. Selection takes place, and half the combined population of parents and offspring enter the next generation. Population members are considered as representative of species, rather than individuals. Such methods are generally simple, robust and highly parallel.

The theory of self-organised criticality gives an insight into emergent complexity in nature [8]. Bak contended that the critical state was "the most efficient state *that can actually be reached dynamically*". In this state, a population in an apparent equilibrium evolves episodically in spurts, a phenomenon known as *punctuated equilibrium.* Local change may affect any other element in the system, and this delicate balance arises without any external, organising force. We have utilised these concepts in developing a new Evolutionary Programming

algorithm that, in addition to the conventional mutation and selection operations, implements a further selection operator to encourage the development of a self-organised critical system. The algorithm is evaluated on a range of test cases drawn from real-world problems, and compared against a selection of algorithms, including gradient descent, direct search methods and a genetic algorithm.

2 EPSOC: An Evolutionary Programming Algorithm Using Self-Organised Criticality

The general optimisation problem for an arbitrary non-linear function f can be stated as:

$$\text{Minimize } f(\mathbf{x}), \quad f : \Re^n \rightarrow \Re^1 \text{ and } \mathbf{x} = \{x_0, ..., x_i, ..., x_n\}, x_i \in \Re. \qquad (1)$$

For a population-based method, the population, p, consists of a set of parameter vectors, $\mathbf{x} : p = \{\mathbf{x}_0, ..., \mathbf{x}_i, ..., \mathbf{x}_m\}$. For real-world engineering design problems, values for $f(\mathbf{x})$ generally can only be derived from execution of complex numerical simulations requiring considerable computation time. Acceptable parameter values are bounded in the majority of cases, due to physical constraints of the problem considered. EPSOC treats all parameters as simply bounded.

As stated in the introduction, EPSOC is built on the basis of an Evolutionary Programming algorithm, with an additional selection operator following the method of Bak. The steps of the algorithm are outlined below:

1. Initialise a random, uniformly-distributed population, p, and evaluate each trial solution, $\mathbf{x}_i \ \forall \ i$.
2. Sort the population by objective function value, $f(\mathbf{x})$.
3. Select a set, B, of the *nbad* worst members of the population. For each member of B, add to the set its two nearest neighbours in parameter space that are not already members of the set, *or in the best half of the sorted population.*
4. Apply a random, uniformly-distributed mutation to the selected set, B, i.e. re-initialise them. For all other members of the population, generate a "child" by applying a small (~10% of parameter range), random, uniformly-distributed mutation to the "parent" member.
5. Evaluate each new trial solution, $f(\mathbf{x})$.
6. If a child has a better objective function value than its parent, replace the parent with the child.
7. Repeat from step 2 until a preset number of iterations have been completed.

As each set of parameters defining a trial solution is independent of all others, it is immediately apparent that the evaluation of trial solutions at steps 1 and 5 can be performed concurrently. Since the evaluation of the objective function completely dominates the execution time, from Amdahl's Law we can expect extremely high parallel efficiency.

Critical states of evolutionary models exhibit a power-law distribution of the size and frequency of extinction events. This is apparent when the number of events

involving given numbers of members of the population approximate a straight line in a double-log statistical plot of the phenomena. In earlier applications of self-organised criticality to optimisation, it has been proposed that a separately computed power-law extinction rate be imposed on a spatial diffusion model, or cellular GA[9]. In Krink and Thomsen's model, population members are chosen at random for extinction, and the algorithm is greedy to the extent of maintaining the single best member, an elite of one. Their algorithm apparently does not attempt to evolve a population in a critical state, but indirectly imposes the observed behaviour of such a population.

In contrast, EPSOC is a straightforward implementation of Bak's nearest-neighbour, punctuated equilibrium model as an optimisation algorithm. By considering the trial solution parameter vectors as defining a location in an n–dimensional parameter space, the spatial behaviour of Bak's model is realized naturally. We also chose to apply a high degree of greediness to the algorithm. Maintaining a large "elite" (in EPSOC, half the total population) can be viewed as a "constructive" operator. Like "Maxwell's demon", it accretes information and encourages a gradual improvement in the better half of the population. This achieved as re-initialised or mutated population members move into the "protected" half of the population, and the median of the population moves toward better objective function values. Since the process of reinitialisation effectively constitutes a uniformly-distributed random search, from the theory of random search it may be conjectured that the algorithm will find global minimizers of a problem, *if allowed sufficient iterations.*

3 The Case Studies

We have assembled a number of case studies drawn from interesting and challenging scientific and engineering applications. These were used to test and assess the performance of the individual algorithms. Generally, the case studies fall into 2 sets:

- Smooth, with a dominant global minimum (Laser 1, Crack 1, Aerofoil - Figures 2(a), 2(c) & 2(e), and Rosenbrock's function)
- Multiple local minima, non-convex (Laser 2, Crack 2, Bead - Figures 2(b), 2(d) & 2(f))

3.1 Laser 1 and 2

A two-dimensional test surface was derived from the computation of a quantum electrodynamical simulation of a laser-atom interaction experiment [10]. The base case, Laser 1, is quite a smooth surface, the dataset containing only 4 minima, of which the global minimum is quite dominant, as can be seen in Figure 2(a). Additive fractal noise was overlaid on this dataset to develop a " noisier", more challenging surface to test the algorithms. This dataset, Laser 2, contained 1157 local minima of varying severity, and is illustrated in Figure 2(b).

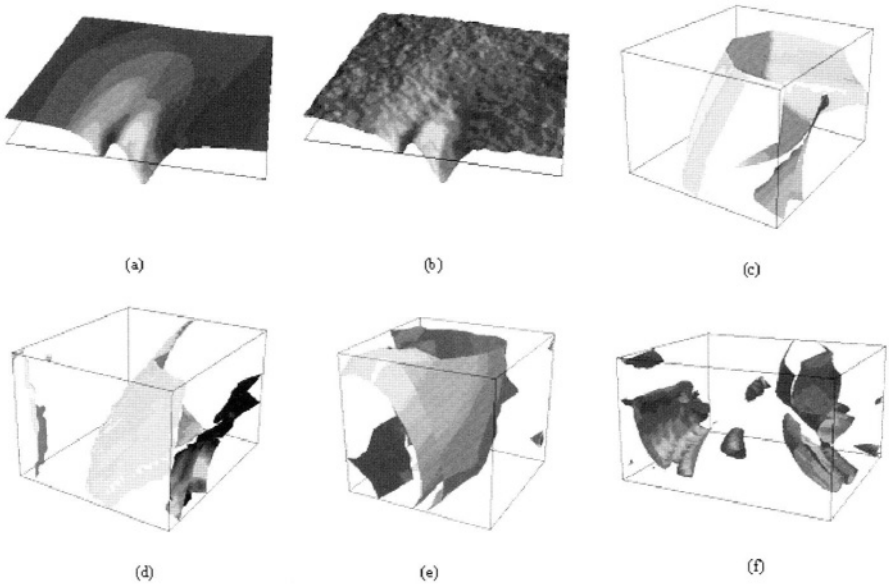


Fig. 2. Test case isosurfaces

3.2 Crack 1 and 2

Finite element analysis of a thin plate under cyclic loading, with a cutout specified by parameters, was used to generate the Crack datasets[11]. Common practice in damage tolerant design has been to minimise the maximum stress under load. Isosurfaces of these stress values are shown in Figure 2(c). This dataset, Crack 1, was reasonably smooth, with only 26 local minima. A new approach in modeling stressed components is to attempt to maximize durability. The finite element model of the crack-stress test case included fatigue cracks at a number of locations, and the objective of this test case, Crack 2, was to maximise the life of the part as determined by the time taken for fatigue crack growth to a defined length. Isosurfaces at a number of values are shown in Figure 2(d). In contrast to Crack 1, this dataset was "noisy", with 540 local maxima, and discontinuous isosurfaces.

3.3 Aerofoil

This test case models the aerodynamic properties of a two dimensional aerofoil. The objective function to be minimised is the lift-drag ratio[12], and this is computed by executing a Computational Fluid Dynamic model of the object. Figure 2(e) shows a number of isosurfaces in the parameter space investigated. The dataset was generally smooth, with only 12 local minima and a dominant global minimum.

3.4 Bead

The application from which this case study was drawn used a ceramic bead to minimise distortion of the radiation pattern of a mobile telecommunications handset during testing[13]. The objective function value, derived from an FDTD full-wave analysis of the cable structure, was a measure of transmission strength through the bead at 1 GHz. The dataset for the Bead case study, of which isosurfaces for a particular value are shown in Figure 2(f), is quite complex and contains 298 local minima.

3.5 Rosenbrock's Function

In order to provide a point of comparison, the well-known Rosenbrock's function in two dimensions was included. The objective function values for this test were directly computed from $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ for $x_i \in [-$ which has one local minimum at $f(1,1) = 0$.

4 Results of Numerical Experiments

For purposes of comparison a set of algorithms were tested, including EPSOC, a GA (Genesis 5.0[14]), a parallel gradient descent method (P-BFGS), Dennis and Torczon's MDS, the Simplex method of Nelder and Mead, a variant of Simplex operating concurrently on multiple vertices with a search direction defined by a reducing set of vertices (RSCS), and variants of Simplex and RSCS utilizing line-searching. All were parallel implementations; no serial algorithms were tested.

The population-based algorithms, EPSOC and Genesis, used a population of 64 members. Other operational parameters of the algorithms, such as mutation and crossover rates for the GA, and extinction rate and mutation range for EPSOC, were tuned for each test case and the best results obtained are reported in this paper. The extinction rate, *nbad,* varied between 5% and 15% of the population. Both algorithms are ideally suited to a master-slave, gather-scatter parallel system.

Each of the algorithms tested was run on each of the test cases from 10 randomly distributed start points. To allow direct comparison, in a given test case the same set of start points were used for each algorithm.

The starting simplices for the direct search methods were right simplices aligned with the coordinate axes. By default they were reasonably large, scaled to 10% of the parameter range for each coordinate. The convergence criterion for most cases was a fractional step-wise gradient of 10^{-3}, except for EPSOC and Genesis, which used fixed limits on iteration count and function evaluations, respectively.

Table 1 shows the best objective function value obtained in 10 runs for each algorithm on each test case, and the Equivalent Serial Function Evaluations (ESFE) required to obtain that result. ESFE, derived from the number of steps involving concurrent evaluation of objective functions, is a measure of the actual

time taken to find an optimal solution. For each test case, the best objective function value obtained by any algorithm, and the fastest ESFE to obtain that value, are highlighted. The ESFE results shown for EPSOC and the GA are the generation counts at which the minimum result was obtained, not the count for termination, which was fixed.

Table 1. Best objective functions values obtained in 10 runs, and time taken

Algorithm	*Laser 1*		*Laser 2*		*Crack 1*		*Crack 2*		*Aerofoil*		*Bead*		*Rosenbrock*	
	Obj.	ESFE	Obj.	ESFE	Obj.	ESFE	Obj.	ESFE	Obj.	ESFE	Obj.	ESFE	Obj.	ESFE
EPSOC	**-0.48**	**10**	**-0.56**	**12**	**187.5**	**5**	5356	20	**-68.64**	20	**-39.85**	14	1e-3	14
GA	**-0.48**	14	**-0.56**	19	188.5	16	5346	11	-68.51	9	**-39.85**	**7**	2e-4	23
P-BFGS	**-0.48**	37	**-0.56**	39	**187.5**	26	5347	12	-67.90	26	-29.71	18	7e-2	10
Simplex	**-0.48**	24	**-0.56**	24	187.6	25	5353	23	**-68.64**	20	-26.98	16	**0**	54
SimplexL1	**-0.48**	15	**-0.56**	24	**187.5**	13	5331	12	-68.63	18	**-39.85**	13	5e-6	48
MDS	**-0.48**	16	**-0.56**	**12**	187.6	13	**5357**	1000	**-68.64**	14	-16.12	7	3e-4	1000
RSCS	**-0.48**	22	**-0.56**	**12**	187.6	9	5347	42	**-68.64**	**7**	-26.91	12	**0**	**39**
RSCSL	**-0.48**	22	**-0.56**	24	**187.5**	15	**5357**	**27**	-67.21	18	-26.98	11	6e-6	88
RSCSL1	**-0.48**	13	**-0.56**	17	187.7	11	5348	8	-68.62	15	-26.98	5	8e-4	33

It may be noted from Table 1 that EPSOC effectively found the global minimum in 6 out of 7 cases (counting the result on the Crack 2 test case as successful, since it was within 0.03% of the global minimum value). In half of these cases, it also achieved this result faster than other algorithms. Its performance on Rosenbrock's function was equivalent to that of the other algorithms. The Kruskal-Wallis H test statistic was used to rank algorithms on each test case, based on all results returned. The Mann-Whitney U test was used for pairwise comparisons of the two highest-ranked algorithms, one of which was EPSOC on every test case. On 4 test cases EPSOC was better than the second-ranked algorithm to a significance level of 0.05. On the remaining cases there was no statistically significant difference.

In general, EPSOC achieved results as good as, or better, than all other algorithms, and its rate of convergence was highly competitive.

5 Conclusion

In this paper we have described a simple new Evolutionary Programming algorithm that utilizes concepts of Self-Organised Criticality. Tested on a range of cases drawn from real-world problems, against a representative set of direct search, gradient descent and genetic algorithms, it has been demonstrated to exhibit superior performance.

Examination of the results from the numerical experiments demonstrates population-based methods, EPSOC and the GA, are competitive against classical gradient descent and direct search methods providing they are executed on a parallel machine with sufficient processors to evaluate all of the population members in one iteration.. The quality of results obtained, and their speed in achieving them were clearly better in a majority of cases. Within the population-based methods, EPSOC outperformed the GA in finding the global minimum

on all but one test case. Even for that case, the median result across multiple runs for EPSOC was better than that of the GA.

Like the GA, EPSOC is highly parallel, evaluating 1280 trial solutions in the same time it took Simplex, for example, to evaluate 81, and to generally better effect. The parallelism and completion time of the algorithm are set independent of problem size. The resulting pre-determined execution behaviour, in terms of resources required and time taken, its simplicity and its easily implemented master-slave parallelism make it well-suited for practical application to real-world problems.

Given the intended use of these algorithms in practical engineering design, they have all, with the exception of Genesis 5.0, been integrated into the Nimrod/O framework [12]. Nimrod/O is a design optimisation toolset. It makes it possible to describe a design scenario using a simple, declarative language, then use parallel or distributed computers seamlessly to execute the experiment. Unlike other systems Nimrod/O combines optimisation, distributed computing and rapid prototyping in a single tool.

Further work is required on EPSOC to determine the efficacy of the method on problems of high dimensionality, and the link between problem dimensionality and optimal population size, extinction rates and mutation factors.

References

1. Fletcher, R.: Practical Methods of Optimization. Wiley, New York, (1987)
2. Lewis, A., Abramson, D., and Simpson, R.: Parallel Non-Linear Optimization: Towards the Design of a Decision Support System for Air Quality Management. Proc. ACM/IEEE SC97 Conf., San Jose, CA, (1997)
3. Nelder, J.A., and Mead, R.: A simplex method for function minimization. Comput. J. **7** (1965) 308–313
4. Wright, M.H.: Direct Search Methods: Once Scorned, Now Respectable. In Numerical Analysis 1995(Proc. 1995 Dundee Biennial Conference in Numerical Analysis, Griffiths, D.F. and Watson, G.A. (eds.), Harlow, UK, (1995) 191–208
5. Torczon, V.: Multidirectional Search. Ph.D. thesis, Rice University, Houston, (1989)
6. Alander, J.T.: An Indexed Bibliography of Genetic Algorithms in Computer Aided Design. Report 94-1-CAD, Department of Information Technology and Production Economics, University of Vaasa, Finland, (1995)
7. Durand, N., and Alliot, J-M.: A Combined Nelder-Mead Simplex and Genetic Algorithm. GECCO'99: Proc. Genetic and Evol. Comp. Conf. Orlando, FL, (1999)
8. Bak, P.: How Nature Works. Springer-Verlag, New York, (1996)
9. Krink, T., and Thomsen, R.: Self-Organized Criticality and Mass Extinction in Evolutionary Algorithms. Proc. CEC2001, Seoul, Korea, (2001)
10. Abramson D., Sosic R., Giddy J. and Hall B.: Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations. The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August (1995)
11. Peachey, T., Abramson, D., Lewis, A., Kurniawan, D. and Jones, R.: Optimization using Nimrod/O and its Application to Robust Mechanical Design. Fifth Int. Conf. Par. Proc. and Appl. Maths (PPAM 2003), Czestochowa, Poland, (2003)

12. Abramson, D., Lewis, A., Peachey, T., Fletcher, C.: An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics. Proc. ACM/IEEE SC2001 Conf., Denver, CO, (2001)
13. Lewis, A., Saario, S., Abramson, D. and Lu, J.: An Application of Optimisation for Passive RF Component Design. Conf. on Electromag. Field Comp., Milwaukee, (2000)
14. Grefenstette, J.J.: GENESIS: A system for using genetic search procedures. In Proceedings of the 1984 Conference on Intelligent Systems and Machines, (1984) 161–165

Weighted Vector Directional Filters Optimized by Genetic Algorithms

Rastislav Lukac[1], Bogdan Smolka[2]*, Andrzej Swierniak[2], Konstantinos N. Plataniotis[3], and Anastasios N. Venetsanopoulos[3]

[1] Slovak Image Processing Center,
Jarkova 343, 049 25 Dobsina, Slovak Republic
lukacr@ieee.org
[2] Department of Automatic Control,
Silesian University of Technology, Akademicka 16 Str., 44-101 Gliwice, Poland
bsmolka@ia.polsl.gliwice.p1
[3] The Edward S. Rogers Sr. Dept. of Electrical and Computer Engineering,
University of Toronto, 10 King's College Road, Toronto, Canada
{kostas, anv}@dsp.utoronto.ca

Abstract. This paper focuses on genetic optimization and filtering efficiency of a recently developed class of Weighted Vector Directional Filters (WVDFs), which minimize the aggregated weighted angular distances between the samples contained in a sliding filtering window. As shown in the paper, the proposed genetically optimized WVDFs hold excellent detail preserving characteristics and are able to remove impulsive noise affecting color images.

1 Introduction

The field of color image filtering has attracted much interest in the recent years, because colors provide very often-crucial information about the objects on the image scene. It is known that a human eye can recognize about 10 million different colors and thus, color information can help in different tasks of computer vision and image understanding.

In the last decade, a rich family of vector filtering algorithms was developed to suppress noisy samples in multichannel signals, such as color images and simultaneously to preserve fine image details and color chromaticity. The popularity of vector filters lies in their respect of a natural correlation among the color channels, where each image sample is processed as a three-dimensional vector of channel intensities. In the case of the impulse noise corruption, a class of filters, such as the examined weighted vector directional filters, based on ordering of input samples is usually preferred.

* B. Smolka is supported by the KBN grant 4T11F01824.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 595–600, 2004.
© Springer-Verlag Berlin Heidelberg 2004

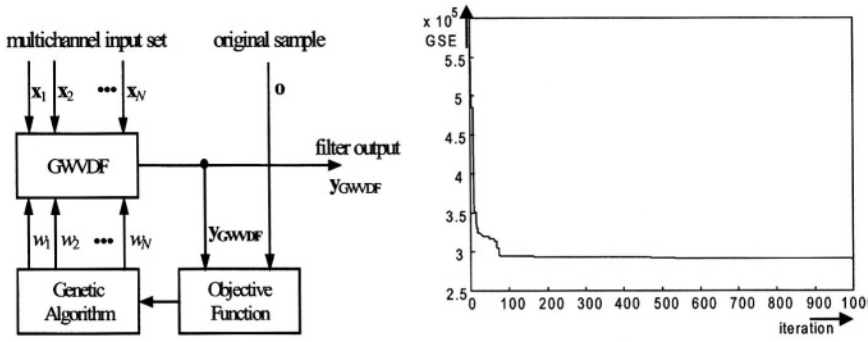


Fig. 1. GWVDF optimization scheme and the corresponding error criteria updated during the optimization procedure.

2 Weighted Vector Directional Filters

Let $y(x) : Z^l \rightarrow Z^m$ represent a multichannel image, where l is an image dimension and m denotes a number of channels. Let $W = \{\mathbf{x}_i \in Z^l; i = 1, 2..., N\}$ represent a filter window of a finite size *N*, where $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ is a set of noisy samples centered around the central sample $\mathbf{x}_{(N+1)/2}$.

Let us assume that $w_1, w_2, ..., w_N$ represent a set of positive real weights, where each weight w_i, for $i = 1, 2, ..., N$, is associated with the input sample $\mathbf{x}_i$, which relates to the aggregated angular quantity β_i defined as follows:

$$\beta_i = \sum_{j=1}^{N} w_j \mathrm{A}(\mathbf{x}_i, \mathbf{x}_j), \text{ for } i = 1, 2, ..., N \tag{1}$$

where

$$\mathrm{A}(\mathbf{x}_i, \mathbf{x}_j) = \cos^{-1}\left(\frac{\mathbf{x}_i \cdot \mathbf{x}_j^T}{|\mathbf{x}_i||\mathbf{x}_j|}\right) \tag{2}$$

represents the angle between two RGB vectors $\mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3})$ and $\mathbf{x}_j = (x_{j1}, x_{j2}, x_{j3})$. If the ordering scheme is given by ordered angular measures $\beta_{(1)} \leq \beta_{(2)} \leq ... \leq \beta_{(N)}$ and the same ordering is implied to the input vector-valued samples $\mathbf{x}_1(\beta_1), \mathbf{x}_2(\beta_2), ..., \mathbf{x}_N(\beta_N)$, it results in the ordered input set $\mathbf{x}_{(1)}; \mathbf{x}_{(2)}; ...; \mathbf{x}_{(N)}$.

The output of the weighted vector directional filter (WVDF) [5] is the sample $\mathbf{x}_{(1)} \in \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ associated with the minimum weighted angular distance $\beta_{(1)} \in \{\beta_1, \beta_2, ..., \beta_N\}$.

3 Genetically Optimized WVDF Filters

Depending on $\{w_1, w_2, ..., w_N\}$, the WVDF can be designed to perform a variety of filtering operations. The appropriate adaptation of the WVDF coefficients can

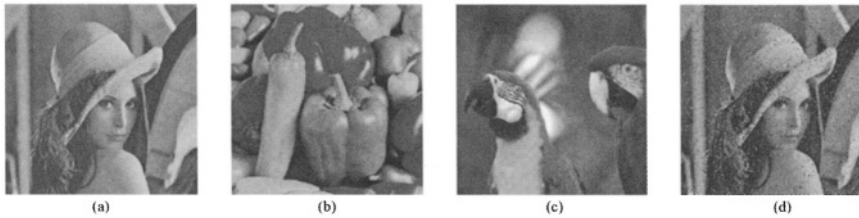


Fig. 2. Test images: (a) Lena, (b) Peppers, (c) Parrots, (d) the test image Lena corrupted 10% impulsive noises (the training set for the GWVDF optimization).

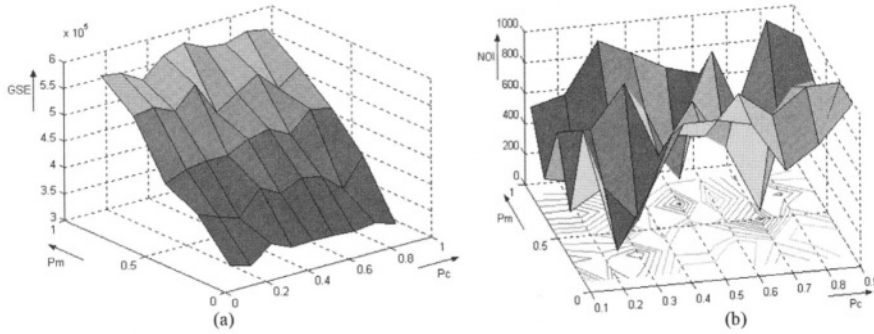


Fig. 3. GWVDF optimization depending on the crossover probability p_c and the mutation probability p_m: (a) GA error function, (b) number of iterations (NOI) necessary for finding the minimum error.

allow finding the optimal solution for many filtering problems. Since the WVDF passes to the output the sample of the input set, these filters are designed to work in environments corrupted by impulsive noise. Moreover, the WVDF can offer better balance between the noise attenuation and signal detail preservation than the presented standard vector filtering schemes.

According to a wide range of filtering operations provided by the WVDF, the genetic algorithms (GAs) [2],[6] can be used as the optimization tool for searching for the globally optimal weight coefficients. Using the genetic optimization, it is possible to adapt the weight coefficients to varying signal and noise statistics. As the result, the genetically optimized WVDF (GWVDF), shown in Fig.1, can achieve excellent balance between the signal-detail preservation and the noise attenuation capabilities.

In general, the proposed optimization problem can be solved using a wide range of possible methods, but it is very difficult to determine the optimal one. The GA belongs to the field of biologically oriented computational techniques. The GA is useful for searching for the optimal solution in situations where other optimization techniques may fail, e.g. due to complex specification of initial conditions, to find the optimal solution. In the GA optimization, an individ-

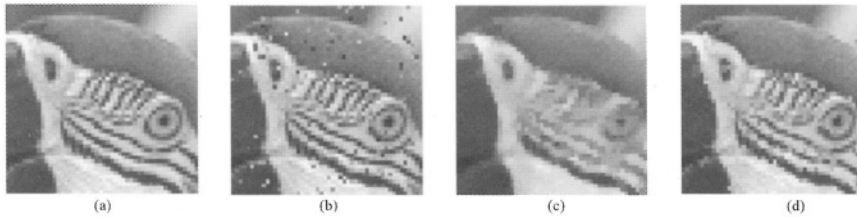


Fig. 4. Zoomed results related to test image Parrots. (a) original image, (b) noisy image (5% impulsive noise), (c) VMF output, (d) GWVDF output.

ual represented by a set of parameters, (genes in chromosomes) expresses the potential solution of the problem. The GA optimization starts with randomly generated solutions of the problem. The number of solutions is represented by the size of population. The optimization requires the quality criterion evaluating the generated solutions. The measure of the individual quality is known as a fitness value reflecting the degree of negotiability enforcing during the evolution.

Let us assume that **o** represents the original signal used to determine the quality of solutions. The following generation is created (via the recombination of the genes) from the chromosomes in the current population. Because of the genetic evolution, the individual (chromosome) with larger fitness tends to yield a superior offspring, which means better solution of the problem. Thus, the significant emphasis is imposed on the set of parent chromosomes that is selected by a random selection. After this selection, the genes of the parent chromosomes are mixed and recombined for the production of the offspring in the next generation. Two basic operations such as crossover and mutation are used. A crossover operator generates the offspring from two individuals of a current population. After the crossover operation, a mutation operator introduces the random information to the offspring. Since the GA utilizes the mechanism of natural selection through the evolution method, the individual can represent an optimal solution with the largest fitness in the last iteration.

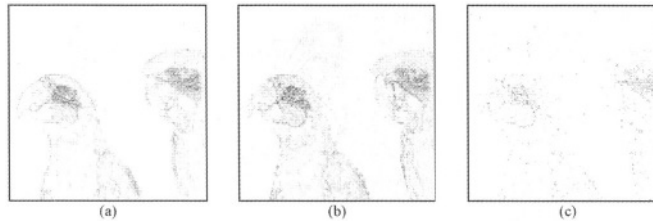


Fig. 5. Estimation errors related to the test image Parrots degraded by 5% impulsive noise: (a) VMF output, (b) BVDF output, (c) GWVDF output.

Table 1. Results achieved using the test image Parrots.

Noise	2% impulsive noise			5% impulsive noise			10% impulsive noise		
Method	MAE	MSE	NCD	MAE	MSE	NCD	MAE	MSE	NCD
Noisy	1.578	186.2	0.0177	3.805	443.6	0.0432	7.526	882.0	0.0857
VMF	2.493	58.0	0.0123	2.669	64.2	0.0132	2.890	69.6	0.0142
BVDF	3.289	101.8	0.0106	3.460	109.0	0.0116	3.630	113.5	0.0127
DDF	2.482	60.2	0.0108	2.645	65.3	0.0117	2.839	69.7	0.0128
GVDF	2.864	84.8	0.0117	3.036	93.6	0.0126	3.188	96.2	0.0137
WVDF1	2.597	83.4	0.0081	2.747	90.2	0.0068	2.362	77.2	0.0083
WVDF2	1.879	53.3	0.0059	2.061	62.2	0.0090	2.949	98.7	0.0101
GWVDF	0.693	20.4	0.0023	0.897	30.0	0.0036	1.341	59.6	0.0061

4 Experimental Results

For the evaluation of the efficiency of the proposed method we used a set of standard color test images (Fig. 2a-c): Lena, Peppers and Parrots. Because the primary filtering purpose of all filtering schemes outputting the lowest ranked sample is to remove impulsive noise, we corrupted the test images by impulsive noise (Fig.2d) [7].

Besides the subjective evaluation, the achieved results were evaluated through commonly used objective measures such as the mean absolute error (MAE), the mean square error (MSE) and normalized color difference (NCD) [7]. The above-mentioned criteria can be viewed as the measures of signal-detail preservation, noise attenuation and also capability to preserve the color chromaticity, respectively.

In order to analyze the GWVDF optimization, we used the image Lena corrupted by 10% impulsive noise as the training set. We tested the GA with the population and subpopulation size set as 120 and 80 chromosomes, respectively. The achieved GA optimization results are shown in Fig.1b and Fig.3. Note that the sub-optimal weight vector used below for filtering purposes corresponds to the crossover probability $p_c = 0.9$ and the mutation probability $p_m = 0.1$. A good solution was achieved after a few, usually 5-10 iteration cycles.

For filtering purposes, we tested the performance of the new method on noisy images, performing a series of tests in which we used the test images Peppers (Fig.2b) and Parrots (Fig.2c) corrupted by a wide range of noise corruption intensities. The results presented in Tab.1 and (Figs.4-5) are related to the filtering of images different from the training set shown in Fig.2d. Note that WVDF1 and WVDF2 characterize the non-adaptive WVDF with weight vector $\mathbf{w} = [2,1,2,1,3,1,2,1,2]$ and $\mathbf{w} = [1,2,1,4,5,4,1,2,1]$, respectively. We used

also Generalized Vector Directional Filter (GVDF) [7], Basic Vector Directional Filter (BVDF) [7], Directional Distance Filter (DDF) [3] and Vector Median Filter (VMF) [1].

It can be observed (Figs.4-5) that the optimized GWVDF filter provides excellent signal-detail preservation capability in comparison with all relevant methods. Standard filtering schemes (VMF, BVDF, DDF) excellently suppress impulses present in the image, however some edges and image details are heavily blurred, especially at transitions between image regions. The outputs of the proposed optimized GWVDF filter is characterized by excellent balance between the signal-detail preservation and the noise suppression. The drawback of the GWVDF method is its noise attenuation capability, which often fails in the presence of a large amount of injected noise.

5 Conclusion

In this paper, we presented the optimization analysis and performance of genetically optimized WVDF filters. Proposed optimized filters followed varying signal and noise statistics. Used GAs were able to determine the optimal weight coefficients for color image filtering purposes. The optimized WVDF filter outperformed standard filtering schemes especially in terms of color/structural content preservation.

Future research will focus on the GWVDF filters applied to television images. In this application, excellent GWVDF detail- preserving characteristics are required as well. Moreover, developing the GWVDF filter based on the 3-D supporting window and combined with the fast motion compensation techniques of [4] can lead to the efficient video filtering scheme.

References

[1] Astola, J., Haavisto, P., Neuvo, Y.: Vector median filters. Proceedings of the IEEE **78** (1990) 678–689

[2] Goldberg, D.: Genetic algorithms in search, optimisation, and machine learning. Addison-Wesley, Reading, Massachutts (1989)

[3] Karakos, D.G., Trahanias, P.E.: Generalized multichannel image-filtering structure. IEEE Trans. Image Processing **6** (1997) 1038–1045

[4] Klima, M., Zahradnik, P., Novak, M., and Dvorak, P.: Simple motion detection methods in TV image for security purposes. Proc. IEEE Conf. on Security Technology, (1993) 41–43

[5] Lukac, R.: Adaptive impulse noise filtering by using center-weighted directional information. Proc. CGIV'02 (2002) 86–89

[6] Man, K.F., Tang, K.S., and Kwong, S.: Genetic algorithms: concept and design. Springer Verlag, London (1999)

[7] Plataniotis, K.N., Venetsanopoulos, A.N.: Color image processing and applications. Springer Verlag (2000)

Systolic Architectures for Soft Computing Algorithms

Jarosław Bilski[1], Jacek Smoląg[1], and Jacek Żurada[2]

[1] Department of Computer Engineering, Technical University of Czestochowa,
Al. Armii Krajowej 36, 42-200 Czestochowa, POLAND,
{bilski,jsmolag}@kik.pcz.czest.pl,
http://kik.pcz.czest.pl
[2] Department of Electrical and Computer Engineering University of Louisville, USA

Abstract. In the last decade various soft computing techniques have been developed. They include neural networks, fuzzy systems, evolutionary algorithms, rough sets and others. In many applications it is desirable that soft computing techniques are implemented in parallel VLSI structures based on systolic arrays. In this paper we present the systolic implementations of the UD RLS learning algorithms for feed-forward neural networks and for probabilistic neural networks.

1 Introduction

In the last decade various soft computing techniques have been developed. They include neural networks, fuzzy systems, evolutionary algorithms, rough sets and others [18, 20, 26, 27, 29,]. Most of them are available in the form of software packages e.g. MatLab or Mathematica. In many applications it is desirable that soft computing techniques are implemented in parallel VLSI structures. In this paper we derive systolic architectures for two soft computing algorithms. A systolic array is a network of processors which rhythmically compute and pass data through the system. They are well suited to VLSI implementation, because of their properties of regularity, modularity, pipelining, and local interconnection.

In section 2 a systolic implementation of the UD RLS learning algorithms for feed-forward neural networks is presented. In section 3 a systolic implementation of probabilistic neural networks is studied. It should be noted that both systolic realizations are applicable to a variety classification, prediction and modeling problems.

2 A Systolic Implementation of the UD RLS Learning Algorithms for Feed-Forward Neural Networks

In this paper we use the same notation as that in [25] and [26]. Several fast learning algorithms based on the RLS method [21] have been studied in [25, 26], whereas systolic implementations have been proposed in [22, 23, 24, 26]. In

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 601–608, 2004.
© Springer-Verlag Berlin Heidelberg 2004

this paper we develop systolic architectures for implementation of the UD RLS algorithm. The matrix P in the classical RLS algorithm [25] is defined as follows:

$$\mathbf{P}_i^{(k)}(n) = \mathbf{U}_i^{(k)}(n)\,\mathbf{D}_i^{(k)}(n)\,\mathbf{U}_i^{(k)T}(n) \tag{1}$$

where

$$\mathbf{U}_i^{(k)}(n) = \begin{bmatrix} 1 & u_{i,\,01}^{(k)}(n) & u_{i,\,02}^{(k)}(n) & \cdots & u_{i,\,0N_0}^{(k)}(n) \\ 0 & 1 & u_{i,\,12}^{(k)}(n) & \cdots & u_{i,\,1N_0}^{(k)}(n) \\ 0 & 0 & 1 & & \vdots \\ \vdots & \vdots & \vdots & \ddots & u_{i,\,N_0-1N_0}^{(k)}(n) \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2}$$

and

$$\mathbf{D}_i^{(k)}(n) = \begin{bmatrix} c_{i,\,0}^{(k)} & 0 & \cdots & 0 \\ 0 & c_{i,\,1}^{(k)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{i,\,N_0}^{(k)} \end{bmatrix} \tag{3}$$

In the above description $\mathbf{U}_i^{(k)}(n)$ is a triangular matrix with zeros in bottom elements and ones in the main diagonal, $.\mathbf{D}_i^{(k)}(n)$ is a diagonal matrix.

This leads to the following UD RLS algorithm (4 – 15):

Step 1

$$\varepsilon_i^{(k)}(n) = f'\left(s_i^{(k)}(n)\right) b_i^{(k)}(n) - \mathbf{x}^{(k)T}(n)\,\mathbf{w}_i^{(k)}(n-1) \approx d_i^{(k)}(n) - y_i^{(k)}(n) \tag{4}$$

$$\mathbf{f}_i^{(k)} = \mathbf{U}_i^{(k)T}(n-1)\,\mathbf{x}^{(k)}(n) \tag{5}$$

$$\mathbf{h}_i^{(k)} = \mathbf{D}_i^{(k)}(n-1)\,\mathbf{f}_i^{(k)} \tag{6}$$

$$\beta_{i,-1}^{(k)} = \lambda \tag{7}$$

Step 2

For $j = 0$ to N_{k-1}

$$\beta_{ij}^{(k)} = \beta_{i,\,,j-1}^{(k)} = +f'^2\left(s_i^{(k)}(n)\right) f_{ij}^{(k)} h_{ij}^{(k)} \tag{8}$$

$$c_{ij}^{(k)}(n) = c_{ij}^{(k)}(n-1)\,\frac{\beta_{i,j-1}^{(k)}}{\beta_{ij}^{(k)}\lambda} \tag{9}$$

$$k_{ij}^{(k)} = h_{ij}^{(k)} \tag{10}$$

$$\mu_{ij}^{(k)} = -f'^2\left(s_i^{(k)}(n)\right)\frac{f_{ij}^{(k)}}{\beta_{i,j-1}^{(k)}} \tag{11}$$

Step 2.1
For $m = 0$ to $j-1 (j > 0)$

$$u_{imj}^{(k)}(n) = u_{imj}^{(k)}(n-1) + \mu_{ij}^{(k)} k_{im}^{(k)} \tag{12}$$

$$k_{im}^{(k)} = k_{im}^{(k)} + u_{imj}^{(k)}(n-1)\, k_{ij}^{(k)} \tag{13}$$

Step 3

$$\mathbf{g}_i^{(k)}(n) = \frac{\left[k_{i0}^{(k)}, \ldots, k_{iN_{k-1}}^{(k)}\right]^T}{\beta_{iN_{k-1}}} \tag{14}$$

$$\mathbf{w}_i^{(k)}(n) = \mathbf{w}_i^{(k)}(n-1) + \mathbf{g}_i^{(k)}(n)\,\varepsilon_i^{(k)}(n) \tag{15}$$

The initial conditions are given by

$$\mathbf{D}_{(i)}^{(k)}(0) = \delta\mathbf{I}, \quad \delta > 0 \tag{16}$$

$$\mathbf{U}_{(i)}^{(k)}(0) = \mathbf{I} \tag{17}$$

where I is the identity matrix and δ is a constant. The initial values of the weight vector $\mathbf{w}_i^{(k)}(n)$ can be chosen as random numbers

The systolic architecture design begins with data dependence graph (DG) to express the recurrence and parallelism [12, 16]. A dependence graph presents the data dependencies of an algorithm. The systolic implementation of the recall phase of algorithm (3) - (15) is the same as that studied in [23]. Therefore, we shall concentrate on the learning phase. All types of operations required for learning phase can be efficiently implemented by the systolic array. Systolic array for learning phase of the UD RLS algorithm is shown in Fig. 1. In this structure we have three types of processing elements. Their dedicated construction can realise formulas (5), (6 - 11) and (12 - 14).

3 A Systolic Implementation of the Generalized Probabilistic Neural Network

In the literature various probabilistic neural networks have been proposed. They are a network reprezentation of nonparametric density estimators [1, 3, 10] and

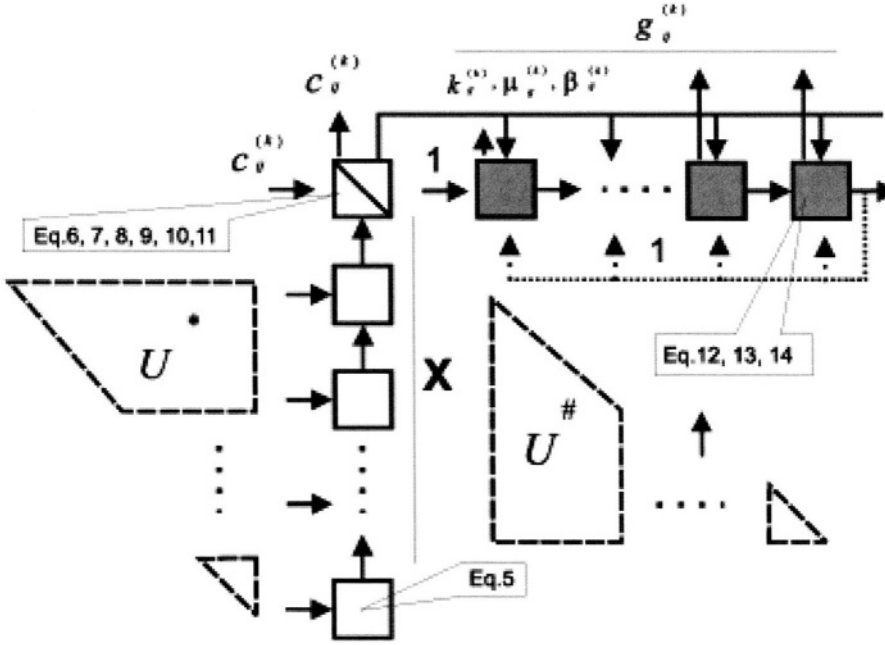


Fig. 1. Systolic array for learning phase of the UD RLS algorithm.

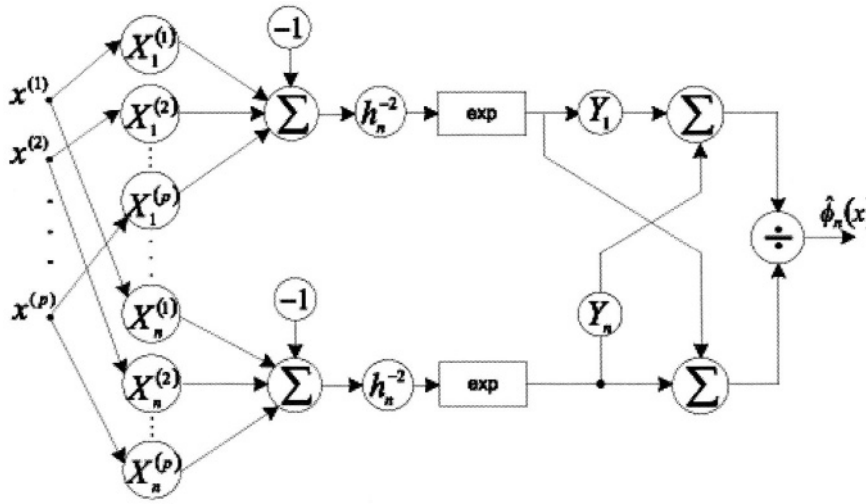


Fig. 2. Generalized regression neural network based on Parzen's kernel.

nonparametric regression estimators [2, 5, 11, 17, 19]. Moreover, they work in stationary [13], quasi-stationary [4, 9] and nonstationary [6, 7, 8, 14, 15, 28] environments. Probabilistic neural networks are based on the Parzen kernel taking

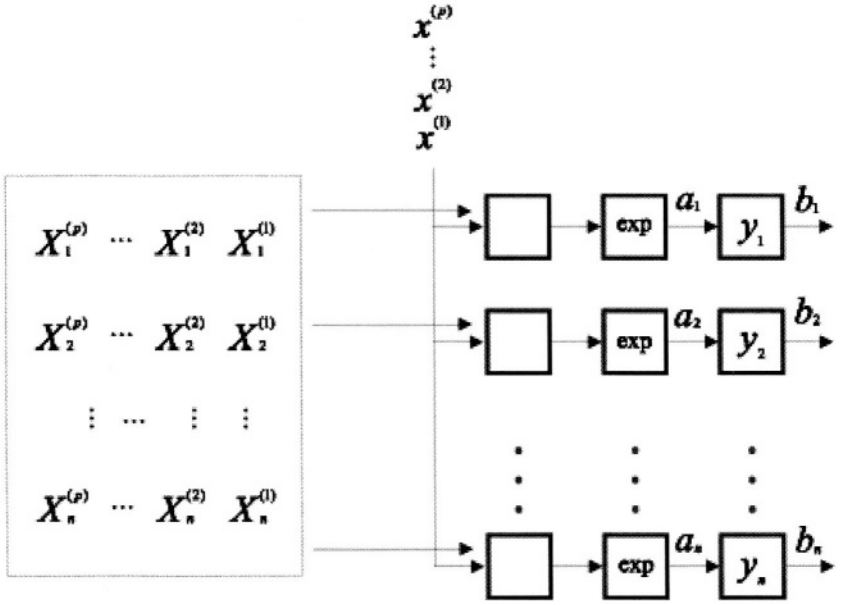


Fig. 3. Systolic implementation of generalized regression neural network based on Parzen's kernel – Part 1.

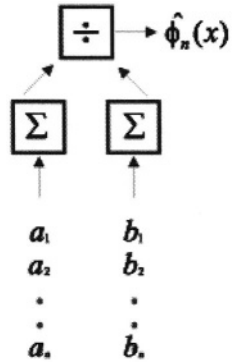


Fig. 4. Systolic implementation of generalized regression neural network based on Parzen's kernel – Part 2.

the following form in the multidimensional case:

$$K(x) = \prod_{i=1}^{p} H\left(x^{(i)}\right) \tag{18}$$

The most popular is the Gaussian kernel

$$H(\nu) = (2\pi)^{-\frac{1}{2}} e^{-\frac{1}{2}\nu^2} \tag{19}$$

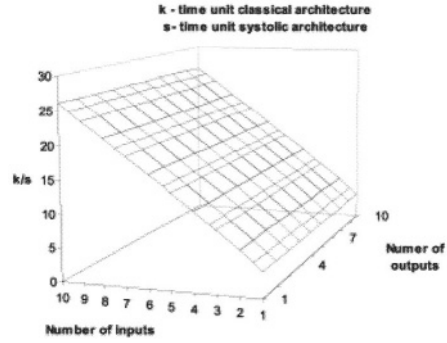


Fig. 5. Performance evaluation for learning phase of the UD RLS algorithm.

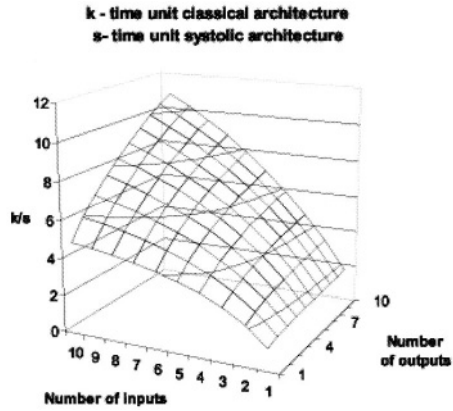


Fig. 6. Performance evaluation of systolic architecture versus serial computation.

Let $(X,\ Y)$, (X_1, Y_1), (X_2, Y_2), ... (X_n, Y_n) be a seqence of i.i.d. random variables. We wish to estimate the regression function $\phi(x) = E[Y|X = x]$. As the estimator we take the following formula [26]

$$\hat{\phi}_n(x) = \frac{\sum_{i=1}^{n} Y_i K\left(\frac{x - X_i}{h_n}\right)}{\sum_{i=1}^{n} K\left(\frac{x - X_i}{h_n}\right)} \tag{20}$$

where h_n is a certain sequence of numbers. Generalized regression neural network based on the Parzen kernel (18) has been derived in [26] and is shown in Fig 2.

Figures 3 and 4 present systolic implementations of probabilistic neural network described by formula (20).

4 Performance Evaluation

In this section we analyse the performance evaluation of systolic architectures developed in previous sections. We compare the systolic implementation with classical sequential architectures. For the UC RLS algorithm we consider single – layered neural networks. We assume that all operations, eg. multiplications or additions, take the same unit time. Performance of the UC RLS algorithm is illustrated in Fig. 5.

We observe that systolic architectures outperform classical architectures, even 25 times for the single–layered network having 10 neurons with 10 inputs. The analogous performance evaluation for probabilistic neural networks is presented in Fig. 6.

References

1. Rutkowski L. (1980): *Sequential estimates of probability densities by orthogonal series and their application in pattern classification,* IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, No 12, pp. 918-920.
2. Rutkowski L. (1981): *Sequential estimates of a regression function by orthogonal series with applications in discrimination,* in: Lectures Notes in Statistics, Vol. 8, pp. 236-244, New York.
3. Greblicki W. and Rutkowski L.(1981): *Density-free Bayes risk consistency of nonparametric pattern recognition procedures,* Proceedings of the IEEE, Vol. 69, No 4, pp. 482-483.
4. Rutkowski L. (1982): *On Bayes risk consistent pattern recognition procedures in a quasi-stationary environment,* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, No 1, pp. 84-87.
5. Rutkowski L (1982): *On system identification by nonparametric function fitting,* IEEE Transactions on Automatic Control, Vol. AC-27, pp. 225-227.
6. Rutkowski L. (1982): *On-line identification of time-varying systems by nonparametric techniques,* IEEE Transactions on Automatic Control, Vol. AC-27, pp. 228-230.
7. Rutkowski L. (1984): *On nonparametric identification with prediction of time-varying systems,* IEEE Transactions on Automatic Control, Vol. AC-29, pp. 58-60.
8. Rutkowski L. (1985): *The real-time identification of time-varying systems by nonparametric algorithms based on the Parzen kernels,* International Journal of Systems Science, Vol. 16, pp. 1123-1130, London.
9. Rutkowski L. (1985): *Nonparametric identification of quasi-stationary systems,* Systems and Control Letters, Vol. 6, pp. 33-35, Amsterdam.
10. Rutkowski L. (1988): *Sequential pattern recognition procedures derived from multiple Fourier series,* Pattern Recognition Letters, Vol. 8, pp. 213-216.
11. Rutkowski L. (1988): *Nonparametric procedures for identification and control of linear dynamic systems,* Proceedings of 1988 American Control Conference, June 15-17, pp. 1325-1326.
12. Kung S. Y.(1988): *VLSI Array Processors,* Prentice Hall
13. Rutkowski L. and Rafajłowicz E. (1989): *On global rate of convergence of some nonparametric identification procedures,* IEEE Transaction on Automatic Control, Vol. AC-34, No.10, pp. 1089-1091.

14. Rutkowski L. (1989): *Nonparametric learning algorithms in the time-varying environments,* Signal Processing, Vol. 18, pp. 129-137.
15. Rutkowski L. (1989): *An application of multiple Fourier series to identification of multivariable nonstationary systems,* International Journal of Systems Science, Vol. 20, No. 10, pp. 1993-2002.
16. Hwang J. N. , Kung S. Y. (1989): *Parallel Algorithms/Architectures for Neural Networks,* Journal of VLSI Signal Processing, Vol.1, pp.221-251.
17. Rutkowski L. (1991): *Identification of MISO nonlinear regressions in the presence of a wide class of disturbances,* IEEE Transactions on Information Theory, Vol. IT-37, pp. 214-216.
18. Żurada J. (1992): *Introduction to Artificial Neural Systems,* West Publishing Company.
19. Rutkowski L. (1993): *Multiple Fourier series procedures for extraction of nonlinear regressions from noisy data,* IEEE Transactions on Signal Processing, Vol. 41, No.10, pp. 3062-3065.
20. Tadeusiewicz R. (1993): *Neural Networks,* Akademicka Oficyna Wydawnicza, Warszawa (in Polish).
21. Rutkowski L. (1994): *Adaptive Signal Processing: Theory and Applications* (in Polish), WNT
22. Smolag J., Rutkowski L. (1996): *A systolic architecture for fast training of feed-forward neural networks,* Proceedings of the Second Conference on Neural Networks and Their Applications, Szczyrk, pp.426-432
23. Smolag J., Rutkowski L., Bilski J. (1997): *Systolic Architectures for Neural Networks, Part I,* Proceedings of the Third Conference on Neural Networks and Their Applications, Kule, pp.614-621
24. Smolag J., Rutkowski L., Bilski J.(1997): *Systolic Architectures for Neural Networks, Part II,* Proceedings of the Third Conference on Neural Networks and Their Applications, Kule, pp.622-625
25. Bilski J., Rutkowski L.(1998): *A fast training algorithm for neural networks,* IEEE Transactions on Circuits and Systems, Part II, Vol. 45 nr 6, pp. 749-753, June
26. Rutkowski L. (2004): *New Soft Computing Techniques for System Modelling,* Pattern Classification and Image Processing, Springer-Verlag
27. Rutkowski L. and Cpałka K. (2003): *Flexible neuro-fuzzy systems,* IEEE Transactions on Neural Networks, vol. 14, pp. 554-574
28. Rutkowski L. (2004): *Adaptive probabilistic neural-networks for pattern classification in time-varying environment,* IEEE Transactions on Neural Networks, vol. 15, March
29. Rutkowski L. (2004): *Flexible Neuro-Fuzzy Systems: Structures, Learning and Performance Evaluation,* Kluwer

Image Compression Based on Soft Computing Techniques

Robert Cierniak

Department of Computer Engineering, Czestochowa University of Technology,
Al. Armii Krajowej 36, 42-200 Czestochowa, POLAND,
cierniak@kik.pcz.czest.pl

Abstract. In this paper a new algorithm for image compression, named predictive vector quantization (PVQ), is developed based on competitive neural networks quantizer and neural networks predictor. The modified closed-loop PVQ methodology is developed. The experimental results are presented and the performance of the algorithm is discussed. A comparison of two feed-forward neural network structures applied for predictor is discussed.

1 Introduction

In the literature several methods for image compression have been proposed. Among them the vector quantization (VQ) technique has emerged as an effective tool in this area of research [5],[14]. A special approach to image compression combines the VQ technique with traditional (scalar) differential pulse code modulation (DPCM) leading to the predictive vector quantization (PVQ). The design of the PVQ scheme requires both a predictor and a VQ codebook determination. Gersho and Grey [14] outlined three approaches to the design problem: open-loop, closed-loop and semi-closed-loop design methodology.

In this paper we propose a novel approach to predictive vector quantization. We develop the so-called modified closed-loop methodology which outperforms three approaches listed above. The vector quantizer will be based on competitive neural networks, whereas the predictor will be designed as the nonlinear neural network - contrary to a heuristic method presented in [5] and linear predictors developed in our previous papers [15],[19]. Our methodology is described in section 3, whereas experimental results are presented in section 4.

2 Preprocessing

In the sequel we assume that an image is represented by an $N_1 \times N_2$ array of pixels $y_{n_1 n_2}, n_1 = 1, 2, ..., N_1,\ n_2 = 1,\ 2, ..., N_2$. The image is portioned into contiguous small blocks of the dimension $n_1 \times n_2$.

$$\mathbf{Y}(m_1, m_2) = \begin{bmatrix} y_{11}(m_1, m_2) \cdots y_{1,n_2}(m_1, m_2) \\ \vdots \quad \ddots \quad \vdots \\ y_{n_1,1}(m_1, m_2) \cdots y_{n_1,n_2}(m_1, m_2) \end{bmatrix}, \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 609–617, 2004.
© Springer-Verlag Berlin Heidelberg 2004

where: $\mathbf{Y}(m_1, m_2)$, $m_1 = 1, 2, ..., N_1/n_1$, $m_2 = 1, 2, ..., N_2/n_2$. The arrays (1) will be represented by the corresponding vectors

$$\mathbf{V}(m_1, m_2) = [v_1(m_1, m_2), v_2(m_1, m_2), ..., v_q(m_1, m_2)]^T, \tag{2}$$

where we identify: $q = n_1 \cdot n_2$, $m_1 = 1, 2, ..., M_1$, $m_2 = 1, 2, ..., M_2$, $v_1(m_1, m_2) = y_{11}(m_1, m_2)$,...,$v_q(m_1, m_2) = y_{n_1, n_2} m_1, m_2$. That means that the original image is represented by $\frac{N_1 \cdot N_2}{q}$.vectors $\mathbf{V}(m_1, m_2)$. The successive input vectors to the encoder $\mathbf{V}(t)$, $t = 1, 2, ..., \frac{N_1 \cdot N_2}{q}$ (see Fig. 1), correspond to vectors $\mathbf{V}(m_1, m_2)$ in the line-by-line order.

3 Neural PVQ Compression Algorithm

The general architecture of the predictive vector quantization algorithm (PVQ) is depicted in Fig.1. This architecture is a straightforward vector extension of the traditional (scalar) differential pulse code modulation (DPCM) scheme [15],[16],[19].

The block diagram of the PVQ algorithm consists of an encoder and decoder, each containing an identical predictor, codebook and vector quantizer. The successive input vectors $\mathbf{V}(t)$ are introduced to the encoder and the difference $\mathbf{E}(t) = [e_1(t), e_2(t), ..., e_q(t)]^T$ given by the equation

$$\mathbf{E}(t) = \mathbf{V}(t) - \overline{\mathbf{V}}(t) \tag{3}$$

is formed, where: $\overline{\mathbf{V}}(t) = [\overline{v}_1(t), \overline{v}_2(t), ..., \overline{v}_q(t)]^T$ is the predictor of $\mathbf{V}(t)$.

As in the scalar DPCM, the difference $\mathbf{E}(t)$ requires fewer quantization bits than the original subimage $\mathbf{V}(t)$. The next step is vector quantization of $\mathbf{E}(t)$. Mathematically, the vector quantization can be viewed as a mapping VQ from the q-dimentional Euclidean space $\Re^q$ into a finite subset $\mathbf{G}$ of $\Re^q$, where $\mathbf{G} = [\mathbf{g}_0, \mathbf{g}_1, ..., \mathbf{g}_J]$ is the set of reproduction vectors (codewords or codevectors) and $\mathbf{g}_j = [g_{1j}, g_{2j}, ..., g_{qj}]^T$. The subset $\mathbf{G} \subset \Re^q$ is commonly called the codebook. For every q-dimensional difference vector $\mathbf{E}(t)$, the distortion (usually the mean square error) between $\mathbf{E}(t)$ and every codeword $\mathbf{g}_j$, $j = 0, 1, ..., J$ is computed. The codeword $\mathbf{g}_{j^0}(t)$ is selected as the representation vector for $\mathbf{E}(t)$ if

$$d_{j^0} = \min_{0 \le j \le J} d_j, \tag{4}$$

a measure d in expression (4) we can take e.g. the Euclidean distance

$$d = \sqrt{\sum_{i=1}^{q} [e_i(t) - g_{ij}]^2}. \tag{5}$$

The index $j^0(t)$ is broadcast via the transmission channel to the decoder. Observe that by adding the prediction vector $\overline{\mathbf{V}}(t)$ to the quantized difference

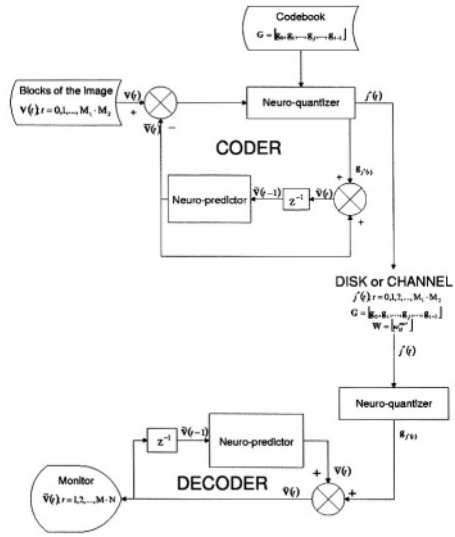


Fig. 1. The architecture of the compression algorithm

vector $\mathbf{g}_{j^0}(t)$ we get the reconstructed approximation $\widetilde{\mathbf{V}}(t)$ of the original input vector $\mathbf{V}(t)$, i.e.

$$\widetilde{\mathbf{V}}(t) = \overline{\mathbf{V}}(t) + \mathbf{g}_{j^0}(t). \tag{6}$$

As a measure of error between the original and reconstructed images one can take the mean square error

$$MSE = \frac{1}{N_1 N_2} \sum_{n_1=1}^{N_1} \sum_{n_2=1}^{N_2} (y_{n_1 n_2} - \widetilde{y}_{n_1 n_2}). \tag{7}$$

where: $\widetilde{y}_{n_1 n_2}, n_1 = 1, 2, ..., N_1,\ n_2 = 1, 2, ..., N_2$, stand for pixels of the reconstructed image. The prediction vector $\overline{\mathbf{V}}(t)$ of the input vector $\mathbf{V}(t)$ is made from past observation of reconstructed vector $\widetilde{\mathbf{V}}(t-1)$. The predictor in our approach is a nonlinear neural network specially designed for this purpose.

It should be noted that alternatively it is possible to apply other soft computing techniques [25] to design predictor, e.g. probabilistic neural networks [1],[3],[7],[10],[12],[11] or neuro-fuzzy systems [18],[20],[21],[22],[23],[24]. The probabilistic neural networks are applicable in the stationary or non-stationary environment [2],[4],[6],[8],[9],[13].

In the decoder, the index $j^0(t)$ transmitted by the channel is inverse vector-quantized and the reconstructed vector $\widetilde{\mathbf{V}}(t)$ is formed in the same manner as in the encoder (see formula (6)).

The modified closed-loop predictive vector quantization procedure. The design of a predictive vector quantization scheme requires both a predictor and a codebook design.

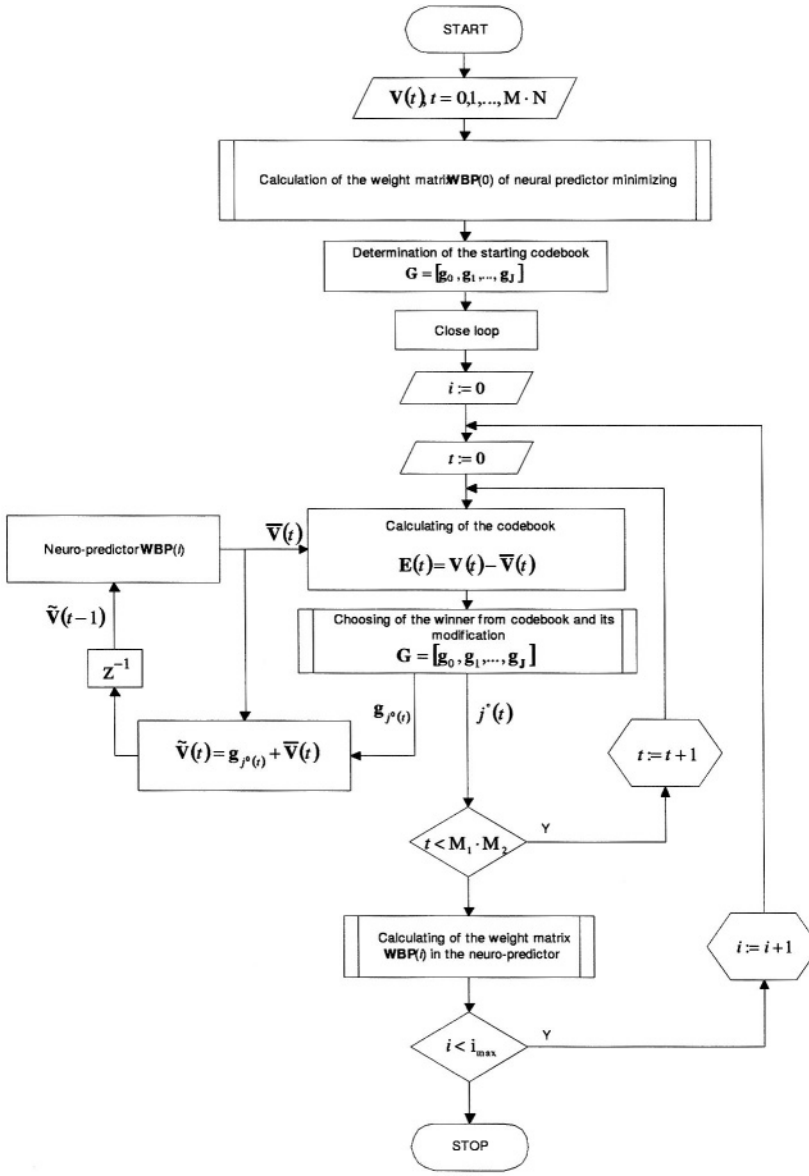


Fig. 2. Diagram of the compression algorithm design

In this paper we use the modified closed-loop methodology as shown in Fig.2. This approach contains several steps:

a)Design of the initial neural predictor based on the residuals;

b) Generation of an initial codebook using open-loop methodology;

c) Sequentially performed procedure of the new codebook calculation using unsupervised learning algorithm and then adjustment of the neural predictor through the supervised learning.

The process is stopped after achieving the satisfied low level of the compression error.

Predictor based on neural networks. The supervised neural network is proposed to design a nonlinear predictor [17]. The first structure is a three layer network with forward propagation This neural network is designed as follows: number of inputs and outputs is equal $q = 16$, the structure of the net is 16-8-16 neurons in particular layers, and the sigmoid activation function of the neurons in the output layer is scaled by 255. The second neural network structure is a single layer network with 8 neurons In both cases, during the learning process we attempt to minimize the mean square error taking the following form

$$Q = \sum_{t=1}^{M_1 M_2} \sum_{i=1}^{q} \left(y_i(t) - v_i(t)\right)^2, \tag{8}$$

where $y_i(t)$ is the k-th neural network output. Obviously, the input of the net is $v_i(t-1)$; $i = 1, 2, ..., q$. The weights matrix $\mathbf{WBP}(s)$ can be found by the back-propagation method.

Generation of an initial codebook. We will find the codebook $\mathbf{G}(0) = [\mathbf{g}_0, \mathbf{g}_1, ..., \mathbf{g}_J]$, $\mathbf{g}_j = [g_{1j}, g_{2j}, ..., g_{qj}]^T$, $j = 0, 1, ..., J$, where J + 1= size of the code-book, minimizing the performance measure

$$D = \sum_{t=1}^{M_1 M_2} d\left[\mathbf{E}(t), \mathbf{g}_{j^0}(t)\right]^2, \tag{9}$$

where:

$$d\left[\mathbf{E}(t), \mathbf{g}_{j^0}(t)\right] = \min_{0 \le j \le J} \left\{d\left[\mathbf{E}(t), \mathbf{g}_j\right]\right\}, \tag{10}$$

and d is the distortion (usually chosen as the mean square error) between the vector $\mathbf{E}(t)$ and the code vector $\mathbf{g}_j$The code-vector $\mathbf{g}_{j^0}(t)$ with the minimum distortion is called the "winner". Because of its properties, the neural network is able to create the code-book $\mathbf{G}$ as a result of learning. For this purpose we can use a single-layer neural network.

The elements of the input vector $\mathbf{E}(t) = [e_1(t), e_2(t), ..., e_q(t)]^T$ are connected to every neural unit having the weights $\mathbf{W}_j(0) = [w_{1j}(0), w_{2j}(0),$

Fig. 3. Lena "original" image

Fig. 4. The reconstructed "Lena" image

$..., w_{qj}(0)]^T$ and the output z_j, $j = 0, 1, ..., J$. The weights $\mathbf{W}_j$ are considered to be the code-vectors, i.e.

$$\mathbf{G}(0) = [\mathbf{g}_0(0), \mathbf{g}_1(0), ..., \mathbf{g}_J(0)] = [\mathbf{W}_1(0), \mathbf{W}_2(0), ..., \mathbf{W}_J(0)], \tag{11}$$

and the number of neural units J+1 is the size of the codebook.

The weights $\mathbf{W}_j(0)$ will be determined by making use of the unsupervised neural networks. We used the frequency-sensitive competitive learning (FSCL) network. In the FSCL network [15],[19] the winning neural unit $j^0(t)$ is selected

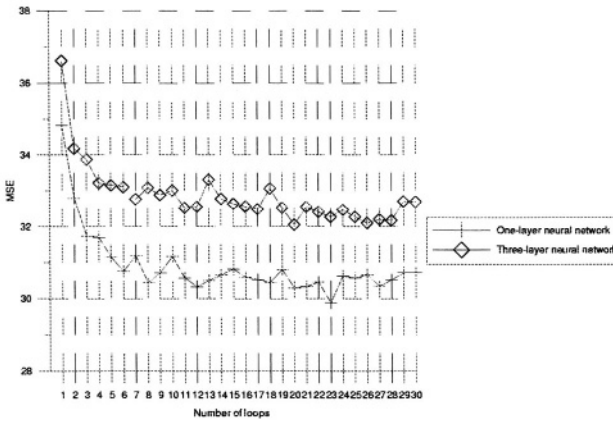


Fig. 5. The comparison of experimental results using two types of neural networks

on the basis of a modified distortion measure

$$d\left[\mathbf{E}\left(t\right),\mathbf{W}_{j}\left(0\right)\left(t\right)\right]=F\left(f_{j}\right)\sqrt{\sum_{i=1}^{q}\left[e_{i}\left(t\right)-w_{ij}\left(0\right)\left(t\right)\right]^{2}},\tag{12}$$

where F is a suitably chosen function of the counter f_j. The counter f_j counts how frequently the neural unit j is the "winner". The recursive procedure takes the form

$$\mathbf{W}_{j^{0}}\left(0\right)\left(t+1\right)=\mathbf{W}_{j^{0}}\left(0\right)\left(t\right)+H\left(f_{j^{0}}\right)\left(\mathbf{E}\left(t\right)-\mathbf{W}_{j^{0}}\left(0\right)\left(t\right)\right),\tag{13}$$

where H is another function of the counter f_{j^0}.

Recursive generation of the codebook and the predictor. After determination of the codebook $\mathbf{G}\left(0\right)=\left[\mathbf{g}_{0}\left(0\right),\mathbf{g}_{1}\left(0\right),...,\mathbf{g}_{J}\left(0\right)\right]$ using the open-loop design we will compute the set of the vectors $\mathbf{E}\left(t\right)$ using the closed-loop scheme. This set is necessary to design the new codebook $\mathbf{G}$ and then the new nonlinear neural predictor. Again we apply the competitive neural network (described above) in order to find the next codebook $\mathbf{G}\left(s\right)$ and the supervised neural network (presented in this section) to design the new predictor, and so on. This sequential process is stopped after some iteration when the satisfied low level of the compression error is achieved.

4 Experimental Results

The tested image was a standard picture "Lena" ($N_1 \times N_2 = 512 \times 512$ frame of size, 256 grey levels for each pixel and blocks of image 4×4 pixels) as shown in Fig.3. The experiment shows realisation of the PVQ design: with nonlinear neural

predictors described in this paper. In Fig. 4 we show the reconstructed "Lena" image for the code-book size = 512, twenty loops and the FSCL algorithm. In this case MSE = 32,70 and SNR = 23.33. In Fig 5 we present the MSE versus number of loops s.

References

[1] Rutkowski L. (1981): *Sequential estimates of a regression function by orthogonal series with applications in discrimination,* in: Lectures Notes in Statistics, **8** (1981) 236-244, New York.

[2] Rutkowski L.: *On Bayes risk consistent pattern recognition procedures in a quasi-stationary environment,* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, **1** (1982) 84-87.

[3] Rutkowski L: *On system identification by nonparametric function fitting,* IEEE Transactions on Automatic Control, **AC-27** (1982) 225-227.

[4] Rutkowski L.: *On-line identification of time-varying systems by nonparametric techniques,* IEEE Transactions on Automatic Control, **AC-27** (1982) 228-230.

[5] Gray R.: Vector quantization. IEEE ASSP Magazine (1984) 4-29

[6] Rutkowski L.: *On nonparametric identification with prediction of time-varying systems,* IEEE Transactions on Automatic Control, **AC-29** (1984) 58-60.

[7] Gałkowski T., and Rutkowski L.: *Nonparametric recovery of multivariate functions with applications to system identification,* Proceedings of the IEEE, **73** (1985) 942-943, New York.

[8] Rutkowski L.: *Nonparametric identification of quasi-stationary systems,* Systems and Control Letters, **6** (1985) 33-35, Amsterdam.

[9] Rutkowski L.: *The real-time identification of time-varying systems by nonparametric algorithms based on the Parzen kernels,* International Journal of Systems Science, **16** (1985) 1123-1130, London.

[10] Gałkowski T. and Rutkowski L.: *Nonparametric fitting of multivariable functions,* IEEE Transactions on Automatic Control, **AC-31** (1986) 785-787.

[11] Kozietulski M. and Rutkowski L.: *A nonparametric procedure for identification of the step response function and its microprocessor implementation,* Advances in Modelling and Simulation, **17** (1989) 25-36.

[12] Rutkowski L. and Rafajłowicz E. : *On global rate of convergence of some nonparametric identification procedures,* IEEE Transaction on Automatic Control, **AC-34** (1989) 1089-1091.

[13] Rutkowski L.: *Nonparametric learning algorithms in the time-varying environments,* Signal Processing, **18** (1989) 129-137.

[14] Gersho A., Gray R. M.: Vector quantization a. signal compression, Kluwer Academic Publishers (1992)

[15] Rutkowski L., Cierniak R.: Image compression by competitive learning neural networks and predictive vector quantization, Applied Mathematics and Computer Science. **6** (1996)

[16] Luo Fa-Long, Unbehauen R.: Applied Neural Networks for Signal Processing. Cabridge University Press (1998)

[17] Manikopoulos C. N.: Neural networks approach to DPCM system designe for image coding. IEE Proceedings-I (1992)

[18] Rutkowski L. and Cpałka K.: *Flexible Structures of Neuro - Fuzzy Systems,* Quo Vadis Computational Intelligence, Studies in Fuzziness and Soft Computing, **54** (2000) 479 - 484, Springer.

[19] Cierniak R., Rutkowski L.: On image compression by competitive neural networks and optimal linear predictors. Signal Processing: Image Communication - a Eurosip Journal, Elsevier Science B. V., **15** (2000) 559-565

[20] Starczewski J. and Rutkowski L.: *Connectionist Structures of Type 2 Fuzzy Inference Systems,* Lecture Notes in Computer Science, **2328** (2001) 634-642.

[21] Rutkowski L. and Cpałka K.: *A neuro-fuzzy controller with a compromise fuzzy reasoning,* Control and Cybernetics, **31** (2002) 297-308.

[22] Scherer R. and Rutkowski L. (2002): *Neuro-fuzzy relational systems,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.

[23] Starczewski J. and Rutkowski L. (2002): *Neuro-fuzzy inference systems of type 2,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.

[24] Rutkowski L. and Cpałka K. (2003): *New Soft Computing Techniques for System Modelling, Pattern Classification and Image Processing,* Springer-Verlag.

[25] Rutkowski L. (2004): *Flexible neuro-fuzzy systems,* IEEE Transactions on Neural Networks, vol. 14, pp. 554-574.

A Flexible Connectionist Fuzzy System

Krzysztof Cpałka

Department of Computer Engineering, Czestochowa University of Technology,
Al. Armii Krajowej 36, 42-200 Czestochowa, POLAND,
cpalka@kik.pcz.czest.pl

Abstract. In this paper we derive flexible neuro-fuzzy systems based on Yager's triangular norms. We incorporate various flexibility parameters into their construction. The parameters are learned by the standard recursive gradient procedures with constraints. The performance is illustrated on a typical approximation problem.

1 Introduction

In the literature various neuro-fuzzy systems have been proposed. They include Mamdani type systems [10], [11], [13], [15], [16], [25], logical type systems [12], [28], relational neuro-fuzzy systems [14], [31], [33], [35], [36], [32], type 2 neuro-fuzzy systems [29], [30], [37], [38], [39], rough neuro-fuzzy systems [4] and hierarchical neuro-fuzzy [6], [7], [8], [34]. Most of neuro-fuzzy structures are based on classical triangular norms, e.g., min max or algebraic. However, it is well known that incorporating additional parameters to be tuned in the system improves its performance. Therefore, in this paper we present flexible neuro-fuzzy systems based on Yager's triangular norms [3]:

$$\overset{\leftrightarrow}{T}_Y \{a_1, a_2; p\} = 1 - \min\left\{1, \sqrt[p]{(1-a_1)^p + (1-a_2)^p}\right\} \tag{1}$$

$$\overset{\leftrightarrow}{S}_Y \{a_1, a_2; p\} = \min\left\{1, \sqrt[p]{a_1^p + a_2^p}\right\} \tag{2}$$

where $p \in [0,\infty)$. The parameter p can be found in the process of learning. Following [23] we propose a compromise fuzzy implication given by

$$I(a,b) = N(\lambda)\, T\{a,b\} + \lambda S\{N(a), b\} \tag{3}$$

where $\lambda \in [0,1]$, $N(\lambda) = 1 - \lambda$. The parameter λ, determining a type of the system, can be also found in the process of learning. We will use soft versions of Yager's triangular norms in the form

$$\tilde{T}\{\mathbf{a}; \alpha\} = (1-\alpha)\frac{1}{n}\sum_{i=1}^{n} a_i + \alpha \overset{\leftrightarrow}{T}\{\mathbf{a}\} = N(\alpha)\, avg(\mathbf{a}) + \alpha \overset{\leftrightarrow}{T}\{\mathbf{a}\} \tag{4}$$

and

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 618–625, 2004.
© Springer-Verlag Berlin Heidelberg 2004

$$\tilde{S}\{\mathbf{a};\alpha\} = (1-\alpha)\frac{1}{n}\sum_{i=1}^{n} a_i + \alpha\overleftrightarrow{S}\{\mathbf{a}\} = N(\alpha)\, avg(\mathbf{a}) + \alpha\overleftrightarrow{S}\{\mathbf{a}\} \tag{5}$$

where $\alpha \in [0,1]$.

2 Flexible Neuro-Fuzzy System Based on Yager's Triangular Norms

In this paper, we consider multi input, single output NFIS mapping $\mathbf{X} \to \mathbf{Y}$, where $\mathbf{X} \subset \mathbf{R}^n$ and $\mathbf{Y} \subset \mathbf{R}$. The fuzzifier performs a mapping from the observed crisp input space $\mathbf{X} \subset \mathbf{R}^n$ to the fuzzy sets defined in $\mathbf{Y}$. The most commonly used fuzzifier is the singleton fuzzifier which maps $\bar{\mathbf{x}} = [\bar{x}_1, \ldots, \bar{x}_n] \in \mathbf{X}$ into a fuzzy set $A' \subset \mathbf{X}$ characterized by the membership function:

$$\mu_{A'}(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{x} = \bar{\mathbf{x}} \\ 0 \text{ if } \mathbf{x} \neq \bar{\mathbf{x}} \end{cases} \tag{6}$$

The fuzzy rule base consists of a collection of N fuzzy IF THEN rules in the form:

$$R^{(k)}\text{: IF}\mathbf{x} \text{ is} A^k \text{ THEN} y \text{ is} B^k \tag{7}$$

where $\mathbf{x} = [x_1, \ldots, x_n] \in \mathbf{X}$, $y \in \mathbf{Y}$, $A_1^k, A_2^k, \ldots, A_n^k$ are fuzzy sets characterized by membership functions $\mu_{A_i^k}(x_i)$, whereas B^k are fuzzy sets characterized by membership functions $\mu_{B^k}(y)$, respectively, $k = 1, \ldots, N$. The fuzzy inference determines a mapping from the fuzzy sets in the input space $\mathbf{X}$ to the fuzzy sets in the output space $\mathbf{Y}$. Each of N rules (7) determines a fuzzy set $\bar{B}^k \subset \mathbf{Y}$ given by the compositional rule of inference:

$$\bar{B}^k = A' \circ \left(A^k \to B^k\right) \tag{8}$$

where $A^k = A_1^k \times A_2^k \times \ldots \times A_n^k$. Fuzzy sets $\bar{B}^k$, according to the formula (8), are characterized by membership functions expressed by the sup-star composition:

$$\mu_{\bar{B}^k}(y) = \sup_{\mathbf{x}\in\mathbf{X}} \left\{ \mu_{A'}(\mathbf{x}) * \mu_{A_1^k \times \ldots \times A_n^k \to B^k}(\mathbf{x}, y) \right\} \tag{9}$$

where * can be any operator in the class of T-norms. It is easily seen that for a crisp input $\bar{\mathbf{x}} \in \mathbf{X}$, i.e. a singleton fuzzifier (6), formula (9) becomes:

$$\mu_{\bar{B}^k}(y) = \mu_{A_1^k \times \ldots \times A_n^k \to B^k}(\bar{\mathbf{x}}, y) = I\left(\mu_{A^k}(\bar{\mathbf{x}}), \mu_{B^k}(y)\right) \tag{10}$$

where $I(\cdot)$ is an "engineering implication' or fuzzy implication. The aggregation operator, applied in order to obtain the fuzzy set based on fuzzy sets $\bar{B}^k$, is the T-norm or S-norm operator, depending on the type of fuzzy implication. The

defuzzifier performs a mapping from a fuzzy set B' to a crisp point $\bar{y}$ in $\mathbf{Y} \subset \mathbf{R}$. The COA (centre of area) method is defined by following formula:

$$\bar{y} = \frac{\int\limits_{\mathbf{Y}} y\mu_{B'}(y)\,dy}{\int\limits_{\mathbf{Y}} \mu_{B'}(y)\,dy} \text{ or by } \bar{y} = \frac{\sum\limits_{r=1}^{N} \bar{y}^r \cdot \mu_{B'}(\bar{y}^r)}{\sum\limits_{r=1}^{N} \mu_{B'}(\bar{y}^r)} \tag{11}$$

in the discrete form, where $\bar{y}^r$ denotes centres of the membership functions $\mu_{B^r}(y)$, i.e. for $r = 1, \ldots, N$:

$$\mu_{B^r}(\bar{y}^r) = \max_{y \in \mathbf{Y}} \{\mu_{B^r}(y)\} \tag{12}$$

Following ideas in [23] we introduce the following flexibility parameters:

- soft strength of firing controlled by parameter α^τ,
- soft implication controlled by parameter α^I,
- soft aggregation of the rules controlled by parameter α^{agr},
- weights in antecedents of the rules $w_{i,k}^\tau \in [0,1]$, $i = 1, \ldots, n$, $k = 1, \ldots, N$,
- weights in aggregation of the rules $w_k^{agr} \in [0,1]$, $k = 1, \ldots, N$.

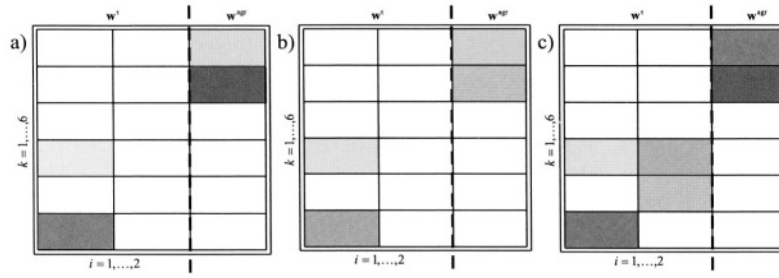


Fig. 1. Weights representation in the Nonlinear Dynamic Plant problem for system (13)-(16) and a) min-max triangular norms, b) algebraic triangular norms, c) Yager's triangular norms

The Yager's triangular norms in connection of antecedents, implication and aggregation are parameterised by parameters p^τ, p^I, p^{agr} respectively. The flexible neuro-fuzzy system based on Yager's triangular norms is given by:

$$\tau_k(\bar{\mathbf{x}}) = \begin{pmatrix} (1-\alpha^\tau)\,\text{avg}\left(\mu_{A_1^k}(\bar{x}_1), \ldots, \mu_{A_n^k}(\bar{x}_n)\right) \\ +\alpha^\tau \overset{\leftrightarrow}{T}^* \left\{\mu_{A_1^k}(\bar{x}_1), \ldots, \mu_{A_n^k}(\bar{x}_n); w_{1,k}^\tau, \ldots, w_{n,k}^\tau, p^\tau\right\} \end{pmatrix} \tag{13}$$

Table 1. Experimental results (min-max and algebraic triangular norms)

Experiment number	Name of flexibility parameter	Initial values	Final values after learning		RMSE (learning sequence)		RMSE (testing sequence)	
			min-max triangular norms	Algebraic triangular norms	min-max triangular norms	Algebraic triangular norms	min-max triangular norms	Algebraic triangular norms
i	λ	0.5	0.0000	0.0000	0.0414	0.0233	0.0218	0.0128
ii	λ	1	-	-	0.0459	0.0270	0.0276	0.0151
iii	λ	0.5	0.0000	0.0000	0.0374	0.0226	0.0198	0.0121
	α^{τ}	1	0.8178	0.9862				
	α^{I}	1	0.7218	0.9634				
	α^{agr}	1	0.9411	0.9780				
iv	λ	0.5	0.0000	0.0000	0.0280	0.0193	0.0172	0.0101
	α^{τ}	1	0.8589	0.9531				
	α^{I}	1	0.7270	0.9780				
	α^{agr}	1	0.9558	0.9205				
	$\mathbf{w}^{\tau}$	**1**	Fig.1-a	Fig.1-b				
	$\mathbf{w}^{agr}$	**1**	Fig.1-a	Fig.1-b				

$$I_{k,r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right)=\left(\begin{array}{c}\left(1-\alpha^{I}\right)\operatorname{avg}\left(\tilde{N}_{1-\lambda}\left(\tau_{k}\left(\bar{\mathbf{x}}\right)\right),\mu_{B^{k}}\left(\bar{y}^{r}\right)\right)\\ +\alpha^{I}\left(\begin{array}{c}\left(1-\lambda\right)\overleftrightarrow{T}\left\{\tau_{k}\left(\bar{\mathbf{x}}\right),\mu_{B^{k}}\left(\bar{y}^{r}\right);p^{I}\right\}\\ +\lambda\overleftrightarrow{S}\left\{1-\tau_{k}\left(\bar{\mathbf{x}}\right),\mu_{B^{k}}\left(\bar{y}^{r}\right);p^{I}\right\}\end{array}\right)\end{array}\right) \tag{14}$$

$$agr_{r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right)=\left(\begin{array}{c}\left(1-\alpha^{agr}\right)\operatorname{avg}\left(I_{1,r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right),\ldots,I_{N,r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right)\right)\\ +\alpha^{agr}\left(\begin{array}{c}\left(1-\lambda\right)\overleftrightarrow{S}^{*}\left\{\begin{array}{c}I_{1,r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right),\ldots,I_{N,r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right);\\ w_{1}^{agr},\ldots,w_{N}^{agr},p^{agr}\end{array}\right\}\\ +\lambda\overleftrightarrow{T}^{*}\left\{\begin{array}{c}I_{1,r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right),\ldots,I_{N,r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right);\\ w_{1}^{agr},\ldots,w_{N}^{agr},p^{agr}\end{array}\right\}\end{array}\right)\end{array}\right) \tag{15}$$

$$\bar{y}=\frac{\sum_{r=1}^{N}\bar{y}^{r}\cdot agr_{r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right)}{\sum_{r=1}^{N}agr_{r}\left(\bar{\mathbf{x}},\bar{y}^{r}\right)} \tag{16}$$

Compromise operator in formula (14) is defined as follows:

$$\tilde{N}_{\nu}\left(a\right)=\left(1-\nu\right)N\left(a\right)+\nu a \tag{17}$$

where $\nu\in[0,1]$, for details see [23].

Table 2. Experimental results (Yager's triangular norms)

Experiment number	Name of flexibility parameter	Initial values	Final values after learning	RMSE (learning sequence)	RMSE (testing sequence)
i	λ	0.5	0.0000	0.0263	0.0124
ii	λ	1	-	0.0419	0.0209
iii	λ	0.5	0.0000	0.0257	0.0119
	p^{τ}	10	6.0634		
	p^{l}	10	9.1289		
	p^{agr}	10	2.0598		
	α^{τ}	1	0.8548		
	α^{l}	1	0.9432		
	α^{agr}	1	0.9567		
iv	λ	0.5	0.0000	0.0236	0.0109
	p^{τ}	10	10.7117		
	p^{l}	10	7.8691		
	p^{agr}	10	3.9769		
	α^{τ}	1	0.9247		
	α^{l}	1	0.9659		
	α^{agr}	1	0.9921		
	$\mathbf{w}^{\tau}$	**1**	Fig. 2-c		
	$\mathbf{w}^{agr}$	**1**	Fig. 2-c		

3 Simulation Results

The flexible neuro-fuzzy system based on Yager's triangular norms, described by formulas (13)-(16), is simulated on Nonlinear Dynamic Plant problem [23]. We consider the second order nonlinear plant studied by Wang and Yen [23]

$$y(k) = g(y(k-1), y(k-2)) + u(k) \tag{18}$$

with

$$g(y(k-1), y(k-2)) = \frac{y(k-1)\, y(k-2)\, (y(k-1) - 0.5)}{1 + y^2(k-1) + y^2(k-2)} \tag{19}$$

The goal is to approximate the nonlinear component $g(y(k-1), y(k-2))$ of the plant with a fuzzy model. In [23], 400 simulated data were generated from the plant model (19). Starting from the equilibrium state (0,0), 200 samples of identification data were obtained with a random input signal $u(k)$ uniformly distributed in [–1.5,1.5], followed by 200 samples of evaluation data obtained using a sinusoidal input signal $u(k) = \sin(2\pi k/25)$. The experimental results

for the Nonlinear Dynamic Plant problem are shown in Tables 1 and 2 for the not parameterised (min-max and algebraic) and parameterised (Yager) triangular norms, respectively. For experiment (iv) the final values (after learning) of weights $w_{i,k}^{\tau} \in [0,1]$ and $w_k^{agr} \in [0,1]$, $i = 1, \ldots, 2$, $k = 1, \ldots, 6$, are depicted in Fig. 1.

4 Final Remarks

It is well known that Mamdani type systems are functionally equivalent to probabilistic neural networks [5] derived from non parametric estimators of regression functions [1], [2], [17], [18], [19], [20], [21], [24], [26], [27]. Therefore, we may expect that neuro-fuzzy systems exhibit desirable probabilistic properties.

References

1. Gałkowski, T., and Rutkowski, L.: Nonparametric fitting of multivariable functions, IEEE Transactions on Automatic Control, Vol. AC-31, (1986) 785-787.
2. Gałkowski, T., and Rutkowski, L.: Nonparametric recovery of multivariate functions with applications to system identification, Proceedings of the IEEE, Vol. 73, (1985) 942-943, New York.
3. Klement, E. P., Mesiar, R., Pap, E.: Triangular Norms, Kluwer Academic Publishers, Netherlands (2000).
4. Nowicki, R., and Rutkowski, L.: Rough-neuro-fuzzy system for classification, 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22 (2002).
5. Nowicki, R., and Rutkowski, L.: Soft techniques for bayesian classification, in: Rutkowski, L., Kacprzyk J. (Eds.), Neural Networks, and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg, New York, (2003) 537-544.
6. Nowicki, R., Scherer, R., and Rutkowski, L.: A hierarchical neuro-fuzzy systems based on s-implication, IJCNN-2003 Conference Proceedings, International Joint Conference on Neural Networks, Portland, Oregano, July 20-24, (2003) 321-325.
7. Nowicki, R., Scherer, R., and Rutkowski, L.: A method for learning of hierarchical fuzzy systems, Proceedings of the 2nd Euro-International Symposium on Computational Intelligence, Vol. 76, (2002) 124-129, Koszyce.
8. Nowicki, R., Scherer, R., and Rutkowski, L.: A neuro-fuzzy system based on the hierarchical prioritized structure, 10th Zittau Fuzzy Colloquium, (2002) 192-198, Germany, September 4-6.
9. Rutkowska, D.: Neuro-Fuzzy Architectures and Hybrid Learning, Springer-Verlag (2001).
10. Rutkowska, D., and Rutkowski, L., Fuzzy neural networks, Proceedings of the Second International Conference on Parallel Processing, and Applied Mathematics, Zakopane, 2-5 September, (1997) 507-519.
11. Rutkowska, D., and Rutkowski, L.: Neural-Fuzzy-Genetic Parallel Computing System as a Tool for Various Applications, Proceedings of the Third International Conference on Parallel Processing and Applied Mathematics (PPAM'99), Kazimierz Dolny, (1999) 489-498.

12. Rutkowska, D., Nowicki, R., and Rutkowski L: Neuro-fuzzy architectures with various implication operators, in: Sincák P., Vascak J., Kvasnicka V., and Mesiar R. (Eds.), The State of the Art in Computational Intelligence, Physica-Verlag, A Springer-Verlag Company, Heidelberg, New York, (2000) 214-219, Springer.
13. Rutkowska, D., Nowicki, R., and Rutkowski, L.: Singleton and Non-Singleton Fuzzy Systems with Nonparametric Defuzzification, in: Computational Intelligence, and Applications, P. S. Szczepaniak (ed.), Springer-Verlag, (1999) 292-301.
14. Rutkowska, D.s., Piliński, M., and Rutkowski, L.: Fuzzy neural controllers, Proceedings of Wismarer Automatisierungssymposium, Hochschule Wismar, 17-18 September, (1996) H2-lH2-H8.
15. Rutkowska, D., Rutkowski, L., and Nowicki, R.: Fuzzy neural networks with nonparametric defuzzification, Proceedings of the 9-th International Conference, System-Modelling-Control, April 27 - May 1, Zakopane (1998).
16. Rutkowska, D., Rutkowski, L., and Nowicki, R.: On processing of noisy data by fuzzy inference neural networks, Proceedings of the IASTED International Conference Signal, and Image Processing (SIP'99), Nassau, Bahamas, (1999) 314-318.
17. Rutkowski, L.: Adaptive probabilistic neural-networks for pattern classification in time-varying environment, IEEE Trans. Neural Networks, vol. 15, May (2004).
18. Rutkowski, L., An application of multiple Fourier series to identification of multivariable nonstationary systems, International Journal of Systems Science, Vol. 20, No.10, (1989) 1993-2002.
19. Rutkowski, L.: Identification of MISO nonlinear regressions in the presence of a wide class of disturbances, IEEE Transactions on Information Theory, Vol. IT-37, (1991) 214-216.
20. Rutkowski, L.: Multiple Fourier series procedures for extraction of nonlinear regressions from noisy data, IEEE Transactions on Signal Processing, Vol. 41, No.10, (1993) 3062-3065.
21. Rutkowski, L.: New Soft Computing Techniques for System Modelling, Pattern Classification and Image Processing. Springer-Verlag (2004).
22. Rutkowski, L., and Cpałka, K.: A General Approach to Neuro-Fuzzy Systems, The 10th IEEE International Conference on Fuzzy Systems, Melbourne (2001).
23. Rutkowski, L., Cpałka, K.: Flexible neuro-fuzzy systems, IEEE Trans. Neural Networks, vol. 14, pp. 554-574, May (2003).
24. Rutkowski, L., and Gałkowski, T.: On pattern classification and system identification by probabilistic neural networks, Applied Mathematics, and Computer Science, Vol. 4, No.3, (1994) 413-422.
25. Rutkowski, L., and Piliński, M.: Neural networks for fuzzy control, Proceedings of the 8-th International Conference, System-Modelling-Control, (1995) 96-98.
26. Rutkowski, L., and Rafajłowicz, E.: On global rate of convergence of some nonparametric identification procedures, IEEE Transaction on Automatic Control, Vol. AC-34, No.10, (1989) 1089-1091.
27. Rutkowski, L., Rutkowska, D., and Gałkowski, T., Probabilistic neural networks and fuzzy logic systems, Proceedings of the 8-th International Conference, System-Modelling-Control (1995) 99-102.
28. Rutkowska, D., Nowicki, R., and Rutkowski, L.: Neuro-Fuzzy System with Inference Process Based on Zadeh Implication, Proceedings of the Third International Conference on Parallel Processing and Applied Mathematics (PPAM'99), Kazimierz Dolny, (1995) 597-602.
29. Rutkowski, L., and Starczewski, J.: From type-1 to type-2 fuzzy interference systems - part 1, Proceedings of the Fifth Conference Neural Networks, and Soft Computing, June 6-10, Zakopane (2000) 46-51.

30. Rutkowski, L., and Starczewski, J.: From type-1 to type-2 fuzzy interference systems - part 2, Proceedings of the Fifth Conference Neural Networks, and Soft Computing, June 6-10, Zakopane (2000) 52-65.
31. Rutkowski, L., and Zapart, K.: Fuzzy neural networks and their applications, Proceedings of the 8-th International Conference, System-Modelling-Control, (1995) 41-46.
32. Scherer, R., and Rutkowski, L.: A fuzzy relational system with linguistic antecedent certainty factors, in: Rutkowski, L., Kacprzyk J. (Eds.), Neural Networks, and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg, New York, (2003) 563-569.
33. Scherer, R., and Rutkowski, L.: A neuro-fuzzy relational system, Fourth International Conference on Parallel Processing, and Applied Mathematics, (2001) 131 - 135, Czestochowa.
34. Scherer, R., and Rutkowski, L.: A survey of hierarchical fuzzy systems, Proceedings of the Fifth Conference Neural Networks, and Soft Computing, June 6-10, Zakopane (2000) 374-379.
35. Scherer, R., and Rutkowski, L.: Neuro-fuzzy relational systems, 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22 (2002).
36. Scherer, R., and Rutkowski, L.: Relational equations initializing neuro-fuzzy system, 10th Zittau Fuzzy Colloquium, Germany, September 4-6 (2002) 212-217.
37. Starczewski, J., and Rutkowski, L.: Connectionist Structures of Type 2 Fuzzy Inference Systems, Lecture Notes in Computer Science, Vol. 2328, (2001) 634-642.
38. Starczewski, J., and Rutkowski, L.: Interval type 2 neuro-fuzzy systems based on interval consequents, in: Rutkowski, L., Kacprzyk J. (Eds.), Neural Networks, and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg, New York, (2003) 570-577.
39. Starczewski, J., and Rutkowski, L.: Neuro-fuzzy inference systems of type 2, 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22 (2002).
40. Yager, R. R., Filev, D. P.: Essentials of Fuzzy Modeling and Control, John Wiley and Sons (1994).

Recursive Probabilistic Neural Networks

Marcin Korytkowski[1], Marcin Gabryel[1], and Adam Gaweda[2]

[1] Department of Computer Engineering, Czestochowa University of Technology,
Al. Armii Krajowej 36, 42-200 Czestochowa, Poland,
{marcink,marcing}@kik.pcz.czest.pl
[2] University of Louisville, Louisville, KY 40292 USA,
agaweda@kdp.louisville.edu

Abstract. In this paper we investigate probabilistic neural networks in a time-varying environment. We apply the idea of dynamic stochastic approximation methods for construction of probabilistic neural networks working in a time-varying environment. A simulation example is given showing tracking properties of probabilistic neural networks.

1 Introduction

Probabilistic neural networks have been proposed in [1]. They are a network realization of non-parametric procedures for density estimation [2,7], regression estimation [3,5] and classification [3,7]. They work in stationary [6,9] and non-stationary [4,8] environments. Probabilistic neural networks have nice convergence properties. They approximate non-linear functions and Bayes decision functions. In this paper we extend the results presented in [11] and investigate probabilistic neural networks in a time varying environment. We apply the idea suggested in [11] for construction of probabilistic neural networks based on dynamic stochastic approximation methods.

2 Generalized Regression Neural Networks in a Stationary Case

Let *(X, Y)* be a pair of random variables. *X* takes values in a Borel set *A*, $A \subset R^p$, whereas *Y* takes values in *R*. Let *f* be the marginal Lebesgue density of *X*. Based on a sample $(X_1, Y_1), ..., (X_n, Y_n)$ of independent observations of *(X, Y)* we wish to estimate the regression function

$$\phi(x) = E[Y|X = x] \tag{1}$$

To estimate function (1) we propose the following formula

$$\widehat{\phi}_n(x) = \frac{\widehat{R}_n(x)}{\widehat{f}_n(x)} \tag{2}$$

where

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 626–631, 2004.
© Springer-Verlag Berlin Heidelberg 2004

$$\widehat{R}_n(x) = \frac{1}{n}\sum_{i=1}^{n} Y_i K_i(x, X_i) \tag{3}$$

and

$$\widehat{f}_n(x) = \frac{1}{n}\sum_{i=1}^{n} K_i(x, X_i) \tag{4}$$

where K_n is a sequence of kernel functions. Observe that procedures (3) and (4) can be expressed in the recursive form

$$\widehat{R}_{n+1}(x) = \widehat{R}_n(x) + \frac{1}{n+1}\left[Y_{n+1}K_{n+1}(x, X_{n+1}) - \widehat{R}_n(x)\right] \tag{5}$$

and

$$\widehat{f}_{n+1}(x) = \widehat{f}_n(x) + \frac{1}{n+1}\left[K_{n+1}(x, X_{n+1}) - \widehat{f}_n(x)\right] \tag{6}$$

$\widehat{R}_0(x) = 0$, $\widehat{f}_0(x) = 0$. In this paper we assume that sequence K_n is based on the Parzen kernel given by:

$$K_n(x, u) = h_n^{-p} K\left(\frac{x-u}{h_n}\right) \tag{7}$$

where h_n is a certain sequence of numbers and K is an appropriately selected function. We assume that

$$K(x) = \prod_{i=1}^{p} H\left(x^{(i)}\right) \tag{8}$$

Consequently, sequence K_n is expressed by formula

$$K_n(x, u) = h_n^{-p} \prod_{i=1}^{p} H\left(\frac{x^{(i)} - u^{(i)}}{h_n}\right) \tag{9}$$

We will illustrate the performance of algorithm (2) applied to estimation of stationary regression in the model:

$$y_n = 5\sin(x_n) \tag{10}$$

where x_n is realization of $N(0,1)$ random variables. We assume that

$$h_n = kn^{-H} \tag{11}$$

where $k = 5$, $H = 0.5$ and H is a Gaussian function. The results are depicted in Fig. 1, which displays comparison of a true regression and estimated by (2) for $n = 1000$.

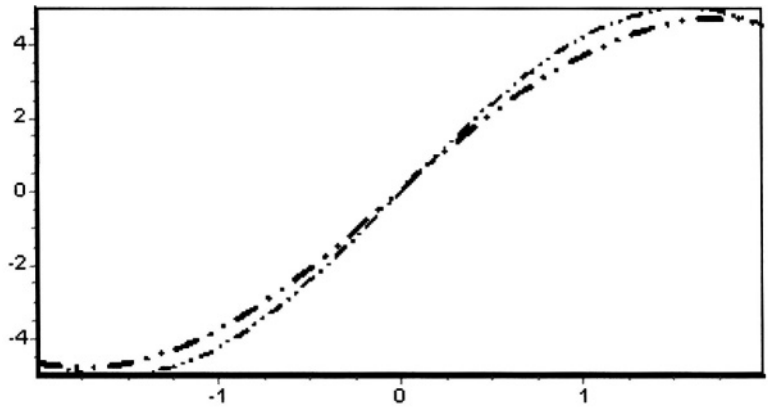


Fig. 1. Estimation of a stationary regression function.

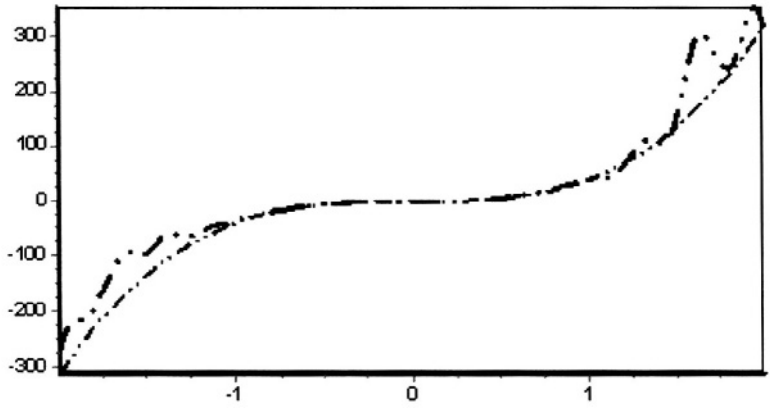


Fig. 2. Estimation of a non-stationary regression function.

3 Generalized Regression Neural Networks in a Non-stationary Case

Let (X_n, Y_n) be a pair of random variables, X_n takes values in a Borel set A, $A \subset R^p$, whereas Y_n takes values in R^1. Let $f(x)$ be the marginal Lebesgue density of X and let $f_n(x, y)$ be the joint non-stationary probability density function. We wish to estimate the regression function

$$\phi_n(x) = E[Y_n | X_n = x], \ n = 1, 2... \tag{12}$$

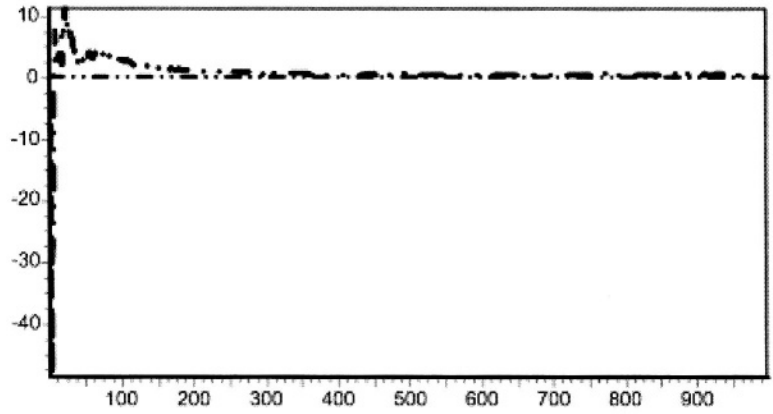


Fig. 3. The non-stationary regression with changing n in point $x = 0.2$.

The generalized regression neural network for tracking function $\phi_n(x)$ is given by:

$$\widehat{\phi}_n(x) = \frac{\widehat{R}_n(x)}{\widehat{f}_n(x)} \tag{13}$$

$$\widehat{R}_{n+1}(x) = \widehat{R}_n(x) + a_n\left[Y_{n+1}K_{n+1}(x, X_{n+1}) - \widehat{R}_n(x)\right] \tag{14}$$

and

$$\widehat{f}_{n+1}(x) = \widehat{f}_n(x) + \frac{1}{n+1}\left[K_{n+1}(x, X_{n+1}) - \widehat{f}_n(x)\right] \tag{15}$$

Procedure (14) can be modified in the spirit of dynamic stochastic approximation algorithms. The modification takes the form:

$$\widehat{R}_{n+1}(x) = M_n(\widehat{R}_n(x)) + a_n\left[Y_{n+1}K_{n+1}(x, X_{n+1}) - M_n(\widehat{R}_n(x))\right] \tag{16}$$

Using arguments similar to those in [11] it can be shown that conditions

$$a_n \to 0$$

$$a_n^{-1}\left|R_{n+1}(x) - M_n(R_n(x))\right| \xrightarrow{n} 0. \tag{17}$$

imply convergence of algorithm (13).

We will illustrate the performance of algorithm (13) applied to estimation of non-stationary regression in the model

$$y_n = 10x^3 n^{0.2}$$

where x_n is realization of $N(0,1)$ random variables. We assume that

$$h_n = 5n^{-0.6}$$

in procedure (14) and

$$h_n = 5n^{-0.5}$$

in procedure (15). Moreover, H is a Gaussian function,

$$a_n = n^{-0.7}$$

and

$$M_n\left(R_n\left(x\right)\right) = \left(1 + \frac{1}{n+1}\right) R_n\left(x\right) \tag{18}$$

The results for $n = 1000$ are depicted in Fig. 2. Figure 3 shows tracking the non-stationary regression with changing n in point $x = 0.2$.

4 Final Remarks

In the paper we investigated probabilistic neural networks in the stationary and non-stationary environments. It has been shown that probabilistic neural networks converge to a true regression. In the future research it would be interesting to describe non-stationary changes linguistically and apply various neuro-fuzzy structures (see [10], [12-16]) in a time-varying environment.

References

1. Specht D. F. (1990): *Probabilistic neural networks,* Neural Networks, Vol. 3, pp. 109-118.
2. Rutkowski L. (1980): *Sequential estimates of probability densities by orthogonal series and their application in pattern classification,* IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, No 12, pp. 918-920.
3. Rutkowski L. (1981): *Sequential estimates of a regression function by orthogonal series with applications in discrimination,* in: Lectures Notes in Statistics, Vol. 8, pp. 236-244, New York, .
4. Rutkowski L. (1982): *On Bayes risk consistent pattern recognition procedures in a quasi-stationary environment,* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, No 1, pp. 84-87.
5. Gałkowski T. and Rutkowski L. (1986): *Nonparametric fitting of multivariable functions,* IEEE Transactions on Automatic Control, Vol. AC-31, pp. 785-787.
6. Rutkowski L. (1988): *Nonparametric procedures for identification and control of linear dynamic systems,* Proceedings of 1988 American Control Conference, June 15-17, pp. 1325-1326.
7. Rutkowski L. (1988): *Sequential pattern recognition procedures derived from multiple Fourier series,* Pattern Recognition Letters, Vol. 8, pp. 213-216.

8. Rutkowski L. (1989): *An application of multiple Fourier series to identification of multivariable nonstationary systems,* International Journal of Systems Science, Vol. 20, No. 10, pp. 1993-2002.
9. Rutkowski L. and Rutkowska D. (1993): *On nonparametric identification of linear circuits by Walsh orthogonal series,* Proceedings of the 7-th International Conference, System-Modelling-Control, Zakopane, Maj 17-21, pp. 146-148.
10. Rutkowski L. and Cpałka K. (2003): *Flexible neuro-fuzzy systems,* IEEE Transactions on Neural Networks, vol. 14, pp. 554-574.
11. Rutkowski L, (2004): *New Soft Computing Techniques for System Modelling, Pattern Classification and Image Processing,* Springer-Verlag.
12. Scherer R. and Rutkowski L. (2000): *A survey of hierarchical fuzzy systems,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 374-379, June 6-10, Zakopane.
13. Starczewski J. and Rutkowski L. (2001): *Connectionist Structures of Type 2 Fuzzy Inference Systems,* Lecture Notes in Computer Science, Vol. 2328, pp. 634-642.
14. Rutkowski L. and Cpałka K. (2002): *A neuro-fuzzy controller with a compromise fuzzy reasoning,* Control and Cybernetics, Vol. 31, No. 2, pp. 297-308.
15. Scherer R. and Rutkowski L. (2002): *Relational equations initializing neuro-fuzzy system,* 10th Zittau Fuzzy Colloquium, pp. 212-217, Germany, September 4-6.
16. Nowicki R., Scherer R. and Rutkowski L. (2002): *A neuro-fuzzy system based on the hierarchical prioritized structure,* 10th Zittau Fuzzy Colloquium, pp. 192-198, Germany, September 4-6.

Neuro-Fuzzy versus Non-parametric Approach to System Modeling and Classification

Robert Nowicki

Department of Computer Engineering, Czestochowa University of Technology,
Al. Armii Krajowej 36, 42-200 Czestochowa, POLAND,
rnowicki@kik.pcz.czest.pl,
http://kik.pcz.czest.pl

Abstract. This paper presents the comparative study concerning selected neuro-fuzzy systems and non-parametric methods. Moreover, a new idea of rough-neuro-fuzzy systems is suggested to solve the problem of missing features. The main applications of methods under study are system modeling and classification. The non-parametric methods are based on density and regression estimates. They converge to the optimal solution when the sample size grows large. The neuro-fuzzy structures do not possess convergence properties however they are applied successfully in modeling and classification problems. The methods are illustrated on several simulation examples.

1 Introduction

In the literature various methods have been developed for system modeling and pattern recognition. They include non-parametric procedures based on density [6, 18] and regression [7, 10, 20, 21] estimates. Non-parametric algorithms converge to the optimal solution when the sample size grows large. Applications include system identification [11, 12, 14, 16, 17, 19] and pattern classification [8, 18, 22]. They have been investigated in a stationary and non-stationary environment [9, 13, 15, 43, 44]. The neuro-fuzzy structures do not possess convergence properties however they are applied successfully in modeling and classification problems. Among various neuro-fuzzy structures we mention: Mamdani-type systems [23, 45], logical-type systems [24, 25, 29, 31, 33, 34, 36, 39, 42, 45], hierarchical systems [1, 2, 3, 5, 28], type-2 systems [32, 26, 27, 38, 41], rough systems [4] and relational systems [30, 35, 37, 40]. So far in the literature there is a lack of analysis and comparision of non-parametric techniques various neuro-fuzzy methods. In this paper we present the comparative study concerning selected neuro-fuzzy systems and non-parametric techniques. Moreover, a new idea of rough-neuro-fuzzy systems will be suggested to solve the problem of missing features.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 632–640, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Neuro-Fuzzy Approach

We consider a multi-input and a muli-output fuzzy system with a fuzzifier which maps $\overline{\mathbf{x}} = [\overline{x}_1, \ldots, \overline{x}_n] \in \mathbf{X}$ into a fuzzy set A' desribed by the membership function

$$\mu_{A'}(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{x} = \overline{\mathbf{x}} \\ 0 \text{ if } \mathbf{x} \neq \overline{\mathbf{x}} \end{cases} \tag{1}$$

The fuzzy rules are in the form

$$R^{(k)} : \text{IF } \mathbf{x} \text{ is } A^k \text{ THEN } y_1 \text{ is } B_1^k \text{ AND } \ldots \text{ AND } y_m \text{ is } B_m^k \tag{2}$$

where $\mathbf{x} = [x_1, \ldots, x_n] \in \mathbf{X}$, $A^k = A_1^k \times A_2^k \times \ldots A_n^k$

The interpretation of rules (2) is different in the case of modeling and in the case of classification. In the case of modeling y_j, $j = 1, \ldots, m$, is the linguistic variable related to the j-th modeled quantity. In the second case y_j, $j = 1, \ldots, m$, is a degree of the membership of input $\overline{\mathbf{x}}$ (or object represented by $\overline{\mathbf{x}}$) to the j-th class. By a compositional rule of inference we have

$$\overline{B}_j^k = A' \circ \left(A^k \rightarrow B_j^k\right)$$

and

$$\mu_{\overline{B}_j^k}(y_j) = \sup_{\mathbf{x} \in \mathbf{X}} \left\{ \mu_{A'}(\mathbf{x}) \overset{T}{*} \mu_{A^k \rightarrow B_j^k}(\mathbf{x}, y_j) \right\}$$

The aggregation is performed by union of fuzzy sets $\overline{B}_j^k$.

$$B_j' = \bigcup_{k=1}^{N} \overline{B}_j^k$$

The defuzzification gives the crisp output

$$\overline{y}_j = \frac{\sum_{k=1}^{N} \overline{y}_j^k \mu_{B_j'}\left(\overline{y}_j^k\right)}{\sum_{k=1}^{N} \mu_{B_j'}\left(\overline{y}_j^k\right)} \tag{3}$$

3 Non-parametric Approach

Let $\mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^N$ be a sequence of i.i.d. random variables having a probability density function f. The Parzen-Rosenblatt estimator of probability density f is given by

$$\widehat{f}_N(x) = \frac{1}{N\sigma_N^n} \sum_{k=1}^{N} K\left(\frac{\mathbf{x} - \mathbf{X}^k}{\sigma_N}\right) \tag{4}$$

where K is an appriopriately chosen function fulfiling the conditions

$$\sup_y |K(y)| < \infty \tag{5}$$

$$\int_{\Re^n} |K(y)|\, dy < \infty \tag{6}$$

$$\lim_{\|y\| \to \infty} \|y\|^n |K(y)| = 0 \tag{7}$$

$$\int_{\Re^n} K(y)\, dy = 1 \tag{8}$$

and the sequence σ_N is a function of N and satisfies the conditions

$$\lim_{N \to \infty} \sigma_N = 0 \text{ and } \lim_{N \to \infty} N\sigma_N = \infty \tag{9}$$

3.1 Regression Estimation

The Parzen-Rosenblatt approach applied to the estimation of a regression function

$$R_N(\mathbf{x}) = E[Y|\mathbf{X} = \mathbf{x}] \tag{10}$$

takes the form

$$\widehat{R}_N(\mathbf{x}) = \frac{\sum_{k=1}^{N} Y^k K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)}{\sum_{k=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)} \tag{11}$$

3.2 Nonparametric Classification

Let $\{(\mathbf{X},Y), (\mathbf{X}^1,Y^1), (\mathbf{X}^2,Y^2), \ldots, (\mathbf{X}^N,Y^N)\}$, be a sequence of i.i.d. pairs of random variables, where Y takes values in set $S = \{1,\ldots, M\}$, $\mathbf{X}$ takes the values in $\Re^n$. The problem is to estimate Y from $\mathbf{X}$ and $V^N = (\mathbf{X}^1,Y^1), (\mathbf{X}^2,Y^2), \ldots, (\mathbf{X}^N,Y^N)$ where V^N is the learning sequence. Supose that p_m is the prior class probabilities, f_m is the class conditional densities and $m = 1, \ldots, M$. We define the Bayes discriminate function

$$g_m(\mathbf{x}) = p_m f_m(\mathbf{x}) \tag{12}$$

Let

$$T_m^k = \begin{cases} 1 \text{ if } Y^k = m \\ 0 \text{ if } Y^k \neq m \end{cases} \tag{13}$$

for $k = 1, 2, \ldots, N$ and $m = 1, 2, \ldots M$. Then Bayes discriminate function (12) can be rewritten as follows

$$g_m(\mathbf{x}) = f(\mathbf{x})\, E\left[T_m^k | \mathbf{X}^k = \mathbf{x}\right] \tag{14}$$

where $f(\mathbf{x}) = \sum_{m=1}^{N} p_m f_m(\mathbf{x})$. We consider a procedure classifying avery $\mathbf{x}$ to a class m, $m \in S$, which maximizes $\widehat{g}_{Nm}(\mathbf{x})$, where $\widehat{g}_{Nm}(\mathbf{x})$ is the following estimate of the Bayes discriminate function

$$\widehat{g}_{Nm}(\mathbf{x}) = \frac{\sum_{k=1}^{N} T_m^k K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)}{\sum_{k=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)} \tag{15}$$

4 Missing Features Problem

The rough set theory is a very useful tool to analyze the significance of individual quantities in classification problems. It allows to specify a quality of classification in case of limited information about classified objects. Moreover the theory specifies the range of classification inaccuracy when some information is missing. In literature we can find also the extension of rough set theory to fuzzy sets. The concepts of rough fuzzy sets *is* useful to built the neuro-fuzzy classifier, which can work even if some information is missing. Of course, the quality of classification will be worse but we are able to specify the range of classification inaccuracy.

Table 1. Results of Glass identification

	1	1	1	1	1	1	1	1	1	1
N-P	0.987	0.996	0.989	0.994	0.977	0.989	0.974	0.970	0.955	0.986
NFS	0.832	0.995	0.997	0.995	0.995	0.992	0.999	0.988	0.928	0.994
	0	0	0	0	0	0	0	0	0	0
N-P	0.001	0.002	0.000	0.001	0.689	0.429	0.292	0.000	0.000	0.000
NFS	0.000	0.000	0.000	0.000	0.000	0.573	0.018	0.000	0.000	0.000

Table 2. Result of the 1st wine class recognition

	1	1	1	1	1	1	1	0	0	0
N-P	1.000	1.000	1.000	1.000	1.000	0.000	1.000	0.000	0.000	0.000
NFS	0.806	0.960	0.976	0.812	0.851	0.990	0.978	0.116	0.049	0.169
	0	0	0	0	0	0	0	0	0	0
N-P	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.120
NFS	0.088	0.095	0.090	0.054	0.109	0.087	0.045	0.090	0.037	0.076

Table 3. Result of the 2nd wine class recognition

	0	0	0	0	0	0	0	1	1	1
N-P	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000
NFS	0.193	0.036	0.023	0.186	0.149	0.009	0.021	0.816	0.938	0.611
	1	1	1	1	1	0	0	0	0	0
N-P	1.000	0.000	1.000	0.473	1.000	1.000	0.000	0.000	1.000	0.000
NFS	0.901	0.878	0.901	0.917	0.883	0.035	0.153	0.145	0.370	0.073

Table 4. Result of the 3rd wine class recognition

	0	0	0	0	0	0	0	0	0	0
N-P	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
NFS	0.001	0.004	0.001	0.001	0.000	0.000	0.000	0.068	0.013	0.220
	0	0	0	0	0	1	1	1	1	1
N-P	0.000	1.000	0.000	0.527	0.000	0.000	1.000	1.000	0.000	0.859
NFS	0.011	0.027	0.009	0.029	0.008	0.878	0.801	0.765	0.593	0.852

Table 5. Result of Rice taste problem

	0.520	0.660	0.500	0.860	0.420	0.820	0.870	0.370	0.680	0.670
N-P	0.500	0.640	0.441	0.820	0.470	0.988	0.860	0.460	0.590	0.660
NFS	0.536	0.618	0.475	0.816	0.458	0.842	0.846	0.407	0.628	0.660
	0.770	0.300	0.660	0.650	0.650	0.990	0.850	0.180	0.790	0.650
N-P	0.818	0.390	0.590	0.690	0.710	0.925	0.820	0.220	0.820	0.660
NFS	0.747	0.383	0.652	0.694	0.706	0.884	0.813	0.226	0.776	0.664
	0.850	0.600	0.700	0.500	0.480	0.830	0.410	0.530	0.720	0.400
N-P	0.850	0.600	0.690	0.600	0.500	0.819	0.470	0.680	0.770	0.460
NFS	0.818	0.535	0.665	0.553	0.506	0.844	0.459	0.520	0.702	0.403

When information about values of some elements of vector $\mathbf{x}$ is missing then we will divide the elements of $\mathbf{x}$ into two sets D and G. D is the set of elements with known values whereas G is the set of elements with unknown values. So, the classification process is based on the properties from set D only. The rough fuzzy set corresponding to set D is defined as a pair $\widetilde{D}A = \left\{\underline{\widetilde{D}}A, \overline{\widetilde{D}}A\right\}$, where $\underline{\widetilde{D}}A$ is a $\widetilde{D}$-lower approximation and $\overline{\widetilde{D}}A$ is the $\widetilde{D}$-upper approximation of set A. The membership functions of $\underline{\widetilde{D}}A$ and $\overline{\widetilde{D}}A$ are defined as follows

$$\mu_{\underline{\widetilde{D}}A}(\widehat{\mathbf{x}}) = \inf_{\mathbf{x}\in[\widehat{\mathbf{x}}]_{\widetilde{D}}} \{\mu_A(\mathbf{x})\} \tag{16}$$

and

$$\mu_{\overline{\widetilde{D}}A}(\widehat{\mathbf{x}}) = \sup_{\mathbf{x}\in[\widehat{\mathbf{x}}]_{\widetilde{D}}} \{\mu_A(\mathbf{x})\} \tag{17}$$

where $\widetilde{D}$ is the equivalence relation and $[\widehat{\mathbf{x}}]_{\widetilde{D}}$ is the equivalence class of relation with element $\widehat{\mathbf{x}}$. Note that if set A is a fuzzy set, then its $\widetilde{D}$-lower and $\widetilde{D}$-upper

approximations are also fuzzy sets. We propose to use two outputs of NFS for each class. The value of first output is the minimal degree of input $\mathbf{x}$ membership to the j-th class, whereas the value of the second output is the maximal degree of input $\mathbf{x}$ membership to the j-th class. For neuro-fuzzy systems described in section 2 the values of $\underline{\overline{y}_j}$ and $\overline{\overline{y}_j}$ are expressed by

$$\underline{\overline{y}_j} = \frac{\sum\limits_{\substack{r=1 \\ r:\overline{\overline{y}}_j^r=1}}^{N} \mu_{\underline{\widetilde{D}}A}(\overline{\mathbf{x}})}{\sum\limits_{\substack{r=1 \\ r:\overline{\overline{y}}_j^r=1}}^{N} \mu_{\underline{\widetilde{D}}A}(\overline{\mathbf{x}}) + \sum\limits_{\substack{r=1 \\ r:\overline{\overline{y}}_j^r=0}}^{N} \mu_{\overline{\widetilde{D}}A}(\overline{\mathbf{x}})} \tag{18}$$

$$\overline{\overline{y}_j} = \frac{\sum\limits_{\substack{r=1 \\ r:\overline{\overline{y}}_j^r=1}}^{N} \mu_{\overline{\widetilde{D}}A}(\overline{\mathbf{x}})}{\sum\limits_{\substack{r=1 \\ r:\overline{\overline{y}}_j^r=1}}^{N} \mu_{\overline{\widetilde{D}}A}(\overline{\mathbf{x}}) + \sum\limits_{\substack{r=1 \\ r:\overline{\overline{y}}_j^r=0}}^{N} \mu_{\underline{\widetilde{D}}A}(\overline{\mathbf{x}})} \tag{19}$$

and in case of using the non-parametric approach

$$\underline{\overline{y}_j} = \inf_{\mathbf{x}\in[\overline{\mathbf{x}}]_{\widetilde{D}}} \widehat{g}_{Nm}(\mathbf{x}) = \inf_{\mathbf{x}\in[\overline{\mathbf{x}}]_{\widetilde{D}}} \frac{\sum\limits_{k=1}^{N} T_m^k K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)}{\sum\limits_{k=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)} \tag{20}$$

$$\overline{\overline{y}_j} = \sup_{\mathbf{x}\in[\overline{\mathbf{x}}]_{\widetilde{D}}} \widehat{g}_{Nm}(\mathbf{x}) = \sup_{\mathbf{x}\in[\overline{\mathbf{x}}]_{\widetilde{D}}} \frac{\sum\limits_{k=1}^{N} T_m^k K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)}{\sum\limits_{k=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{X}^k}{\sigma_N}\right)} \tag{21}$$

5 Experimental Results

The result of three experiments is presented. Glass identification: The task was to classify the samples of glass. There are two classes: window glass and nonwindow glass. Each sample is described by nine attributes (RI: refractive index, Na2O, MgO, Al2O3, SiO2, K2O, CaO, BaO, Fe2O3). All attributes are continuous. Wine recognition: The task was to classify the samples of wine. They come from three different vineyards in the same region of Italy. The input to the inference process contains 13 parameters of wine samples, i.e. alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines and proline. Rise taste: The task was to specify the taste (0-1) of rice sample Each sample is described by five attributes (flavour, appearance, taste, stickiness, toughness).

6 Conclusions

In the paper two approaches, neuro-fuzzy versus non-parametric for system modeling and classification have been investigated and compared. The results are summarized as follows:
(i). The performance of both techniques is very similar.
(ii). The methods are described by similar formulas (3) and (11).
(iii). A design of neuro-fuzzy systems is time-consuming comparing with design of non-parametric algorithms which do not require learning procedure.
(iv). A parameter N in case of the non-parametric approach is a sample size and should be sufficiently large. In case of neuro-fuzzy systems parameter N is a number of rules and could be even reduced in the process of system design.

Finally, we note that both approaches give similar results, however in case of missing features the neuro-fuzzy approach outperforms non-parametric techniques.

References

1. Nowicki R., Scherer R. (2001): *A hierarchical fuzzy system with fuzzy intermediate variables,* Proceedings of The 9th Zittau Fuzzy Colloquium 2001, pp. 88-93.
2. Nowicki R., Scherer R., Rutkowski L. (2002): *A hierarchical fuzzy system with a new way of learning,* Proceedings of the 2nd Euro-International Symposium on Computational Intelligence, Koszyce.
3. Nowicki R., Scherer R., Rutkowski L. (2002): *A neuro-fuzzy system based on the hierarchical prioritized structure,* 10th Zittau Fuzzy Colloquium, 2002, Zittau, Germany, pp. 192-198.
4. Nowicki R., Rutkowski L. (2002): *Rough-neuro-fuzzy system for classification,* Proc. of Fuzzy Systems and Knowledge Discovery, Singapure, pp. 149.
5. Nowicki R., Scherer R., Rutkowski L. (2003): *A neuro-fuzzy system based on the hierarchical prioritized structure,* International Joint Conference on Neural Networks 2003 Conference Proceedings ,Portland Oregon, USA.
6. Rutkowski L. (1980): *Sequential estimates of probability densities by orthogonal series and their application in pattern classification,* IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, No 12, pp. 918-920.
7. Rutkowski L. (1981): *Sequential estimates of a regression function by orthogonal series with applications in discrimination,* in: Lectures Notes in Statistics, Vol. 8, pp. 236-244, New York, .
8. Greblicki W. and Rutkowski L.(1981): *Density-free Bayes risk consistency of nonparametric pattern recognition procedures,* Proceedings of the IEEE, Vol. 69, No 4, pp. 482-483.
9. Rutkowski L. (1982): *On Bayes risk consistent pattern recognition procedures in a quasi-stationary environment,* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, No 1, pp. 84-87.
10. Rutkowski L (1982): *On system identification by nonparametric function fitting,* IEEE Transactions on Automatic Control, Vol. AC-27, pp. 225-227.
11. Rutkowski L. (1982): *On-line identification of time-varying systems by nonparametric techniques,* IEEE Transactions on Automatic Control, Vol. AC-27, pp. 228-230.

12. Rutkowski L. (1982): *Nonparametric identification of weighting function by orthogonal series method,* in: Proceedings of 1982 American Control Conference, Arlington, Virginia, June 14-16, pp. 170-171.
13. Rutkowski L. (1984): *On nonparametric identification with prediction of time-varying systems,* IEEE Transactions on Automatic Control, Vol. AC-29, pp. 58-60.
14. Gałkowski T., and Rutkowski L. (1985): *Nonparametric recovery of multivariate functions with applications to system identification,* Proceedings of the IEEE, Vol. 73, pp. 942-943, New York.
15. Rutkowski L. (1985): *Nonparametric identification of quasi-stationary systems,* Systems and Control Letters, Vol. 6, pp. 33-35, Amsterdam.
16. Gałkowski T. and Rutkowski L. (1986): *Nonparametric fitting of multivariable functions,* IEEE Transactions on Automatic Control, Vol. AC-31, pp. 785-787.
17. Rutkowski L. (1988): *Nonparametric procedures for identification and control of linear dynamic systems,* Proceedings of 1988 American Control Conference, June 15-17, pp. 1325-1326.
18. Rutkowski L. (1988): *Sequential pattern recognition procedures derived from multiple Fourier series,* Pattern Recognition Letters, Vol. 8, pp. 213-216.
19. Rutkowski L. and Rafajłowicz E. (1989): *On global rate of convergence of some nonparametric identification procedures,* IEEE Transaction on Automatic Control, Vol. AC-34, No.10, pp. 1089-1091.
20. Rutkowski L. (1991): *Identification of MISO nonlinear regressions in the presence of a wide class of disturbances,* IEEE Transactions on Information Theory, Vol. IT-37, pp. 214-216.
21. Rutkowski L. (1993): *Multiple Fourier series procedures for extraction of nonlinear regressions from noisy data,* IEEE Transactions on Signal Processing, Vol. 41, No. 10, pp. 3062-3065.
22. Rutkowski L., and Gałkowski T. (1994): *On pattern classification and system identification by probabilistic neural networks,* Applied Mathematics and Computer Science, Vol. 4, No.3, pp. 413-422.
23. Rutkowska D. and Rutkowski L. (1999): *Neural-Fuzzy-Genetic Parallel Computing System as a Tool for Various Applications,* Proceedings of the Third International Conference on Parallel Processing & Applied Mathematics (PPAM'99), Kazimierz Dolny, pp. 489-498.
24. Rutkowski L. and Cpałka K. (2000): *Flexible Structures of Neuro - Fuzzy Systems,* Quo Vadis Computational Intelligence, Studies in Fuzziness and Soft Computing, Vol. 54, pp. 479 - 484, Springer.
25. Cpałka K. and Rutkowski L. (2000): *Soft Neuro - Fuzzy Systems,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 296-301, June 6-10, Zakopane.
26. Rutkowski L. and Starczewski J. (2000): *From type-1 to type-2 fuzzy interference systems - part 1,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 46-51, June 6-10, Zakopane.
27. Rutkowski L. and Starczewski J. (2000): *From type-1 to type-2 fuzzy interference systems - part 2,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 52-65, June 6-10, Zakopane.
28. Scherer R. and Rutkowski L. (2000): *A survey of hierarchical fuzzy systems,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 374-379, June 6-10, Zakopane.
29. Cpałka K. and Rutkowski L. (2001): *Compromise neuro - fuzzy system,* Fourth International Conference on Parallel Processing and Applied Mathematics, pp. 33 - 40, Czestochowa.

30. Scherer R. and Rutkowski L. (2001): *A neuro - fuzzy relational system,* Fourth International Conference on Parallel Processing and Applied Mathematics, pp. 131 - 135, Czestochowa.
31. Rutkowski L. and Cpałka K. (2001): *A general approach to neuro - fuzzy systems,* Proceedings of the 10th IEEE International Conference on Fuzzy Systems, Melbourne, published on CD, December 2-5.
32. Starczewski J. and Rutkowski L. (2001): *Connectionist Structures of Type 2 Fuzzy Inference Systems,* Lecture Notes in Computer Science, Vol. 2328, pp. 634-642.
33. Rutkowski L. and Cpałka K. (2002): *A neuro-fuzzy controller with a compromise fuzzy reasoning,* Control and Cybernetics, Vol. 31, No. 2, pp. 297-308.
34. Rutkowski L. and Cpałka K. (2002): *Compromise approach to neuro-fuzzy systems,* Proceedings of the 2nd Euro-International Symposium on Computational Intelligence, Vol. 76, pp. 85-90, Koszyce.
35. Scherer R. and Rutkowski L. (2002): *Relational equations initializing neuro-fuzzy system,* 10th Zittau Fuzzy Colloquium, pp. 212-217, Germany, September 4-6.
36. Rutkowski L. and Cpałka K. (2002): *Flexible weighted neuro-fuzzy systems,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.
37. Scherer R. and Rutkowski L. (2002): *Neuro-fuzzy relational systems,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.
38. Starczewski J. and Rutkowski L. (2002): *Neuro-fuzzy inference systems of type 2,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.
39. Rutkowski L. and Cpałka K. (2003): *Compromise weighted neuro-fuzzy systems,* in: Rutkowski L., Kacprzyk J. (Eds.), Neural Networks and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg , New York, pp. 557-562.
40. Scherer R. and Rutkowski L. (2003): *A fuzzy relational system with linguistic antecedent certainty factors,* in: Rutkowski L., Kacprzyk J. (Eds.), Neural Networks and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg , New York, pp. 563-569.
41. Starczewski J. and Rutkowski L. (2003): *Interval type 2 neuro-fuzzy systems based on interval consequents,* in: Rutkowski L., Kacprzyk J. (Eds.), Neural Networks and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg , New York, pp. 570-577.
42. Rutkowski L. and Cpałka K. (2003): *Flexible neuro-fuzzy systems,* IEEE Transactions on Neural Networks, vol. 14, pp. 554-574.
43. Rutkowski L, (2004): *Adaptive probabilistic neural-networks for pattern classification in time-varying environment,* IEEE Trans. Neural Networks, vol. 15, March.
44. Rutkowski L, (2004): *New Soft Computing Techniques for System Modelling, Pattern Classification and Image Processing,* Springer-Verlag.
45. Rutkowski L, (2004): *Flexible Neuro-Fuzzy Systems: Structures, Learning and Preformance Evaluation,* Kluwer.

On Designing of Neuro-Fuzzy Systems

Robert Nowicki[1], Agata Pokropińska[1], and Yoichi Hayashi[2]

[1] Department of Computer Engineering, Technical University of Czestochowa,
Al. Armii Krajowej 36, 42-200 Czestochowa, POLAND,
{rnowicki,agatap}@kik.pcz.czest.pl,
[2] Department of Computer Science, Meiji University,
Tama-ku, Kawasaki 214-8571, JAPAN,
hayashiy@cs.meiji.ac.jp

Abstract. This paper presents a new method to design the neuro-fuzzy systems. The procedure is composed of several separated techniques such as the WTA algorithm developed for fuzzy sets, learning from exceptions and the gradient learning for neuro-fuzzy systems. The main goal of the presented algorithm is to find the simplest neuro-fuzzy system which meets design requirements; the system should be built with the smallest number of elements. As the performance measure we take the mean square error or a number of mistakes in the classification. The alternative way based on the reduction of rules is also presented for comparison. The results of an experimental research are depicted for both methods.

1 Introduction

In the last decade various neuro-fuzzy systems (NFS) have been developed. They include Mamdani systems [1, 3, 23, 26], logical systems [5, 6, 10, 12, 14, 15, 17, 20], relational systems [11, 16, 18, 21], type-2 systems [7, 8, 13, 19, 22], hierarchical sytems [9] and others [2, 4]. They are characterized by learning properties and natural language description. The neuro-fuzzy systems have been applied to pattern classification, system identyfication, approximation, prediction and control. The main problem is to design neuro-fuzzy systems which are transparent and give good performance in terms of accuracy (mean square error or percentage of mistakes). In this paper we present two methods for designing neuro-fuzzy systems. The results are illustrated on a classification problem.

2 Neuro-Fuzzy Systems

The neuro-fuzzy architectures comprise the fuzzifier, fuzzy inference engine, fuzzy rule base and defuzzifier in one structure. In most cases the fuzzifier applies the singleton fuzzification

$$\mu_{A'}(\mathbf{v}) = \begin{cases} 1 \text{ if } \mathbf{v} = \overline{\mathbf{v}} \\ 0 \text{ if } \mathbf{v} \neq \overline{\mathbf{v}} \end{cases} \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 641–649, 2004.
© Springer-Verlag Berlin Heidelberg 2004

where $\mathbf{v} = [v_1, \ldots, v_n] \in \mathbf{V}$ is the input linguistic variable, $\overline{\mathbf{v}} = [\overline{v}_1, \ldots, \overline{v}_n] \in \mathbf{V}$ is the input, i.e. crisp value of $\mathbf{v}$ and A' is the input fuzzy set used in promise $\mathbf{v}$ is A'.

In classification task the fuzzy rule base includes rules in the form

$$R^{(r)} : \text{IF } \mathbf{v} \text{ is } A^r \text{ THEN } x \in \omega_1 \left(z_1^r\right), x \in \omega_2 \left(z_2^r\right), \ldots x \in \omega_m \left(z_m^r\right) \qquad (2)$$

where z_j^k is the membership degree of object x to the j-th class. Moreover, we assume that

$$z_j^r = \begin{cases} 1 \text{ if } x \in \omega_j \\ 0 \text{ if } x \notin \omega_j \end{cases}$$

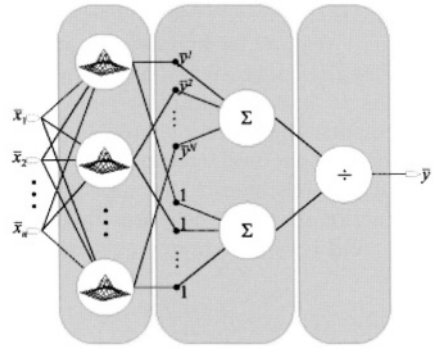


Fig. 1. The neuro-fuzzy architecture with "engineering implications".

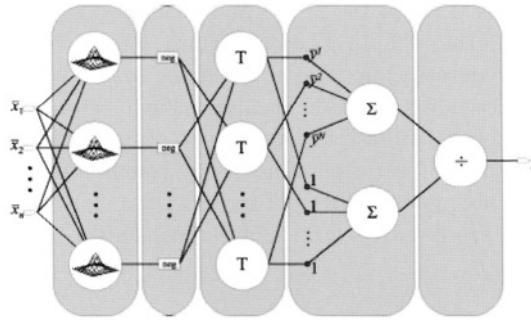


Fig. 2. The neuro-fuzzy architecture with S-implication.

The inference process is performed by the composition inference rule expressed as follows

$$\overline{B}_j^r = A' \circ \left(A^r \rightarrow B_j^r\right)$$

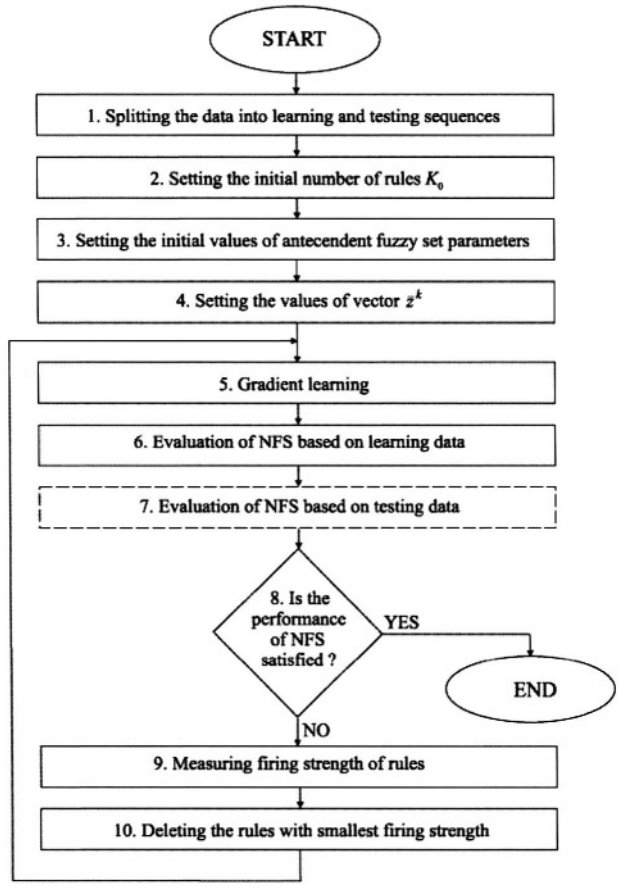


Fig. 3. Designing of NFS based on deleting the rules with smallest firing strength.

The sup –T is commonly used definition of composition, so the membership function of inference is given by

$$\mu_{\overline{B}_j^r}(z_j) = \sup_{\mathbf{v}\in\mathbf{V}} \left\{ \mu_{A'}(\mathbf{v}) \overset{T}{*} \mu_{A^r \to B_j^r}(\mathbf{v}, z_j) \right\} \tag{3}$$

where $\overset{T}{*}$ is any t-norm. The aggregation operator, applied in order to obtain the fuzzy set B_j' based on fuzzy sets $\overline{B}_j^r$, is the t-norm or t-conorm operator, depending on the type of fuzzy implication. In case of Mamdani approach, the aggregation is carried out by

$$B_j' = \bigcup_{r=1}^{N} \overline{B}_j^r \tag{4}$$

When we use the logical model, the aggregation is carried out by

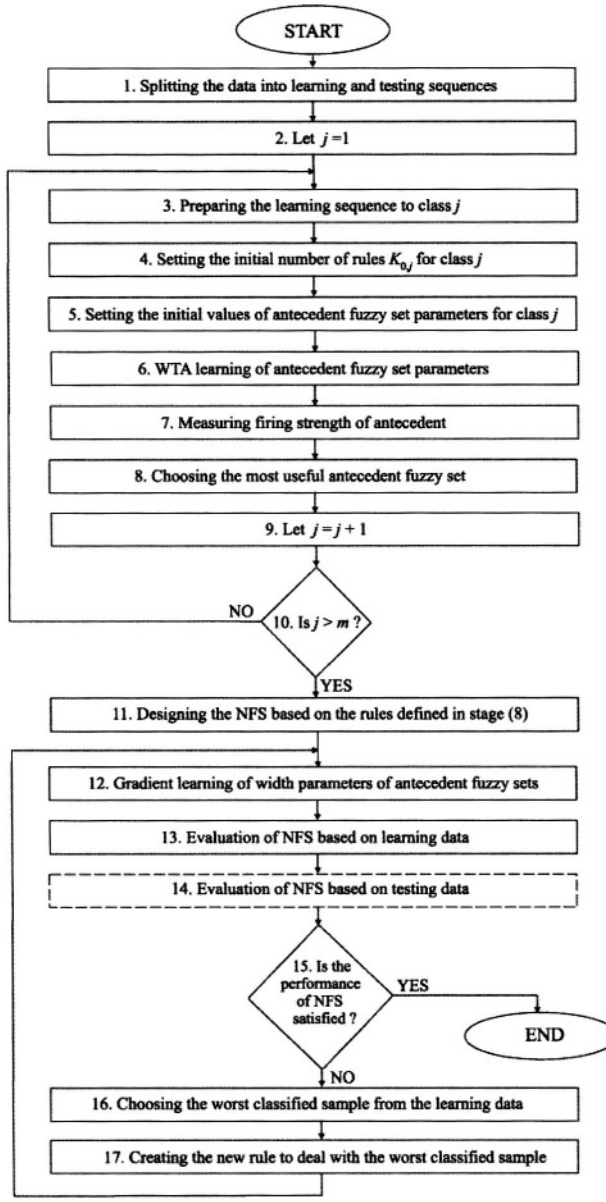


Fig. 4. Designing of NFS based on the WTA algorithm.

$$B'_j = \bigcap_{r=1}^{N} \overline{B}_j^r \tag{5}$$

The crisp output value of a fuzzy system is determined in the process of defuzzification. In classification problems we need the fuzzy system having the same number of outputs as number of classes. So, in such problems the defuzzification

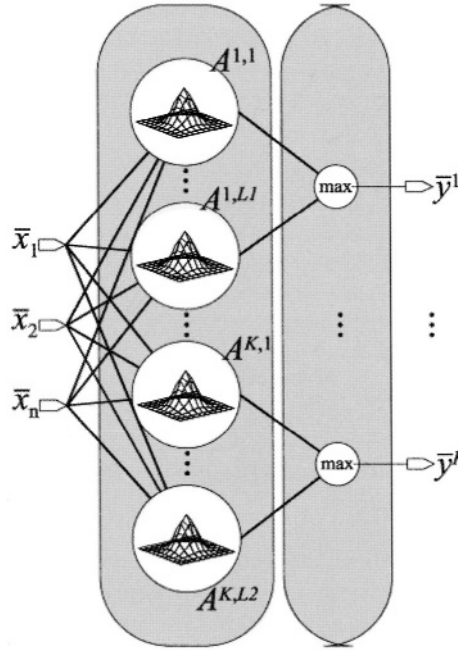


Fig. 5. A part of the neuro-fuzzy architecture for the WTA learning.

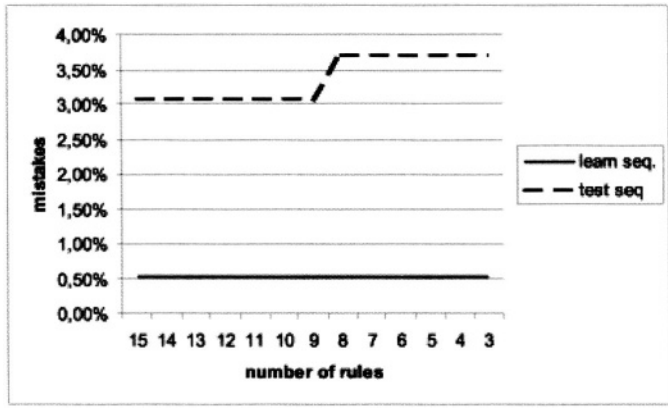


Fig. 6. Mistakes versus number of rules - method based on rule pruning.

for the j-th output is described by

$$\overline{z}_j = \frac{\sum_{r=1}^{N} \overline{z}_j^r \mu_{B'}\left(\overline{z}_j^r\right)}{\sum_{r=1}^{N} \mu_{B'}\left(\overline{z}_j^r\right)} \tag{6}$$

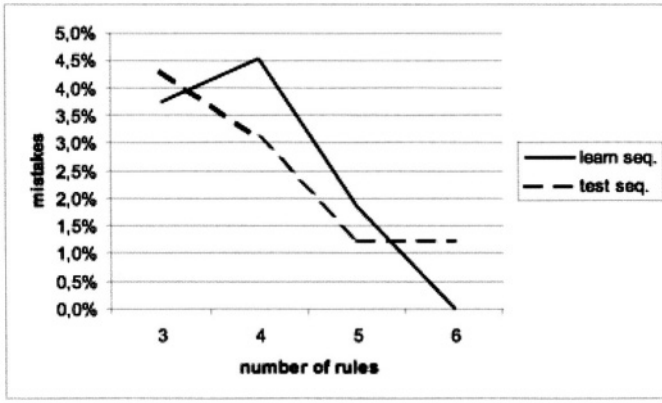


Fig. 7. Mistakes versus number of rules - method based on the WTA learning.

The complete description of the neuro-fuzzy system depends on the chosen definition of a fuzzy implication, i.e. form of expression $\mu_{A^r \to B_j^r}(\mathbf{v}, z_j)$ in (3). However with some simplified assumptions we obtain a common description of architectures for whole groups of fuzzy implication. We present two groups as an example. For the so-called Mamdani approach we get

$$\overline{z}_j = \frac{\sum_{r=1}^{N} \overline{z}_j^r \mu_{A^r}(\overline{\mathbf{v}})}{\sum_{k=1}^{N} \mu_{A^k}(\overline{\mathbf{v}})} \tag{7}$$

Figure 1 shows the architecture described by (7). For S-implications (logical approach) the complete description of the neuro-fuzzy system is expressed by

$$\overline{z}_j = \frac{\sum_{r=1}^{N} \overline{z}_j^r \underset{k=1,k\neq r}{\overset{N}{T}} \{1 - \mu_{A^r}(\overline{\mathbf{v}})\}}{\sum_{k=1}^{N} \underset{k=1,k\neq r}{\overset{N}{T}} \{1 - \mu_{A^r}(\overline{\mathbf{v}})\}} \tag{8}$$

Figure 2 shows the architecture described by (8).

The values $\overline{z}_j^r$ are related to the learning data set and the main problem is to find parameters of fuzzy sets A^r and the number of rules N.

3 Two Methods of NFS Designing

In this section we describe two methods for designing neuro-fuzzy systems. The first method is based on the rule pruning (see [1]). In the algorithm shown in Fig. 3 we measure the firing strenght of rules and delete the rules with smallest values

of the firing strenghts. The second method employs the WTA learning algorithm. The algorithm is illustrated in Fig. 4. The step 6 in Fig. 4 is concerned only with a part of the neuro-fuzzy architecture shown in Fig. 5. We determine in the process of learning parameters w of a fuzzy set A^r, $r = 1, \ldots, N$, for which the membership function $\mu_{A^r}(\mathbf{v})$ attains the maximum in each iteration of learning. The learning process is described by formula

$$w(t+1) = \begin{cases} w(t) + \eta \frac{\partial \mu_{A^r}(\mathbf{v})}{\partial w} (1 - \mu_{A^r}(\mathbf{v})) \text{ if } x \in \omega_j \\ w(t) + \eta \frac{\partial \mu_{A^r}(\mathbf{v})}{\partial w} (0 - \mu_{A^r}(\mathbf{v})) \text{ if } x \notin \omega_j \end{cases}$$

4 Experimental Results

We will illustrate the designing process on a typical benchmark problem. The task is to classify the samples of wine. They come from three different vineyards in the same region of Italy. The input to the inference process contains 13 parameters of wine samples, i.e. alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines and proline.

For the method based on rule pruning we started with 15 rules. The results are depicted in Fig. 6.

For the method based on the WTA learning we started with 30 rules from which 3 were chosen based on the WTA algorithm. With growing number of rules the number of mistakes decreases and the best result is achieved with 6 rules. The results are depicted in Fig. 7.

5 Conclusions

In the paper a new idea for designing neuro-fuzzy systems has been presented. The method based on the WTA algorithm leads to better results i.e. smaller number of mistakes (compare Fig. 6 and 7). In this method we can always reduce the number of mistakes in a learning sequence to zero. It should be noted that alternative, and functionally equivalent, techniques to neuro-fuzzy systems are based on probabilistic neural networks applied to pattern classification [24] and regression estimation [25] in stationary and non-stationary environment.

References

1. Gorzałczany M. B. (2002): *Computational Inteligence Systems and Applications, Neuro-Fuzzy and Fuzzy Neural Synergisms,* Studies in Fuzziness and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg.
2. Hayashi Y., Buckley J. J., Czogała E. (1993): *Fuzzy neural networks with fuzzy signals and weights,* Int. J. Intell. Syst., vol. 8, no. 4, pp. 527-537.
3. Rutkowska D. and Rutkowski L. (1999): *Neural-Fuzzy-Genetic Parallel Computing System as a Tool for Various Applications,* Proceedings of the Third International Conference on Parallel Processing & Applied Mathematics (PPAM'99), Kazimierz Dolny, pp. 489-498.

4. Scherer R. and Rutkowski L. (1999): *Survey of selected fuzzy rule pruning methods,* Proceedings of the Fourth Conference Neural Networks and Their Applications, pp. 308-313, 1999.
5. Rutkowski L. and Cpałka K. (2000): *Flexible Structures of Neuro - Fuzzy Systems,* Quo Vadis Computational Intelligence, Studies in Fuzziness and Soft Computing, Vol. 54, pp. 479 - 484, Springer.
6. Cpałka K. and Rutkowski L. (2000): *Soft Neuro - Fuzzy Systems,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 296-301, June 6-10, Zakopane.
7. Rutkowski L. and Starczewski J. (2000): *From type-1 to type-2 fuzzy interference systems - part 1,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 46-51, June 6-10, Zakopane.
8. Rutkowski L. and Starczewski J. (2000): *From type-1 to type-2 fuzzy interference systems - part 2,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 52-65, June 6-10, Zakopane.
9. Scherer R. and Rutkowski L. (2000): *A survey of hierarchical fuzzy systems,* Proceedings of the Fifth Conference Neural Networks and Soft Computing, pp. 374-379, June 6-10, Zakopane.
10. Cpałka K. and Rutkowski L. (2001): *Compromise neuro - fuzzy system,* Fourth International Conference on Parallel Processing and Applied Mathematics, pp. 33 - 40, Czestochowa.
11. Scherer R. and Rutkowski L. (2001): *A neuro - fuzzy relational system,* Fourth International Conference on Parallel Processing and Applied Mathematics, pp. 131 - 135, Czestochowa.
12. Rutkowski L. and Cpałka K. (2001): *A general approach to neuro - fuzzy systems,* Proceedings of the 10th IEEE International Conference on Fuzzy Systems, Melbourne, published on CD, December 2-5.
13. Starczewski J. and Rutkowski L. (2001): *Connectionist Structures of Type 2 Fuzzy Inference Systems,* Lecture Notes in Computer Science, Vol. 2328, pp. 634-642.
14. Rutkowski L. and Cpałka K. (2002): *A neuro-fuzzy controller with a compromise fuzzy reasoning,* Control and Cybernetics, Vol. 31, No. 2, pp. 297-308.
15. Rutkowski L. and Cpałka K. (2002): *Compromise approach to neuro-fuzzy systems,* Proceedings of the 2nd Euro-International Symposium on Computational Intelligence, Vol. 76, pp. 85-90, Koszyce.
16. Scherer R. and Rutkowski L. (2002): *Relational equations initializing neuro-fuzzy system,* 10th Zittau Fuzzy Colloquium, pp. 212-217, Germany, September 4-6.
17. Rutkowski L. and Cpałka K. (2002): *Flexible weighted neuro-fuzzy systems,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.
18. Scherer R. and Rutkowski L. (2002): *Neuro-fuzzy relational systems,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.
19. Starczewski J. and Rutkowski L. (2002): *Neuro-fuzzy inference systems of type 2,* 9th International Conference on Neural Information Processing (ICONIP'02), Orchid Country Club, Singapore, November 18-22.
20. Rutkowski L. and Cpałka K. (2003): *Compromise weighted neuro-fuzzy systems,* in: Rutkowski L., Kacprzyk J. (Eds.), Neural Networks and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg , New York, pp. 557-562.

21. Scherer R. and Rutkowski L. (2003): *A fuzzy relational system with linguistic antecedent certainty factors,* in: Rutkowski L., Kacprzyk J. (Eds.), Neural Networks and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg , New York, pp. 563-569.
22. Starczewski J. and Rutkowski L. (2003): *Interval type 2 neuro-fuzzy systems based on interval consequents,* in: Rutkowski L., Kacprzyk J. (Eds.), Neural Networks and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg , New York, pp. 570-577.
23. Rutkowski L. and Cpałka K. (2003): *Flexible neuro-fuzzy systems,* IEEE Transactions on Neural Networks, vol. 14, pp. 554-574.
24. Rutkowski L, (2004): *Adaptive probabilistic neural-networks for pattern classification in time-varying environment,* IEEE Trans. Neural Networks, vol. 15, March.
25. Rutkowski L, (2004): *New Soft Computing Techniques for System Modelling, Pattern Classification and Image Processing,* Springer-Verlag.
26. Rutkowski L, (2004): *Flexible Neuro-Fuzzy Systems: Structures, Learning and Performance Evaluation,* Kluwer.

Multi-expert Systems

Danuta Rutkowska

Department of Computer Engineering, Technical University of Czestochowa,
Armii Krajowej 36, 42-200 Czestochowa, Poland,
drutko@kik.pcz.czest.pl,
http://kik.pcz.czest.pL

Abstract. In this paper, a multi-expert classification system (MECS), composed of two main parts performing the so-called multi-stage classification (MSC) and multi-expert classification (MEC), is proposed. The former (MSC) produces either correct decisions or the "I do not know" (IDNK) answers, so there are not misclassifications. The latter (MEC) is a parallel system that includes different classifiers, for the objects not classified by the MSC system (resulting in the IDNK answers). A medical diagnosis example illustrates the perception-based approach employed in the MSC system, and the need for application of the MEC system.

1 Introduction

Expert systems, in Artificial Intelligence, are computer programs that try to emulate the reasoning process of a human expert within a specific domain of knowledge. For example, an expert system in medical application domain may infer a diagnosis based on measurements of attributes of a disease. The system of this kind imitates a medical doctor who is the human expert with regard to the medical diagnosis.

A classification problem is usually the main task of the diagnosis systems. Results of the classification depend on the data, i.e. the measurements of the attributes. The data vectors represent records of the attribute values, which characterize objects being classified. It may be easy or very difficult for an expert system to infer correct classification decisions for particular objects. If a data vector belongs to a region of overlapping classes of diagnosis, very often the system carries out a misclassification.

It is very important, especially in medical diagnosis, to have systems that perform without misclassifications. Thus, in this paper, the so-called perception-based classification method, proposed by the author, is applied to recognize those input data vectors which are easiest to be classified. With regard to the objects which are most difficult to be assigned to proper classes, the system that employs this method carries out "I do not know" answers concerning the classification decisions.

A number of various classification methods can be found in the literature. However, they usually accept some misclassifications. The main aim of the expert systems considered in this paper is to produce always correct decisions, and only

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 650–658, 2004.
© Springer-Verlag Berlin Heidelberg 2004

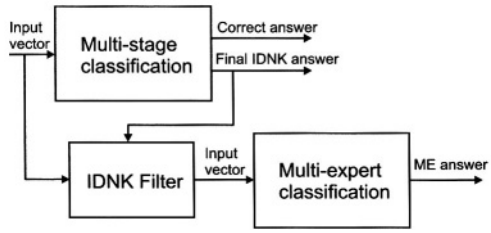


Fig. 1. Multi-expert classification system (MECS)

some "I do not know" answers instead of misclassifications. Of course, the "I do not know" response should not occur often.

With regard to the region of overlapping classes, that contains data vectors which are most difficult to be recognized, in this paper, the approach of applying many classification methods, in parallel, is proposed. In this way, decisions of many experts are incorporated into the perception-based classification system. Thus, the system of this kind is called the multi-expert system. An example of such systems, for medical diagnosis applications, is presented; see Section 4.

2 Multi-expert Classification Systems

In this paper, the multi-expert classification system (MECS), illustrated in Fig.1, is proposed. The system is composed of two main parts, which perform the so-called multi-stage classification (MSC) and multi-expert classification (MEC). The former is employed in order to recognize those input data vectors which are easiest to be classified. In this way, all the cases that can be assigned to proper classes without any doubth receive correct answers, carried out by the MSC unit, concerning the class to which the input vector belongs. Other input data vectors, those which are not so easy to be correctly classified by the MSC unit, are transmitted by the IDNK filter, to the MEC unit. Those data vectors receive the final IDNK (which means "I do not know") answer, produced by the MSC unit. Thus, only the input data vectors with the final IDNK answer are allowed, by the IDNK filter, to enter the MSC unit. The latter part of the MECS, i.e the MEC unit, performs the classification of the data vectors which are most difficult to be classified. This unit realizes a parallel classification by many expert classifiers.

The classification methods realized by the MSC and MEC units, respectively, are described in Sections 3 and 5. Different multi-expert systems can be constructed depending on the classification algorithms incorporated into the MEC unit. These algorithms can be performed in parallel, and the output answer of the MEC unit is obtained as an aggregation of the answers produced by each of the expert classifiers.

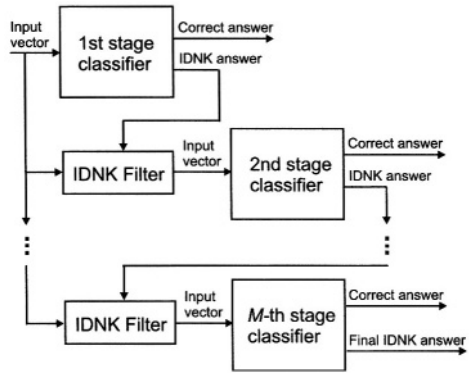


Fig. 2. Multi-stage classification (MSC unit)

3 Perception-Based Multi-stage Classification

In this section, the MSC unit, which is the first part of the MECS system portrayed in Fig.1, is considered. The MSC unit is illustrated in Fig.2. This is the classifier that performs the so-called perception-based multi-stage classification, proposed in [7], [8].

The MSC unit is composed of M classifiers corresponding to M stages of the classification process. Each of these classifiers is a neuro-fuzzy system, in the form of a connectionist network, that reflect fuzzy IF-THEN rules, which are perception-based rules [7], [8]. The perception-based approach refers to the computational theory of perceptions, introduced by Zadeh [9].

The number of the classifiers, M, depends on the data to be classified. For example, the medical data applied in [7] need only one or two stages, and in addition, the MEC units is not required. The iris classification problem [6], with the well-known iris data [2], can be solved by use of four stages of the MSC, and does also not need the MEC units; the results are described in [8]. However, the medical data employed in this paper are much more difficult for classsification, so the MECS system that includes both MSC and MEC units is necessary. The multi-stage method, in application to this medical diagnosis problem, is illustrated in Section 4.

The neuro-fuzzy classifiers, at every stage of the MSC unit, are represented in the form of the connectionist networks portrayed in Fig.3. The components of the input vectors are values of the attributes that characterize objects to be recognized (classified). The output values correspond to the classes, and the maximal value points out the class to which the input vector belongs, according to the inference performed by this neuro-fuzzy system. The nodes (neurons) of the first layer of this network realize membership functions of the fuzzy sets in the antecedent parts of the fuzzy IF-THEN rules. The nodes of the second layer perform the multiplication which realizes the Cartesian product of the antecedent fuzzy sets. Thus, the output values of these nodes express the so-called degree

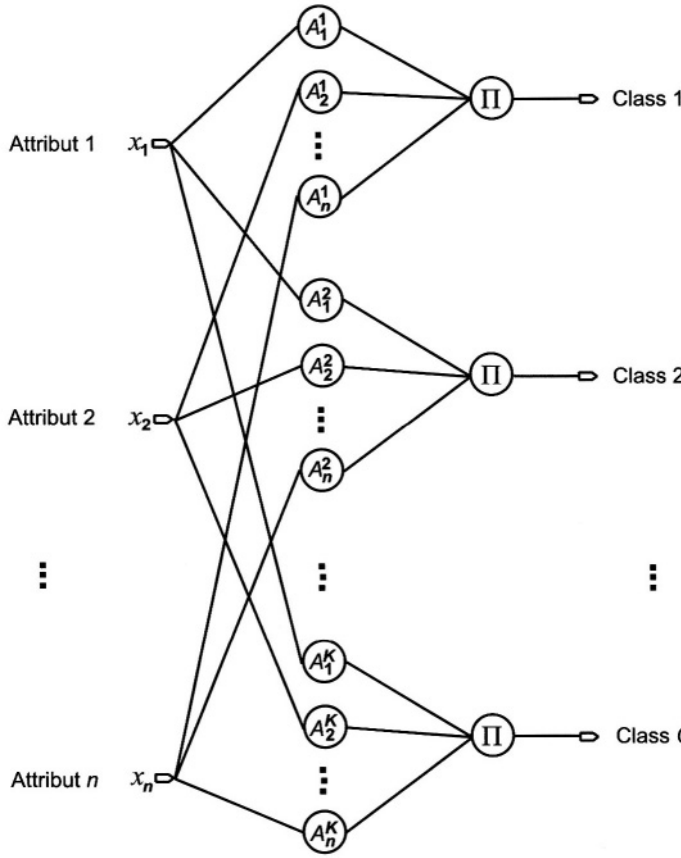


Fig. 3. Neuro-fuzzy classifier

of rule activation (firing strength), so the maximal value of the rule activation corresponds to the rule with the proper class, which is indicated in the consequent (conclusion) part of this rule. The inference process, in the neuro-fuzzy classifiers, is based on the fuzzy IF-THEN rules, which are formulated as the perception-based rules. This means that the fuzzy sets represent the linguistic terms that express our perceptions concerning the features (attributes) which characterize the objects being classified. Examples of such rules can be found in [7], [8], as well as in Section 4, in this paper. With regard to the multi-stage classification, shapes of the membership functions of the fuzzy sets are very important. Trapezoidal membership functions, determined using the fuzzy granulation of the data set, should be applied; see Section 4.

The main advantage of the perception-based neuro-fuzzy classifier is its performance without misclassifications. This means that the classifier produces either correct responses or the "I do not know" answers. The data vectors which are not classified at the 1st stage (IDNK answers) are transmitted by the IDNK

filter to the 2nd stage classifier. Then, the data vectors with the IDNK answer produced by the 2nd stage neuro-fuzzy classifier are allowed, by the IDNK filter, to enter the 3rd stage classifier, and so on. The data vectors that cannot be classified by the last (M-th stage) neuro-fuzzy classifier may proceed to the MEC unit.

4 Multi-stage Classifier for Heart Disease Diagnosis

As an illustration of the application of the MECS system, proposed in this paper, the heart disease data from the Cleveland Clinic Foundation, available on the Internet [4], have been employed in order to solve the medical diagnosis problem. The particular data items contain 14 attributes which represent features such as age, sex, chest pain type, resting blood pressure, serum cholestorol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise induced angina, etc. The final attribute is the diagnosis of heart disease, expressed as values 0 for absence of the disease, and values 1, 2, 3, 4 for presence of the heart disease. Usually, most experiments with the Cleveland database have been concentrated on simply attempting to distinguish presence from absence of the disease. In this paper, 5 classes are considered: one for absence of the disease (negative diagnosis — class 0) and four classes for presence of various kinds of the heart disease (positive diagnosis — classes 1, 2, 3, 4). The final attribute (diagnosis) is the predicted attribute, which depends on the values of the 13 previous attributes, which characterize patients to be recognized as healthy (absence of the heart disease) or sick (having the heart disease). Values of these attributes are components of the input data vectors to be classified. The data set contains 303 data items (data vectors).

In order to solve the classification problem (medical diagnosis), the perception-based fuzzy IF-THEN rules have been formulated based on the data set. At the first stage, as well as at the next ones, the number of these rules equals to the number of the classes, so 5 rules have been used at every stage of the perception-based multi-stage classification method, performed by the MSC unit. Thus, at every stage, the neuro-fuzzy network, shown in Fig.3, includes the number of the nodes determined by $K = C = 5$, and $n = 13$. The fuzzy IF-THEN rules, reflected by this neuro-fuzzy network, are formulated as follows

$$R^{(k)}: \textbf{IF } x_1 \text{ is } A_1^k \text{ AND } x_2 \text{ is } A_2^k \text{ AND } \ldots \text{ AND } x_n \text{ is } A_n^k \quad \textbf{THEN Class } j \tag{1}$$

where $k = 1, \ldots, K$, and $j = 1, \ldots, C$.

The fuzzy sets A_i^k, for $i = 1, \ldots, n$, represent linguistic descriptions of the perceptions about attribute values. These fuzzy sets are characterized by trapezoidal membership functions, determined based on the granulated ranges of the values of the attributes. For deatails concerning the fuzzy granulation, applied in this method, see [7], [8].

Table 1. Results of the classification by the MSC unit for the heart disease diagnosis

stage	Class 0	Class 1	Class 2	Class 3	Class 4	total
1	12	1	1	3	1	18
2	15	1	1	1	2	20
3	2	1	0	3	1	7
4	3	0	1	0	1	5
5	0	0	0	2	0	2
6	2	1	1	0	0	4
7	2	0	1	0	1	4
8	0	0	0	0	0	0
total	36	4	5	9	6	60

For the heart disease diagnosis problem, the MSC unit with 8 stages formed by the neuro-fuzzy classification based on the rules (1) have been applied, resulting in 60 data vectors correctly classified. This means that only about 20% input vectors have been assigned to proper classes by the MSC unit. Other data vectors have received the IDNK answer at the output of the MSC unit. Thus, those data vectors should be classified by the MEC unit.

Table 1 illustrates how many data vectors have been properly assigned to particular classes at each stage of the classification performed by the MSC unit. The last row shows total numbers of input vectors correctly classified by the MSC unit.

The heart disease data require the MEC unit to continue the classification task. However, it is worth reminding that there are medical diagnosis problems (for example that presented in [7]), which can be solved, resulting in correct classification decision for every (or almost every) input data, by use only the MSC unit.

5 Parallel Multi-expert Classification

In this section, the MEC unit of the MECS system, portrayed in Fig.1, is considered. The MEC unit is the multi-expert classifier, illustrated in Fig.4. The MEC unit is composed of N expert classifiers, which perform different classification algorithms, and work in parallel. The input vectors are those vectors which are too difficult to be classified by the MSC unit. Therefore, other classification methods are employed, and the results inferred by every expert classifier can be aggregated, for instance as a majority votes of the experts, and accepted as the multi-expert decision (ME answer).

The are many classification methods known in the literature, so the particular expert classifiers can realize any of these methods, for example probabilistic algorithms, various neural or neuro-fuzzy classifiers, hybrid methods with evolutionary algorithms, and many others; see e.g. [1], [3]. The number of the classification methods applied (with many variants of these methods, e.g. differ-

ent architectures of neuro-fuzzy systems [5]) equals to the number of the expert classifiers, *N,* in the MEC unit.

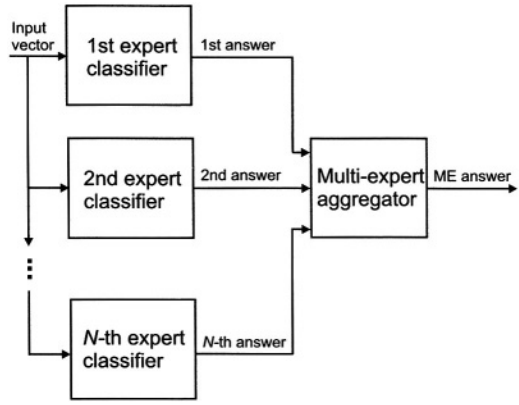


Fig. 4. Multi-expert classification (MEC unit)

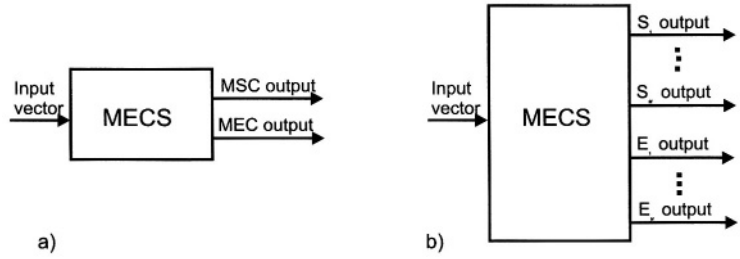


Fig. 5. Multi-expert classification system: two-output (a), and multi-output (b)

6 Conclusions

The multi-expert classification system, proposed in this paper, can be considered as a two-output or multi-output system; see Fig.5. The former is the MECS with only one output from the MSC unit (MSC output) and only one output from the MEC unit (MEC output). The latter MECS is the system with *M* outputs from the MSC unit, and *N* outputs from the MEC unit.

In the two-output MECS, if a decision, concerning the class to which the input vector belongs, is produced at the first output (MSC output), this means that this is the correct decision. If the MSC output shows the IDNK answer, the

decision concerning the class to which the input vector belongs is displayed at the MEC output, but in this case the answer is an aggregated opinion of every expert classifier (ME answer).

In the multi-output MECS, we can observe decisions produced by the neuro-fuzzy classifiers at every stage of the MSC unit, shown at the $S_1, \ldots, S_M$ outputs, respectively, and the answers of every expert classifiers (of the MEC unit), displayed at the $E_1, \ldots, E_N$ outputs. In this case, we have more information about the classification results. Apart from the decision about the class to which the input vector belongs, we know at what stage the decision has been produced by the neuro-fuzzy classifier, and with regard to the MEC unit we know the decisions of particular expert classifiers. The information about the stage of the MSC is important, because it allows to evaluate how difficult is to classify the input vector. If it has been classified at the first stage, so it was very easy, this means that the decision is correct without any doubt. However other classification results inferred by the MSC unit are also correct, but not so easily obtained as at the first stage. The information about decisions of each particular expert classifier, carried out independently, in parallel, is also very important. For example, the majority votes of the experts gives a decision, but does not provide an information about a percentage of the majority votes, which expresses the difficulty of the input vector classification.

It should be emphasized that the main advantage of the MECS system, proposed in this paper, is to perform (for at least some amount of input data) a classification task without misclassifications. This is especially important with regard to medical diagnosis applications, where a mistake can be very dangerous for the patient being recognized with regard to a specific disease. The MECS system allows to distinguish the cases properly classified by the MSC unit from those being recognized by the MEC unit, while most of classification systems accept some percetage of misclassifications, without any additional information concerning particular data items.

References

1. Duda R.O., Hart P.E.: Pattern Classification and Sciene Analysis. John Wiley & Sons. New York (1973)
2. Fisher R.A.: The use of multiple measurements in taxonomic problems. Ann. Eugenics **7** (1936) 179-188
3. Kuncheva L.I.: Fuzzy Classifier Design. Physica-Verlag. A Springer-Verlag Company. Heidelberg. New York (2000)
4. Mertz C.J., Murphy P.M.: UCI repository of machine learning databases. http://www.ics.uci.edu/pub/machine-learning-databases
5. Rutkowska D.: Neuro-Fuzzy Architectures and Hybrid Learning. Physica-Verlag. A Springer-Verlag Company. Heidelberg. New York (2002)
6. Rutkowska D.: A perception-based classification system. Proc. CIMCA 2003 Conference. Vienna. Austria (2003) 52-61
7. Rutkowska D.: Perception-based systems for medical diagnosis. Proc. Third EUSFLAT 2003. Zittau. Germany (2003) 741-746

8. Rutkowska D.: Perception-based expert systems. Soft Computing Journal (2003) submitted
9. Zadeh L.A.: From computing with numbers to computing with words – from manipulation of measurements to manipulation of perceptions. IEEE Trans. Circuits and Systems - I: Fundamental Theory and Applications. **45**.1 (1999) 105-119

New Methods for Uncertainty Representations in Neuro-Fuzzy Systems

Rafał Scherer[1], Janusz Starczewski[1], and Adam Gawęda[2]

[1] Department of Computer Engineering, Częstochowa University of Technology, Al. Armii Krajowej 36, 42-200 Częstochowa, Poland,
{rscherer,jasio}@kik.pcz.czest.pl
[2] University of Louisville, Louisville, KY 40292 USA,
agaweda@kdp.louisville.edu

Abstract. In this paper we discuss a new method for uncertainty representations in neuro-fuzzy systems. Expert uncertainty concerning antecedent fuzzy linguistic values are expressed in the form of linguistic values e.g. roughly, more or less. That idea is incorporated into relational neuro-fuzzy systems. In the paper both type-1 and type-2 fuzzy systems are considered. Numerical simulations of the new fuzzy model are presented.

1 Introduction

Mostly used fuzzy systems are type-1 linguistic and functional (Takagi-Sugeno) models. Linguistic systems store an input-output mapping in the form of fuzzy IF-THEN rules with linguistic terms both in antecedents and consequents. The functional fuzzy systems use linguistic values in the condition part of rules, but the input-output mapping is depicted by functions of inputs in a rule consequent part. The above models are used in all fields of machine learning and computational intelligence. In this paper we design a new relational fuzzy systems of type-2. Relational fuzzy systems (see e.g. [12][30][31][32]) relate input fuzzy linguistic values to output fuzzy linguistic values thanks to discrete fuzzy relation. That allows setting fuzzy linguistic values in advance and fine-tuning model mapping by changing relation elements.

2 Fuzzy Relational Systems

Fuzzy relational models [12] can be regarded as a generalization of linguistic fuzzy systems, where each rule has more than one linguistic value defined on the same output variable, in their consequents. Fuzzy rules in a SISO relational model have the following form

$$\begin{aligned} &R^k: \text{ IF } x \text{ is } A^k \text{ THEN} \\ &y \text{ is } B^1\ (r_{k1}), y \text{ is } B^m\ (r_{km}), \ldots, y \text{ is } B^M\ (r_{kM}) \end{aligned}, \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 659–667, 2004.
© Springer-Verlag Berlin Heidelberg 2004

where r_{km} is a weight, responsible for the strength of connection between input and output fuzzy sets. Relational fuzzy systems store associations between the input and the output linguistic values in the form of a discrete fuzzy relation

$$\mathbf{R}(A, B) \in [0, 1]. \tag{2}$$

Detailed desription of fuzzy and neuro-fuzzy relational systems is given in [31] [32]. In [30] an idea of a fuzzy relational system with linguistic antecedent certainty factors was introduced. Instead of modifying antecedent or consequent linguistic values, additional fuzzy sets were introduced. In this approach, a fuzzy relational system with linguistic values defined on a unitary interval is used. These values are elements of a fuzzy relation matrix R connecting antecedent or consequent linguistic values. In this case, the relation matrix contains fuzzy sets C_{km} defined on a unitary interval

$$\mathbf{R} = \begin{bmatrix} C_{11} & C_{11} & \cdots & C_{1M} \\ C_{21} & C_{22} & \cdots & C_{2M} \\ \vdots & \vdots & C_{km} & \vdots \\ C_{K1} & C_{K2} & \cdots & C_{KM} \end{bmatrix} \tag{3}$$

Then, if we define several fuzzy linguistic values on unitary interval (e.g. see Fig. 1), an expert can express his uncertainty concerning antecedent terms by a linguistic description. In SISO systems, or MISO systems with multidimensional antecedent fuzzy sets, the expert can define rules similar to the following exemplary ones

$$\begin{aligned} R^1 &: \text{IF } \mathbf{x} \text{ is exactly } A^1 \text{ THEN } y \text{ is } B^1 \\ R^2 &: \text{IF } \mathbf{x} \text{ is more or less } A^1 \text{ THEN } y \text{ is } B^2 \\ R^3 &: \text{IF } \mathbf{x} \text{ is roughly } A^1 \text{ THEN } y \text{ is } B^3 \end{aligned} \tag{4}$$

Rules (4) do not modify antecedent values. The membership degree of an antecedent fuzzy set is divided into several intervals by fuzzy sets C_{km}. Instead of defining many antecedent sets we use a smaller number of input fuzzy sets and several sets C_{km}. Every fuzzy set A^k has up to M defined linguistic values C_{km}. In Fig. 1 there is also the set not at all, which meaning is similar to the standard hedge not. It is activated when its input fuzzy set A^k is not active. The inference in this system is similar to the sup-min composition, but min operation is replaced by a membership degree $\mu_{C_{km}}(\tau^k)$, where τ^k is the membership degree of the k-th multivariate input fuzzy set. The vector of crisp memberships is obtained by

$$\mu_m = \underset{k=1}{\overset{K}{\mathrm{S}}} \left[\mu_{C_{km}}(\mu_{A^k}(\bar{\mathbf{x}}))\right] \tag{5}$$

Eq. (5) reflects fuzzy hedge modifier operation. For example instead of quadratic function for concentration operation very, we use a fuzzy set *exactly* (Fig. 1).

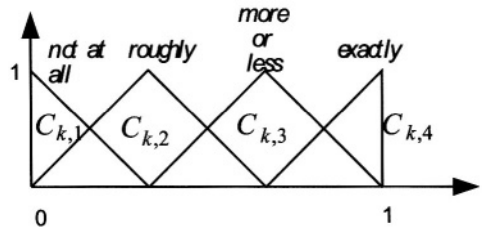


Fig. 1. Example of fuzzy linguistic values, expressing uncertainty in rule antecedents.

Interpretation and operation of the sets in Fig. 1 is different from standard linguistic hedges. For example, standard fuzzy hedge *more or less* dilates an input fuzzy set, whereas our *roughly, more or less* divide membership degree range into several intervals.

3 Type-2 Fuzzy Logic Systems

Type-2 fuzzy logic systems incorporate a concept of equipping each fuzzy set with a fuzzy membership function instead of traditional crisp MF. Fuzzy sets of type-2 are an extension given by Zadeh [39] of classical fuzzy sets. Let us define a fuzzy set of type-2. A fuzzy set of type-2 $\tilde{A}$ in the real line $\mathbb{R}$, is a set

of ordered pairs $\{x, \mu_{\tilde{A}}(x)\}$, which is denoted by $\tilde{A} = \int_{x \in \mathbb{R}} \mu_{\tilde{A}}(x)/x$, where x is an element of the fuzzy set associated with the fuzzy membership grade $\mu_{\tilde{A}}(x)$ being a classical fuzzy subset of the unit interval [0,1], i.e.,

$$\mu_{\tilde{A}}(x) = \int_{u \in [0,1]} f_x(u)/u,$$

where $f_x \colon [0,1] \to [0,1]$. The fuzzy membership grade will be called a fuzzy truth value, since its domain is the truth interval [0,1]. With the use of type-2 fuzzy sets the rule base takes into account uncertainties as to the memberships of antecedents and consequents, which can be described by K rules of the form

$$R^k : \text{IF } x \text{ is } \tilde{A}^k \text{ THEN } y \text{ is } \tilde{B}^k.$$

The intersection of two fuzzy sets of type-2 $\tilde{A}$ and $\tilde{B}$, with their membership functions $\mu_{\tilde{A}}(x) = \int_{u \in [0,1]} f_x(u)/u$ and $\mu_{\tilde{B}}(x) = \int_{v \in [0,1]} g_x(v)/v$, respectively may be computed according to the generalized extension principle. Therefore having arbitrary t-norms T and T_*, an extended operation T based on T_* is as follows

$$\tilde{T}_{T_*}(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \int_{w \in [0,1]} \sup_{T(u,v)=w} T_*(f_x(u), g_x(v))/w. \tag{6}$$

Note that operation T, that has been extended, and T_*, aggregating memberships, are not required to be necessarily the same. This result is however difficult in computing, because the resultant membership grade is the maximum for all equivalent pairs $\{u, v\}$, i.e., such that they produce the same element w. Analogically an extended operation S based on T_* is described by the following formula

$$\tilde{S}_{T_*}\left(\mu_{\tilde{A}}\left(x\right), \mu_{\tilde{B}}\left(x\right)\right) = \int_{w\in[0,1]} \sup_{S(u,v)=w} T_*\left(f_x\left(u\right), g_x\left(v\right)\right)/w.$$

These extended operations together with type reduction procedures allow to construct type-2 fuzzy logic systems (see eg. [7], [9], [10], [36], [38], [37]). The usefullness of the type-2 fuzzy logic approach may be explained by the following example.

Example 1. Suppose we have two mating gear wheels. As input variables we may chose their module pitches, i.e. x_1 and x_2. The best matching of gears is when $x_1 = x_2 = 2.4$ mm. It has been proven empirically that when durability of the mechanism dimminishes to 70% or 50%, depending on the area of testing, when a dimensional tolerance of pitch x_1 is 8% and a tolerance of pitch x_2 is 6%. The type-2 fuzzy logic system makes a selection of matching gears in respect of durability, which should translate itself into the price of the mechanism. The system consists of two rules. The antecedents and the consequents were shown in Fig. 2. The comparison of output signals of the type-2 fuzzy logic system and a system of type-1 (the system has been transformed from the system of type-2 in a way that lower bounds of antecedent membership functions are equal to upper bounds and the 3-rd rule consequent is a singleton at 60%) is shown in Table 1.

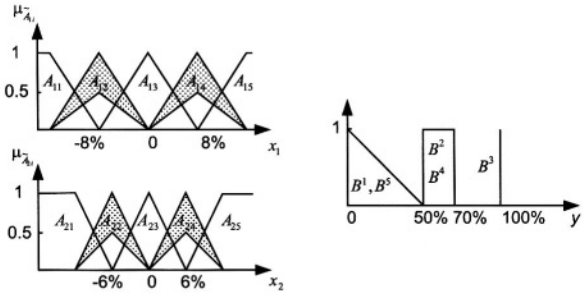


Fig. 2. Antecedents and consequents of exemplary fuzzy logic system of type-2

So far it is hard to find a real problem when some of the fuzzy rules are much more precise than others. Usually some inputs of the system may be considered as more or less certain. But in this kind of problems a type-2 approach cannot

Table 1. Output values of type-1 and type-2 fuzzy logic systems for $x_2 = 5\%$

x_1	y_{T1FLS}	y_{T2FLS}	$y_{minT2FLS}$	$y_{maxT2FLS}$
0%	100%	100%	100%	100%
1%	83.3%	79.9%	76.47%	83.33%
2%	75%	72.02%	69.05%	75%
4%	66.67%	64.71%	61.77%	67.65%
6%	62.5%	61.7%	58.51%	64.9%
8%	60%	60%	56%	63.33%

overperform a traditional type-1 approach, although in [7], [9], [10] the contrary has been shown. Unfortunately in cases when all antecedent and consequent type-2 fuzzy sets are characterized with the same uncertainty, the system of type-2 reduces to the system of type-1, since the left and the right spread of the output uncertainty are lost in the final defuzzification process [37]. More details will be shown in our future work.

4 Type-2 Fuzzy Relational System with Linguistic Antecedent Certainty Factors

We can represent a type-2 fuzzy logic system as a relational system. Moreover we can equipp such a system with fuzzy antecedent certainty factors. Therefore the rules are defined in the following way

$$R^{k,1}: \text{IF } x \text{ is exactly } \tilde{A}^k \text{ THEN } y \text{ is } \tilde{B}^{k,1},$$
$$R^{k,2}: \text{IF } x \text{ is more or less } \tilde{A}^k \text{ THEN } y \text{ is } \tilde{B}^{k,2},$$
$$R^{k,3}: \text{IF } x \text{ is roughly } \tilde{A}^k \text{ THEN } y \text{ is } \tilde{B}^{k,3}.$$

The fuzzy degree of compatiblity between input value $\bar{x}$ and the type-2 antecedent $\tilde{A}^k$ is here transformed by several triangular fuzzy sets $C_{k,m}$. With the use of the extension principle we obtain the following formula

$$\mu_{C_{k,m}}\left(\tilde{\mu}_{A^k}\left(\bar{x}\right)\right) = \int_{v\in[0,1]} g_{C_{k,m}}\left(v\right)/v = \int_{u\in[0,1]} \sup f_{\bar{x}}\left(u\right)/\mu_{C_{k,m}}\left(u\right).$$

The process of the component activation of the rule is shown in Fig. 3. The type-1 fuzzy output of the system may be calculated by any of the type reduction methods *TRM* (see [7], [10]).

$$\tilde{B}' = \underset{m=1,\ldots,M}{TRM}\left(\underset{k=1,\ldots,K}{\tilde{S}} \mu_{C_{k,m}}\left(\tilde{\mu}_{A^k}\left(\bar{x}\right)\right), \tilde{B}^m\left(y\right)\right)$$

Such a type-1 fuzzy output may be finally defuzzified by the classical COA method.

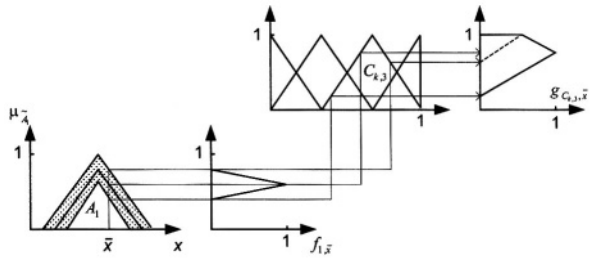


Fig. 3. Processing in type-2 relational system with fuzzy certainty degrees

5 Numerical Simulations

This section presents application of the new relational system to approximate two-input single-output nonlinear function

$$y = \left(1 + x_1^{-2} + x_2^{-1,5}\right)^2, \quad 1 \leqslant x_1, x_2 \leqslant 5. \tag{7}$$

Table 2. Root mean square error for nonlinear function

Singleton model	Relational model	Relational model with fuzzy certainty degrees
0.084	0.044	0.198

We learned and tested systems on original 50-element data set taken from [34]. All parameters were tuned by the backpropagation algorithm. The singleton model had 6 rules and algebraic Cartesian product. The relational system had both 6 input fuzzy sets and output fuzzy sets, related to each other by matrix. The relational system with fuzzy certainty degrees had similar structure, however crisp values were replaced by Gaussian fuzzy sets with fixed widths. Only their centers were tuned. Root mean square error (RMSE) error after 100 000 iterations for each type of model is given in Table 2. In this problem the relational system outperforms other systems, and relational system with fuzzy certainty degrees performs worst.

6 Conclusions

We proposed a new relational type-2 neuro-fuzzy system. The system allows learning all of its parameters (relation matrix elements and membership function parameters) by the backpropagation algorithm. Extending the system to relational type-2 allows for greater flexibility and approaching towards natural

language processing. In future research it would be interesting to compare relational type-2 fuzzy systems with probabilistic neural networks [3] [16] [17] [28] in problems of classifications, modeling and prediction.

References

1. Babuska, R.: Fuzzy Modeling For Control. Kluwer Academic Publishers, Boston (1998)
2. Cpałka, K., Rutkowski L.: Soft Neuro - Fuzzy Systems. Proceedings of the Fifth Int'l Conf. Neural Networks and Soft Computing, Zakopane (2000) 296-301
3. Gałkowski, T., Rutkowski, L.: Nonparametric fitting of multivariable functions. IEEE Transactions on Automatic Control, Vol. AC-31 (1986) 785-787
4. Gaweda, A.E., Zurada, J.M.: Data-driven Linguistic Modeling Using Relational Fuzzy Rules. IEEE Trans. on Fuzzy Systems, vol. 11, No. 1 121-134 (2003)
5. Gaweda, A.E., Zurada, J.M., Aronhime, P.B.: Efficient Data-Driven Modeling With Fuzzy Relational Rule Network, Proceedings of the 11th IEEE International Conference of Fuzzy Systems FUZZ-IEEE'02, Honolulu, HI, May 12-17 (2002)
6. Gaweda, A.E., Zurada, J.M., Setiono R.: Input Selection in Data-Driven Fuzzy Modeling, Proceedings of the 10th IEEE International Conference of Fuzzy Systems FUZZ-IEEE'01, Melbourne Australia, December 2-5, (2001)
7. Karnik, N. N., Mendel, J. M.: An Introduction to Type-2 Fuzzy Logic Systems, Univ. Southern California Rep., October (1998)
8. Karnik, N. N., Mendel, J. M.: Operations on type-2 fuzzy sets, Fuzzy Sets and Systems **122** (2000) 327–348
9. Karnik, N. N., Mendel, J. M., Liang, Q.: Type-2 Fuzzy Logic Systems, IEEE Trans. on Fuzzy Systems 7 (6) (1999) 643–658
10. Mendel, J. M.: Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions, Prentice Hall PTR, Upper Saddle River, NJ 2001
11. Nowicki R. and Rutkowski L.: Soft techniques for bayesian classification, in: Rutkowski L., Kacprzyk J. (Eds.), Neural Networks and Soft Computing, Physica-Verlag, A Springer-Verlag Company, Heidelberg , New York (2003) 537-544
12. Pedrycz, W.: Fuzzy Control and Fuzzy Systems. Research Studies Press, London (1989)
13. Ripley B.D.: Pattern Recognition and Neural Networks. Cambridge University Press (1996)
14. Rutkowski, L., Cpałka, K.: A general approach to neuro - fuzzy systems. Proceedings of the 10th IEEE Int'l Conf. on Fuzzy Systems, Melbourne (2001)
15. Rutkowski L. and Cpałka K.: A neuro-fuzzy controller with a compromise fuzzy reasoning. Control and Cybernetics, Vol. 31, No. 2 (2002) 297-308
16. Rutkowski, L.: Multiple Fourier series procedures for extraction of nonlinear regressions from noisy data. IEEE Transactions on Signal Processing, Vol. 41, No.10 (1993) 3062-3065
17. Rutkowski,L., Rafajłowicz, E.: On global rate of convergence of some nonparametric identification procedures. IEEE Transaction on Automatic Control, Vol. AC-34, No.10 (1989) 1089-1091
18. Rutkowski, L.: Flexible Neuro-Fuzzy Systems: Structures, Learning and Performance Evaluation. Kluwer Academic Press, 2004.
19. Rutkowski, L., Cpałka, K.: Compromise approach to neuro-fuzzy systems. Proceedings of the 2nd Euro-Int'l Symposium on Computational Intelligence, Vol. 76, Koszyce, (2002) 85-90

20. Rutkowski, L., Cpałka, K.: Flexible neuro-fuzzy systems. IEEE Transactions on Neural Networks, vol. 14 (2003) 554-574
21. Rutkowski, L., Cpałka, K.: Flexible weighted neuro-fuzzy systems. 9th Int'l Conf. on Neural Information Processing (ICONIP'02), Singapore (2002)
22. Rutkowska, D., Rutkowski, L.: Fuzzy Neural Networks. Proceedings of the Second Int'l Conference on Parallel Processing and Applied Mathematics (1997) 507-519
23. Rutkowska, D., Rutkowski, L., Nowicki, R.: Fuzzy neural networks with nonparametric defuzzification. Proceedings of the 9-th Int'l Conference System-Modelling-Control, Zakopane (1989)
24. Rutkowska, D., Nowicki, R., Rutkowski, L.: Singleton and Non-Singleton Fuzzy Systems with Nonparametric Defuzzification. in: Computational Intelligence and Applications, P. S. Szczepaniak (ed.), Springer-Verlag (1999) 292-301
25. Rutkowska, D., Rutkowski, L.: Neural-Fuzzy-Genetic Parallel Computing System as a Tool for Various Applications. Proceedings of the Third Int'l Conf. on Parallel Processing & Applied Mathematics (PPAM'99)(1999) 489-498
26. Rutkowska, D., Nowicki, R., Rutkowski, L.: Neuro-Fuzzy System with Inference Process Based on Zadeh Implication. Proceedings of the Third International Conference on Parallel Processing & Applied Mathematics (PPAM'99)(1999) 597-602
27. Rutkowska, D., Rutkowski, L., Nowicki R.: On processing of noisy data by fuzzy inference neural networks. Proceedings of the IASTED Int'l Conf. Signal and Image Processing (SIP'99)Nassau, Bahamas (1999) 314-318
28. Rutkowski L.: New Soft Computing Techniques for System Modelling. Pattern Classification and Image Processing, Springer-Verlag (2004)
29. Rutkowski, L., Cpałka K.: Flexible Structures of Neuro - Fuzzy Systems. Quo Vadis Computational Intelligence, Studies in Fuzziness and Soft Computing, Vol. 54, Springer (2000) 479 - 484
30. Scherer, R., Rutkowski L.: A Fuzzy Relational System with Linguistic Antecedent Certainty Factors. Advances in Soft Computing. Proceedings of the Sixth Int'l Conf. on Neural Network and Soft Computing, Springer Physica-Verlag (2003) 563-569
31. Scherer, R., Rutkowski, L.: Relational Equations Initializing Neuro-Fuzzy System. 10th Zittau Fuzzy Colloquium, Zittau, Germany (2002)
32. Scherer, R., Rutkowski, L.: Neuro-Fuzzy Relational Systems. 2002 Int'l Conf. on Fuzzy Systems and Knowledge Discovery, Singapore (2002)
33. Setness, M., Babuska, R.: Fuzzy Relational Classifier Trained by Fuzzy Clustering. IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics, Vol. 29, No. 5, October (1999) 619-625
34. Sugeno, M., Yasukawa T.: A Fuzzy-Logic-Based Approach to Qualitative Modeling. IEEE Transactions on Fuzzy Systems, vol. 1, no. 1 (1993) 7-31
35. Rutkowska D., Rutkowski L. and Nowicki R.: Neuro - fuzzy systems with inference based on bounded product. in: Advances in Neural Networks and Applications, Word Scientific and Engineering Society Press (2001) 104-109
36. Starczewski, J., Rutkowski, L.: Connectionist Structures of type 2 Fuzzy Inference Systems, in: R. Wyrzykowski et al. (Eds.) Parallel Processing and Applied Mathematics, 4th Int'l Conf. on Parall. Process. & Appl. Math., Nałęczów, September 2001, Lecture Notes in Computer Science **2328,** Springer (2002) 617–624
37. Starczewski, J., Rutkowski, L.: Neuro-Fuzzy Systems of Type 2, 1st Int'l Conf. on Fuzzy Systems and Knowledge Discovery **2**, Singapore, November (2002) 458–462

38. Starczewski, J., Rutkowski, L.: Interval Type 2 Neuro-Fuzzy Systems Based on Interval Consequents, in: L. Rutkowski and J. Kacprzyk (Eds.) Neural Networks and Soft Computing, 6th Int'l Conf. on Neural Networks & Soft Computing, Zakopane, June 2002, Advances in Computing, Springer (2003) 570–577
39. Zadeh, L. A., The Concept of a Linguistic Variable and its Application to Approximate Reasoning — I, Information Sciences **8** (1975) 199–249

Interval Comparison Based on Dempster-Shafer Theory of Evidence

Pawel Sevastjanow

Institute of Comp. & Information Sci., Czestochowa University of Technology
Dabrowskiego 73, 42- 200 Czestochowa, Poland
sevast@icis.pcz.czest.pl

Abstract. The problem of crisp and fuzzy interval (number) comparison is of perennial interest, because of its direct relevance in practical modeling and optimization of real-world processes under uncertainty. There are many approaches to this problem presented in literature, but in all cases the authors propose the methods which give the result of interval comparison in form of real or Boolean number. On the other hand, it is easy to see that all arithmetic operations on intervals give us intervals. So, it seems quite natural to expect the result of interval comparison as interval as well. Indeed, when comparing intervals, we factually order the sets, and it should be preferable to get the result as the some type of set (interval). To do this, we propose the approach, which can derive us the results of comparison as the probability interval. For this purpose, we use the Dempster-Shafer theory of evidence with its probabilistic interpretation.

1 Introduction

Theoretically, crisp intervals and fuzzy numbers can only be partially ordered and hence cannot be compared. However, when interval or fuzzy numbers are used in practical applications or when a choosing has to be made among alternatives, the comparison is needed.

There exist numerous definitions of the ordering relation over fuzzy quantities (as well as crisp intervals). In most cases the authors use some quantitative indices. The values of the such indices present degree to which one interval (fuzzy or crisp) is greater/less than another interval. In some cases, even several indices are used simultaneously. Although some of these methods have shown more consistency and better performance in difficult cases, no single method of fuzzy interval comparison may be putted forward as the best one.

The existing approaches to fuzzy interval comparison may be clustered into three groups: methods of only qualitative fuzzy and crisp intervals ordering, methods, permitting quantitative ordering by means of some indices obtained from the base definitions of fuzzy sets theory and methods based on representation of fuzzy number as α-levels.

It must be emphasized, that last group of methods have some advantages.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 668–675, 2004.
© Springer-Verlag Berlin Heidelberg 2004

At first, they permit to operate with all types of membership functions without any restrictions. This feature is of large practical important, but can be actually used in the case of numerical computation. Secondly, α-levels, are in essence, the set of usual crisp intervals. Hence, the powerful tools of interval arithmetic can be used in addition to solve the problem of fuzzy intervals ordering.

In this paper we present the further development of such methods. Proposed approach is based on α-level representation of fuzzy intervals and probability or belief interval (on the base of Dempster-Shafer theory of evidence, DST) estimation of that certain interval is greater/equal than another interval.

The idea to use the probability interpretation of interval is not in principle novel. But now we can cite only few works which are based on it [1-12].

The attraction of such approach is based on possibility to obtain for compared intervals A and B the completed set of probabilities $P(A < B)$, $P(A > B)$ and $P(A = B)$ with only one assumption that intervals are the supports of uniform distributions of random values $a \in A, b \in B$. Nevertheless, different expressions for probabilities estimation were obtained in the works [1-12]. We suspect this is a consequence of the limited possibility for purely probability approach to deal with the such objects as intervals or fuzzy intervals. The problem is that the probability theory allows us to represent only uncertainty, but interval and fuzzy interval mathematics in addition are inherently characterized by imprecision and ambiguity.

To solve this problem, we propose to use the Depmster-Shafer theory of evidence.

The rest of the article in organized as follows. In Section 2, we brief recall some basic definitions of Dempster-Shafer theory of evidence (DST) needed in our consideration. Section 3 is devoted to presentation of main results of using the DST for comparison of crisp intervals. The extension of proposed approach to the case of fuzzy interval comparison is presented in Section 4.

2 The Basic Ideas of Dempster-Shafer Theory of Evidence

The origins of Dempster-Shafer theory (DST) go back to the work by A. P. Dempster [13,14] who developed a system of upper and lower probabilities. Following this his student G. Shafer [15], in his 1976 book "A Mathematical Theory of Evidence" added to Dempster's work, including a more thorough explanation of belief functions. In [16], the authors provide a collection of articles by some of the leading researchers in this field. The close connection between DS structure and random sets is discussed in [17]. In the following, we provide brief introduction to the basic ideas of this theory.

Assume V is a variable whose domain is the set X. It is important to note that variable V may be treated also as a question or proposition and X as a set of propositions or mutually exclusive hypotheses or answers [18,19].

A DS belief structure has associated with it a mapping m, called basic assignment function, from subsets of X into the unit interval $m : 2^X \to [0,1]$ such that $m(\emptyset) = 0$, $\sum_{A \subset X} m(A) = 1$.

The subsets of X for which the mapping does not assume a zero value are called the focal elements . We shall denote these as A_i, for $i = 1$ to n. We note that the null set is never a focal element. In [15] Shafer introduced a number of measures associated with this structure.

The measure of belief is a mapping $Bel : 2^X \to [0,1]$ such that for any subset B of X

$$Bel(B) = \sum_{i=1}^{n} m(A_i), \quad A_i \subseteq B, \ i = 1 \text{ to } n.$$

With V a variable taking its value in the set X under the semantics provided by Shafer [15], $Bel(B)$ is degree of our belief that the value of V lies in the set B. In [15] it is shown that m can be uniquely recovered from Bel.

A second measure introduced by Shafer [15] is the measure of plausibility.

The measure of plausibility associated with m is a mapping $Pl : 2^X \to [0,1]$ such that for any subset B of X, $Pl(B) = \sum_{i=1}^{n} m(A_i)$, $A_i \cap B \neq \emptyset$, $i = 1$ to n.

The semantics associated with this measure is that $Pl(B)$ is the degree of plausibility that the value of V lies in the set B. It is easy to see that $Bel(B) \leq Pl(B)$.

DS provides an explicit measure of ignorance about an event B and its complementary $\overline{B}$ as the length of the interval $[Bel(B), Pl(B)]$ called belief interval, BI. It can also be interpreted as imprecision on the "true probability" of B [19].

3 Crisp Interval Relations Based on DST

There are only two non trivial cases of comparing intervals placing, which deserve consideration (see Fig.1).

Let $A = [a_1, a_2]$ and $B = [b_1, b_2]$ be independent intervals and $a \in [a_1, a_2]$, $b \in [a_1, a_2]$ be random values distributed on these intervals. As we are dealing with usual crisp intervals, the natural assumption is that the random values a and b are distributed uniformly. There are some subintervals, which play an important role in our analysis. For example (see Fig. 1a), the falling of random $a \in [a_1, a_2]$, $b \in [a_1, a_2]$ into subintervals $[a_1, b_1], [b_1, a_2], [a_2, b_2]$ may be treated as a set of independent random events.

3.1 Case of Overlapping Intervals (Fig. 1a)

Only four mutually exclusive evens H_i, $i = 1$ to 4 may take place in considered situation:

$$\begin{array}{ll} H_1 : a \in [a_1, b_1] \& b \in [a_2, b_2], & H_2 : a \in [a_1, b_1] \& b \in [b_1, a_2], \\ H_3 : a \in [b_1, a_2] \& b \in [b_1, a_2], & H_4 : a \in [b_1, a_2] \& b \in [a_2, b_2]. \end{array} \quad (1)$$

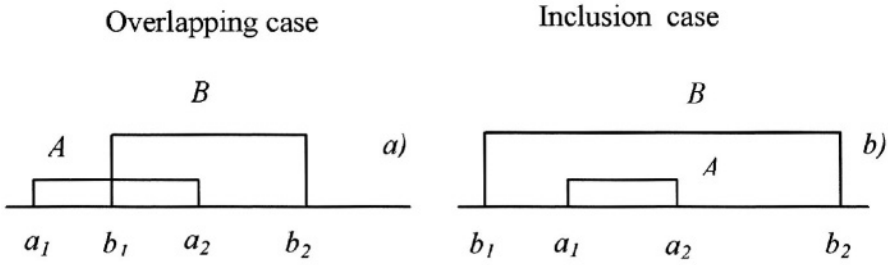


Fig. 1. The examples of interval relations

For the probabilities of events H_1–H_4 from the simple geometric reasons we obtain

$$P(H_1) = \frac{b_1 - a_1}{a_2 - a_1}\frac{b_2 - a_2}{b_2 - b_1}, \qquad P(H_2) = \frac{b_1 - a_1}{a_2 - a_1}\frac{a_2 - b_1}{b_2 - b_1},$$
$$P(H_3) = \frac{a_2 - b_1}{a_2 - a_1}\frac{a_2 - b_1}{b_2 - b_1}, \qquad P(H_4) = \frac{a_2 - b_1}{a_2 - a_1}\frac{b_2 - a_2}{b_2 - b_1}. \tag{2}$$

It can easily be proved that

$$P(H_1) + P(H_2) + P(H_3) + P(H_4) = 1. \tag{3}$$

Thus, in the sense of *DST* the probabilities $P(H_i)$, $i = 1$ to 4, can be used to construct a basic assignment function, *bpa.* Since in case of overlapping intervals $a_1 < b_1$ and $a_2 < b_2$, there are only two interval relations which make sense: $A < B$, $A = B$.

It is easy to see that events H_1, H_2 and H_4 may be considered as the "strong" evidences of $A < B$, otherwise H_3 can be treated as only the "weak" evidence of $A < B$ because it simultaneously is the witness of $A = B$. In DST's notation we obtain:

$$m(\{A < B\}) = P(H_1) + P(H_2) + P(H_4), \tag{4}$$

$$m(\{A < B, A = B\}) = P(H_3). \tag{5}$$

Then using Eq.(3) with (4) and (5) we get

$$Bel(A < B) = m\{A < B\} = 1 - P(H_3) = 1 - \frac{(a_2 - b_1)^2}{(a_2 - a_1)(b_2 - b_1)}, \tag{6}$$

$$Pl(A < B) = m(\{A < B\}) + m(\{A < B, A = B\}) = 1. \tag{7}$$

In the similar way, the pair of DST's estimations for $A = B$ has been inferred:

$$Bel(A = B) = 0, \quad Pl(A = B) = P(H_3) = \frac{(a_2 - b_1)^2}{(a_2 - a_1)(b_2 - b_1)}. \tag{8}$$

Observe that $Bel(A < B)$ in addition may be treated as the probability$P(A < B)$ and $Pl(A = B)$ as the $P(A = B)$. Then belief intervals, *BI,* may be performed as

$$BI(A < B) = [Bel(A < B), Pl(A < B)] = [P(A < B), 1]. \tag{9}$$

So, using DST's approach we obtain interval estimations for the degree of interval inequality and equality.

An important property of $BI(A < B)$ and $BI(A = B)$ for overlapping A and B is $BI(A = B) < BI(A < B)$.

It is worth noting that last inequality is not always a " strong" one, since we have $BI(A = B) \cap BI(A < B) = \emptyset$ only when $P(A - B) > 0.5$.

In other words, this means that in the overlapping case an equality $A = B$ is the only "weak" relation, whereas $A < B$ *is* the "strong" relation.

In the case of $a_1 = b_1$, $a_2 = b_2$, i.e. $A \equiv B$ from Eq.(6)–Eq.(10) we get $BI(A < B) = BI(A = B) = [0,1]$. We introduce the degree of imprecision or ambiguity, *ID,* of interval relations, *rel,* as a whole:

$$ID(rel) = BI(A < B) + BI(A = B) = \\ = \left[1 - \frac{(a_2 - b_1)^2}{(a_2 - a_1)(b_2 - b_1)}, 1 + \frac{(a_2 - b_1)^2}{(a_2 - a_1)(b_2 - b_1)}\right].$$

It easy to see that, as it should be, the width of $ID(rel)$, which may be considered as natural real number type estimation of imprecision, decreases with lowering area of overlapping $a_2 - b_1$.

Obviously, introduced interval form of estimations is a real embodiment of usually implicitly expressed but pivotal inherent demand of interval arithmetic: the result of interval operation must be an interval, too.

3.2 Inclusion Case (Fig. 1b)

In this case we have three possible evens:

$$H_1 : a \in [a_1, a_2] \& b \in [b_1, a_1], H_2 : a \in [a_1, a_2] \& b \in [a_1, a_2], \\ H_3 : a \in [a_1, a_2] \& b \in [a_2, b_2].$$

Since $b_1 \leq a_1$,in this case the relation $A > B$ may becomes true. For instance, there no doubts that $A > B$ if $b_1 < a_1$ and $b_2 = a_2$.

We can observed the elementary evidences of events $A < B$, $A = B$, $A > B$ in the situation observed and we can take them into account to construct the *Bel* and *Pl* functions using nearly the same reasoning as in the case of overlapping intervals.

Finally, we get

$$BI(A < B) = [Bel(A < B), Pl(A < B)] = \left[\frac{b_2 - a_2}{b_2 - b_1}, \frac{b_2 - a_1}{b_2 - b_1}\right], \tag{10}$$

$$BI(A = B) = [Bel(A = B), Pl(A = B)] = \left[0, \frac{a_2 - a_1}{b_2 - b_1}\right], \tag{11}$$

$$BI(A > B) = [Bel(A > B), Pl(A > B)] = \left[\frac{a_1 - b_1}{b_2 - b_1}, \frac{a_2 - b_1}{b_2 - b_1}\right]. \tag{12}$$

It is easy to prove, that in inclusion case we have $BI(A = B) < BI(A > B)$, $BI(A > B)$ at least in the "weak" sense, since $BI(A = B) \cap BI(A > B) \neq \emptyset$ and/or $BI(A = B) \cap BI(A < B) \neq \emptyset$.

As in the case of overlapping intervals we introduce the overall degree of imprecision or ambiguity of interval relations rel as

$$ID(rel) = BI(A < B) + BI(A = B) + BI(A > B) = \\ = \left[1 - \frac{a_2 - a_1}{b_2 - b_1}, 1 + 2\frac{a_2 - a_1}{b_2 - b_1}\right].$$

In contrast to the overlapping case we have an asymmetrical in relation to 1 interval for degree of imprecision or ambiguity.

Finely, several real number criteria may be applied in order to make a reasonable final choice when comparing intervals. Non-exhaustively, we can distinguish:

- strong preference: $B > A$ if $Bel(B > A) > Pl(A < B)$,
- weak preference: $B > A$ if $Bel(B > A) > Bel(B < A)$,
- mixed preference: $B > A$ if $MP(B > A) > MP(B < A)$,

where $MP(\cdot) = aBel(\cdot) + (1 - a)Pl(\cdot)$ with $0 \leq a \leq 1$ (the value a reflects the risk adversity of the decision maker).

Obviously, the mixed preference is the more flexible criterion.

4 Fuzzy Interval Relations Based on DST

Let $\tilde{A}$ and $\tilde{B}$ be fuzzy intervals (numbers) on X with corresponding membership functions $\mu_A(x)$, $\mu_B(x) : X \to [0, 1]$. We can represent $\tilde{A}$ and $\tilde{B}$ by the sets of α-cuts $\tilde{A} = \bigcup_\alpha A_\alpha$, $\tilde{B} = \bigcup_\alpha B_\alpha$, where $A_\alpha = \{x \in X : \mu_A(x) \geq \alpha\}$, $B_\alpha = \{x \in X : \mu_B(x) \geq \alpha\}$ are the crisp intervals.

Then all fuzzy interval relations $\tilde{A}$ rel $\tilde{B}$, $rel = \{<, =, >\}$ may be represented by the set of α-cut relations

$$\tilde{A} \; rel \; \tilde{B} = \bigcup_\alpha A_\alpha \, rel \, B_\alpha \tag{13}$$

Since in the framework of DST all crisp interval relations A_α rel B_α give as the result some crisp intervals $BI(A_\alpha \; rel \; B_\alpha)$, we state that in the right hand side of Eq. (13) we have fuzzy interval. More strictly,

$$\tilde{A}\ rel\ \tilde{B} = \bigcup_{\alpha} A_\alpha\, rel\, B_\alpha = \bigcup_{\alpha} BI(A_\alpha\, rel\, B_\alpha).$$

where $BI(A_\alpha rel\ B_\alpha) = [Bel(A_\alpha rel\ B_\alpha),\ Pl(A_\alpha\ rel\ B_\alpha)]$ are the belief intervals corresponding to the crisp interval relations $A_\alpha\ rel\ B_\alpha$ on the initial α-cuts.

Using the mathematical tools presented in Section 3 we can calculate all values $Bel(A_\alpha\ rel\ B_\alpha)$, $Pl(A_\alpha\ rel\ B_\alpha)$ needed to determine fuzzy interval representing the result of fuzzy interval relation. For practical purposes, it is useful to introduce some real number indices. The simple but perhaps most useful one is the next:

$$BI_R(\tilde{A}\, rel\, \tilde{B}) = \frac{\sum_{\alpha} \alpha BI(A_\alpha\, rel\, B_\alpha)}{\sum_{\alpha} \alpha}.$$

5 Conclusion

Using the Demster-Shafer theory of evidence (DST) make it possible to obtain the closet set of relation (<, =, >) between crisp intervals. Relations between real value and crisp interval are the particle cases of crisp interval relation. All the results are obtained as the probability intervals, but some convenient real value final indices based on interval performance of comparison are proposed, too. The fuzzy extension of crisp interval relations is provided using α-cut representation of compared fuzzy intervals (numbers). In this way, the fuzzy interval as result of fuzzy interval comparison is obtained. It performed by corresponding set of α-cuts, each of which is a belief interval in a spirit of DST. Interval and real value indices based on defuzzification of fuzzy interval comparison results are proposed.

References

1. D. Wadman, M. Schneider, and E Schnaider: On the use of interval mathematics in fuzzy expert system. International Journal of intelligent Systems, **9** (1994) 241-259.
2. R.R. Yager, M Detyniecki, B. Bouchon–Meunier: A context-dependent method for ordering fuzzy numbers using probabilities. Information Sciences, **138** (2001) 237-255.
3. S. Kundu: Min-transitivity of fuzzy leftness relationship and its application to decision making. Fuzzzy Sets and Systems, **86** (1997) 357-367.
4. R. Krishnapuram, J.M. Keller and Y. Ma: Quantitative analysis of properties and spatial relations of fuzzy image regions. IEEE Trans. Fuzzy Systems, **1** (1993) 222-233
5. K. Nakamura: Preference relations on set of fuzzy utilities as a basis for decision making. Fuzzy Sets and Systems, **20** (1986) 147-162.
6. A. Sengupta, T. K. Pal: On comparing interval numbers. European Journal of Operational Research, **127** (2000) 28-43.
7. S. Kundu: Preferance relation on fuzzy utilities based on fuzzy leftness relation on interval. Fuzzy Sets and Systems, **97** (1998) 183-191.

8. P. Sevastjanov, A. Venberg: Modeling and simulation of power units work under interval uncertainty. Energy, **3** (1998) 66-70 (in Russian).
9. P. Sevastjanov, A. Venberg: Optimization of technical and econmic parameters of power units work under fuzzy uncertainty. Energy, **1** (2000) 73-81 (in Russian) .
10. P. V. Sevastjanov, P. Rog: A probabilistic approach to fuzzy and interval ordering. Task Quarterly, Special Issue "Artificial and Computational Intelligence", **7** (2003) 147-156.
11. P. Sevastianov, P. Rog, K. Karczewski: A Probabilistic Method for Ordering Group of Intervals. Computer Science, Czestochowa University of Technology, **2** (2002) 45-53.
12. P. Sewastianow, P. Rog, A. Venberg: The Constructive Numerical Method of Interval Comperison. LNCS, **2328** (2001) 756-761.
13. A.P. Dempster: Upper and lower probabilities induced by a muilti-valued mapping. Ann. Math. Stat., **38** (1967) 325-339.
14. A.P. Dempster: A generalization of Bayesian inference (with discussion). J. Roy. Stat. Soc., Series B, **30** (1968) 208-247.
15. G. Shafer: A mathematical theory of evidence. Princeton University Press, 1976.
16. R.R. Yager, J. Kacprzyk, M. Fedrizzi: Advances in Dempster-Shafer Theory of Evidence. Wiley, New York, 1994.
17. I.R. Goodman., H.T. Nguyen: Uncertainty Models for Knowledge-Based System. North-Holand, Amsterdam, 1985.
18. P. Vasseur, C. Pegard, E. Mouaddib, L. Delahoche: Perceptual organization approach based on Dempster-Shafer theory. Pattern Recognition, **32** (1999) 1449-1462.
19. B. Bloch: Some aspects of Dempster-Shafer evidence theory for classification of multi-modality images taking partial volume effect into account. Pattern Recognition Letters, **17** (1996) 905-919.
20. M. Beynon: DS/AHP method: A mathematical analysis, including an understanding of uncertainty. European Journal of Operational Research, **140** (2002) 148-164.

Distributed Spatial Data Warehouse

Marcin Gorawski and Rafal Malczok

Silesian University of Technology, Institute of Computer Science,
Akademicka 16 street, 44-101 Gliwice, Poland
{Marcin.Gorawski, Rafal.Malczok}@polsl.pl

Abstract. Data warehouses are used to store large amounts of data. A data model makes possible separating data categories and establishing relations between them. In this paper we introduce for the first time the new concept of distributed spatial data warehouse based on the multi-dimensional data model called cascaded star schema [1]. We decided to use the idea of new aggregation tree, that indexes our model in order to fully exploit capabilities of the cascaded star. After close discussion on the cascaded star schema and aggregation tree, we introduce the new idea of distributing data warehouse based on the cascaded star schema. Using Java we implemented both system running on a single computer as well as distributed system. Then we carried out the tests which results allow us to compare the performance of both systems. The tests results show that by distribution one may improve the performance of spatial data warehouse.

1 Introduction

The idea of integrating data into a high quality data repository is the basis of every data warehouse (DW) project. Usually, in DW one creates the hierarchies for individual dimensions. In order to reduce the time required for query evaluation the data are aggregated on various levels of those hierarchies. Spatial dimensions may contain many hierarchies not defined during the design process. In case of supervising automobile traffic it is required that the number of cars in a given area is represented by aggregated spatial data. In [2] authors introduce an idea of aR-Tree that is used for storing aggregates on various levels. They also provide algorithms for updating the contents and materializing the dynamic structure of aR-Tree. Spatial multidimensional data is modeled with a star with spatial dimensions, so-called cascaded star. Data warehouses based on this model are called spatial data warehouses (SDW). In this paper we present for the first time the cascaded star schema used in the architecture of a distributed spatial data warehouse (DSDW). We also introduce the concept of a new aggregation tree which indexes the cascaded star.

[3] introduces the idea of distributing data warehouse based on the star schema. Our project creates and uses aggregation trees, so we had to change and develop the RAID-3 algorithm based concept of data warehouse stripping

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 676–681, 2004.
© Springer-Verlag Berlin Heidelberg 2004

used in [3]. We also introduce two groups of distributing data algorithms applied in the new concept of distributed data warehouse based on the cascaded star schema.

Other aspects are developed in works [4,5,6,7,8]. In [5] authors present a distributed architecture (CBS start) for multidimensional star-schema based databases, which are clustered with respect to their multiple hierarchical dimensions. The problems of number of clusters that should be used for query evaluation as well as the OLAP query classification are the topics of [6]. Authors develop PowerDB project which seems to be an important step towards a two-step query optimizer. A cluster of databases was used as an architecture for developing new FAS (Freshness-Aware Scheduling) protocol which, according to the results, outperforms synchronous replication [7]. Very important problem, directly connected with the distributed architecture, is the dynamic load balancing. In [4] authors propose a scheduling strategy that simultaneously considers both processors and disks efficiency in a Shared Disk architecture. Developed by the authors, unique way of measuring the best cost performance point for network in PC clusters was introduced in [8].

Below, we present the new concept of distributed data warehouse based on the cascaded star schema.

2 Cascaded Star Schema

A star schema contains a main fact table and a few smaller dimension tables. The fact table contains mainly numerical data, while the dimension tables contain attributes describing every row in the fact table. The cascaded star schema [1] contains a main fact table that interconnects the dimensions. Each individual dimension is a separate star schema itself. In case of cascaded star schema the dimension tables' foreign keys are references to smaller star schemas. Those schemas contain information of given dimensions. Each dimension apart from its attributes also contains different information describing those attributes.

An example of this kind of data is an information about cities located in a specified region modeled with a cascaded star schema. This model consists of a central fact table and three dimensions *(population, education, culture).* Every single dimension is described by three sub-dimensions: attributes, spatial, and time.

3 Aggregation Tree

The optimization of query evaluation in SDW requires the appropriate construction of an aggregation tree. The aggregation tree is an index evaluated separately for every dimension. The conception of the aggregation tree presented in [2] was modified in order to match the requirements of SDW based on {Reg_city} model. An aggregation tree consists of nodes. Every node contains the aggregates of some part of the region. The aggregation tree is created for a specified region in a given time period (Fig. 1).

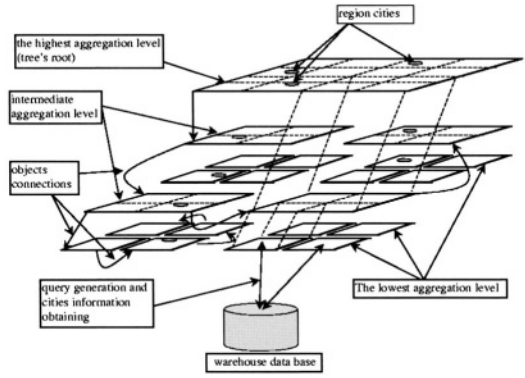


Fig. 1. Aggregation tree's construction details

The algorithm for constructing the aggregation tree is as follows:

- get tree parameters from the user and check them,
- for every smallest aggregation unit a sequence of queries to the SDW base is executed in order to obtain the information about the cities located in the area of a given unit,
- create the next tree levels. This process consists of aggregating the values of a few elements and creating a new element which becomes an upper node.

4 Implementation of Distributed SDW

Using the application written in Java, user can see a map of the region and use an aggregation window. The aggregation window is a part of the region for which user may calculate aggregations values. After selecting the chosen windows user starts the process of evaluating aggregates. The course of the evaluating query process is as follows:

- sorting user-selected windows according to size,
- splitting aggregation windows,
- evaluating aggregation values for every window.

In our example the region map of the model {Reg_city} concerns various numbers of cities in the time period of 101 years.

The next project stage is DSDW based on the {Reg_city} model. The best example of distributing DW based on the star schema is the realization of the RAID algorithm [3]. However, the aggregation trees used in our SDW require a different distribution algorithm.

Algorithms distributing data over the computers can be divided into two groups according to aggregation tree size. The first group contains algorithms which distribute the data in a way that the trees constructed on the servers are of the same size as the tree in a single computer system. This group of algorithms

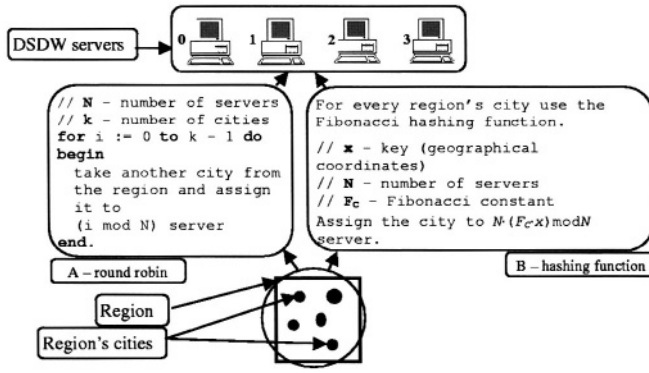


Fig. 2. First group data distribution algorithms.

contains a round-robin algorithm (Fig. 2, frame A) as well as the algorithm based on the hashing function (Fig. 2, frame B).

The second group consists of algorithms which distribute the city information according to the city's geographical coordinates. After generating the data for a specified city the algorithm sends the data to the server which stores the information about the region in which the city is located. Such approach results in smaller trees size in comparison to the trees created on a single computer. This fact has a positive influence on the query evaluation time and resolves the problems with no sufficient memory.

In all cases the data concerning an individual city are collected on one server, which obviates the necessity of communication between the servers.

In the case of SDW running on a single computer, two separate programs are required: an installation program loading data to the warehouse and a program that allows the formulation of queries to the DW. But as far as the DSDW is concerned, we also need to run a server program on the remote computers. In the case of our system the DSDW server services may be divided into three categories: server management (checking, connecting, shutting down), access to the server's data base and access to server's aggregation trees.

In our tests every computer ran a DSDW server, and one computer (number 1) ran the installation and client program. The configuration of computers 1 to 5 was as follows:

- computer 1 (*iceland*) (Athlon 2Ghz, 1024 MB, Oracle 9i, Java 1.4),
- computer 2 (*wyga*) (Pentium II 350MHz, 196 MB, Oracle 9i, Java 1.4),
- computer 3 - 5 (Pentium IV 1.7GHz, 256 MB, Oracle 9i, Java 1.4).

Untasked computers were connected with local network (transmission speed: 100 Mb/s).

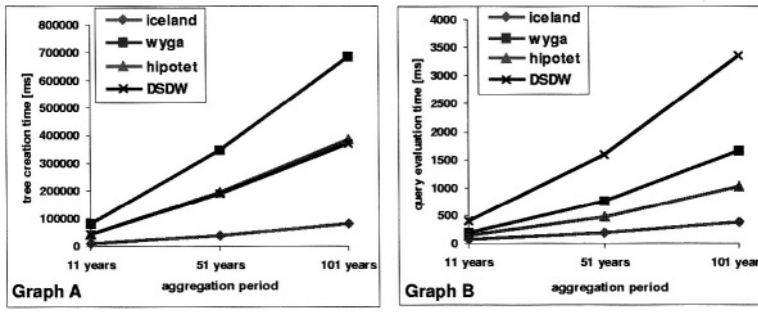


Fig. 3. Graphical representation of the tests results

5 Performance Analysis

The cascaded star schema was filled with data generated by the installation program. Aggregation windows (queries to DSDW) were generated by a specially prepared program. The windows may be divided into two categories. The first one contains windows of random dimensions. The second category contains windows of small size which results in a reduced probability of window overlap. For distributing data over servers we used the round-robin algorithm.

In order to model real DSDW work conditions when many times for report generation complex data aggregation operations are required, in the tests we applied three kinds of aggregation operation. The first kind (level 1) was the simple adding of two values. The second one (level 2) was adding 95% of each sum element. The last kind (level 3) was evaluating aggregation values according to following formula:

$$((sin(\mid a \mid \cdot \pi) \div e) + (cos(\mid b \mid \cdot e) \div \pi)) \cdot currentTimeMillis$$

where a and b are aggregated values, e and π are mathematical constants, and *currentTimeMilis* is the number of seconds since 1.01.1970 up to the moment of evaluating the aggregates.

Data in DW concern regions of various sizes and of various numbers of cities. The tests were conducted on three difficulty levels - 300, 200, and 100 random and small windows for three time periods - 11, 51 and 101 years.
The test results show that the influence of an aggregation difficulty level on the aggregation time is low. This fact is caused by the most frequent and the most long-lasting operation, which is the hard drive access. Time spent on calculating aggregates and constructing all tree levels is not more than 2% of the total time needed for constructing the tree. However, the aggregation difficulty level has significant influence on query evaluation time.

The tests carried out on a single computer system were identical to those on DSDW system based on the {Reg_city} schema. We performed the tests on two computers - *iceland* and *wyga.* Because those two machines differ significantly on performance, after obtaining tests results we calculated the average values comparable to those of an average model of office computer (we named it *hipotet*).

In Fig. 3, Graph A we can see a graph ilustrating relation between tree creation time and aggregation period for various computer systems. Tree creation times for the DSDW system and *hipotet* are almost identical (DSDW times are a bit shorter). The slowest computer's tree creation times are much longer than the rest.

In Fig. 3, Graph B there is a graph illustrating the dependency between queries evaluation times and aggregation periods for various system configurations. In this case the DSDW system was the slowest the cause being the necessity of communication between computers. Yet, we should note that with tree creation times the difference between the wyga computer and the DSDW system is about 5 minutes, whereas in case of queries evaluation times it is less than 2 seconds.

6 Conclusions

In this paper we presented the spatial data warehouse based on the cascaded star schema. Then we introduced the new concept of the aggregation tree that indexes the cascaded star. By means of those concepts we created distributed as well as single computer SDW. In case of DSDW the test results are not ideal because the computer performance was very different. However, from the results we can see that applying distribution of SDW may bring significant profit without the necessity of computer infrastructure development.

Our future work includes tests of the distributing data algorithm from the second group as well as investigating distributed data warehouse queries optimization methods.

References

1. N. Adam, V. Atluri, Y. Yesha, S. Yu: Efficient Storage and Management of Environmental Information. IEEE Symposium on Mass Storage Systems, April 2002
2. D. Papadias, P. Kalnis, J. Zhang, Y. Tao: Efficient OLAP Operations in Spatial Data Warehouses. Spinger Verlag, LNCS 2001
3. J. Bernardino, H. Madera: Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing. Conference on Advanced Information Systems Engineering, Stockholm, Sweden June 5th-6th 2000
4. H. Märtens, E. Rahm, T. Stöhr: Dynamic Query Scheduling in Parallel Data Warehouses. Euro-Par 2002 Paderborn, Germany
5. D. Theodoratos, A. Tsois: Heuristic optimization of OLAP queries in multidimensionally hierarchically clustered databases. DOLAP 2001
6. F. Akal, K. Böhm, H.J. Schek: OLAP Query Evaluation in a Database Cluster: a Performance Study on Intra-Query Parallelism. ADBIS 2002
7. U. Rohm, K. Bohm, H.J. Schek, H. Schuldt: FAS- a Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Components. VLDB 2002
8. C. Kurmann, F. Rauch, T.M. Strieker: Cost/Performance Tradeoffs in Network Interconnects for Clusters of Commodity PCs. ETH Zürich, 2003

Improving Load Balance and Fault Tolerance for PC Cluster-Based Parallel Information Retrieval*

Jaeho Kang[1], Hyunju Ahn[2], Sung-Won Jung[2], Kwang Ryel Ryu[2], Hyuk-Chul Kwon[2], and Sang-Hwa Chung[2]

[1] Center for Intelligent and Integrated Port Management Systems, Dong-A University, 840, Hadan-Dong, Saha-Ku, Busan, Korea
jhkang@pusan.ac.kr

[2] Division of Electrical and Computer Engineering, Pusan National University, San 30, Jangjeon-Dong, Kumjeong-Ku, Busan, Korea
{gus,swjung,krryu,hckwon,shchung}@pusan.ac.kr

Abstract. Information service providers and companies have typically been using expensive mid-range or mainframe computers when they need a high performance information retrieval system for massive data sources such as the Internet. In recent years, companies have begun considering the PC cluster system as an alternative solution because of its cost-effectiveness as well as its high scalability. However, if some of the cluster nodes break down, users may have to wait for a long time or even may not be able to get any result in the worst case. This paper presents a duplicated data declustering method for PC cluster-based parallel information retrieval in order to achieve fault tolerance and to improve load balance in an efficient manner at low cost. The effectiveness of our method has been confirmed by experiments with a corpus of two million newspaper articles on an 8-node PC cluster.

1 Introduction

Modern information retrieval (IR) systems are confronted with the situation of having to process an enormous amount of data and quantities of user requests due to the rapid growth of the Internet. Recently, information service providers or companies have begun considering the PC cluster system as an alternative solution to the mid-range or mainframe computers for building IR systems because of its cost-effectiveness and high scalability. For a commercial IR service to be successful, however, it is very important to minimize the out-of-service time of the system as well as to maximize the system performance. Advanced technologies these days provide us with hardware systems with sufficient reliability. But, the software systems are getting increasingly complex and thus becoming more susceptible to breakdown, which makes fault tolerance still a fatal issue.

* This work was funded by the University Research Program supported by Ministry of Information and Communication in Korea under contract 2002-005-3.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 682–687, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Although there has been a lot of research on fault tolerance in the database area, little research work has been reported on this topic for PC cluster-based parallel IR.

This paper presents a data partitioning and duplicated distribution method to realize a PC cluster-based parallel IR system with fault tolerance and improved load balance. The basic idea is that the data partition in each node is duplicated at the next node for fault tolerance. A positive side effect of this simple strategy is that we are given opportunities to better balance the workloads between the adjacent nodes. The next section describes how data are partitioned and distributed to the nodes of a PC cluster-based parallel IR system. Section 3 presents the main idea of our proposed method of duplicated data distribution for fault tolerance and load balancing. Experimental results with two million newspaper articles on an 8-node PC cluster are reported in section 4. Then, related works are discussed in section 5 followed by conclusions and future research in section 6.

2 Data Declustering for a PC Cluster-Based IR System

Figure 1 shows the flow of overall query processing of an IR system. When a query is input, the IR system parses it and starts the retrieval step in which relevant documents are found and ranked according to their relevancy to the given query. In the retrieval step, the data needed for document ranking are read from the so-called inverted index file (IIF) stored in the disk. The IIF structure enables indexing the data of relevant documents by using the query terms. Given a term t_i which happens to appear in m different documents, the IIF lookup returns term t_i's record $((d_{1i}, w_{1i}), (d_{2i}, w_{2i}), \ldots, (d_{mi}, w_{mi}))$, where (d_{ji}, w_{ji}) indicates that d_{ji} is the id of the j-th document containing t_i and w_{ji} is the weight of t_i in d_{ji}. Once all the records of the query terms are retrieved from the IIF, document scores are calculated based on the term weights using a simple but efficient measure such as the well-known cosine similarity [1].

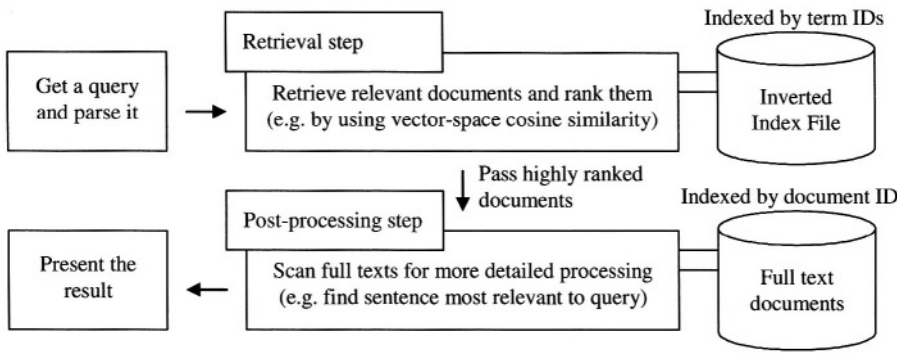


Fig. 1. Flow of overall query processing

The post-processing step does a more sophisticated and detailed computation on those highly ranked documents screened out in the retrieval step. Some examples of the IR operations in this post-processing step are finding for each document the sentence most relevant to the query, summarizing the documents, clustering, and so on. Many commercial IR systems nowadays support some of these post-processing operations. To parallelize all of these IR tasks using a PC cluster, we have to decluster both types of data, i.e., the IIF and the full text documents, to the cluster nodes.

The IIF can be deculstered to the nodes in two different ways, i.e. by term-based or document-based declustering [2]. In term-based declustering, an IIF is built for the whole documents of the corpus and the term records are allocated to the nodes in an interleaving manner. For an 8-node PC cluster for example, term records $\{t_1, t_9, ...\}$ could be allocated to $Node_1$, $\{t_2, t_{10}, ...\}$ to $Node_2$, and so on. In contrast, the document-based distribution first assigns the whole documents to each of the nodes evenly and then an IIF is built at each node for those documents assigned to that node. The document-based declustering does not incur any inter-node communication when calculating a document's score because all the necessary information is self-contained within the node to which the document is assigned. This is not the case for the term-based declustering [3]. Our approach adopts a document-based declustering scheme and thus scales up easily due to the low communication cost.

3 Fault Tolerance and Load Balancing Strategies

While the simplest strategy to provide fault tolerance would be duplicating the whole system, it would be prohibitively expensive. In this paper, we propose a low-cost IIF record duplication method for fault tolerance together with a novel dynamic load balancing strategy exploiting the data duplication. In our method each IIF generated by the document-based declustering is stored in two nodes; the original IIF is stored in the primary node and its copy in the secondary node adjacent to the primary node. For example, if an IIF is stored in $Node_1$ of Fig. 2 as the primary node, $Node_2$ becomes its secondary node. Following this strategy, the system can operate unless two consecutive nodes break down simultaneously. Figure 2 shows a double fault situation (simultaneous breakdown of $Node_2$ and $Node_6$) in an 8-node IR system. The system can still function by using the duplicated IIFs in $Node_3$ and $Node_7$.

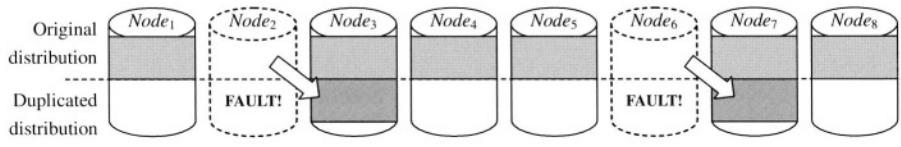
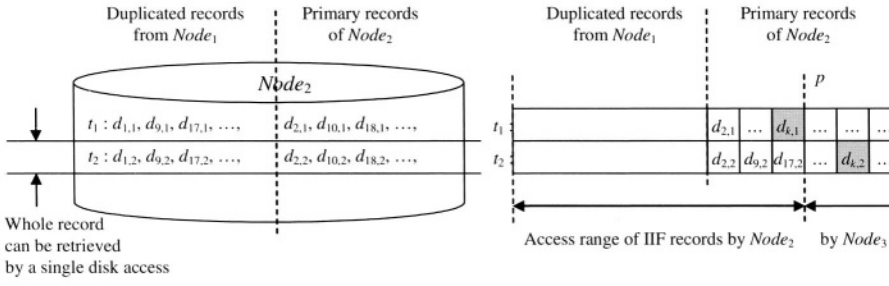


Fig. 2. An example situation of an IR system with a static load balancing strategy

A simple backing-up strategy when a node goes down would be to have its secondary node process all the faulty node's data. This static load balancing strategy is not very satisfactory because the amount of data processed by the secondary node is twice as large as those of the other nodes, as illustrated by $Node_3$ and $Node_7$ in Fig. 2. A better solution would be to distribute the additional load caused by the faulty node to other nodes as evenly as possible. For example, we can make $Node_3$ to process all the data of $Node_2$ and only 1/3 of the data primarily allocated to itself, and $Node_4$ to process 2/3 of $Node_3$'s data and 2/3 of its own data, and $Node_5$ to process 1/3 of $Node_4$'s data and all of its own. This dynamic load balancing strategy can of course be extended to more number of nodes, imposing less extra burden to each of those nodes.

Figure 3(a) shows the disk file structure of a node, namely $Node_2$ in this example, in which the primary and secondary IIFs are merged together into a single file. The two IIF files are not simply concatenated. The concatenation is done at the level of the term records. When $Node_2$ becomes a backup node of $Node_1$ in case of a breakdown, it would need two disk accesses per term for reading relevant document data if the IIFs were merged by a simple file-level concatenation. The record-level concatenation guarantees minimum disk access which is one of the critical delaying factors in the IR service.

Figure 3(b) shows an example situation in which $Node_2$ has to read the records of the terms t_1 and t_2 when $Node_1$ is not operational. Assume that $Node_2$ is supposed to read all of $Node_1$'s data and half of its own according to our dynamic load balancing strategy. Then, $Node_2$ will read half of both t_1's and t_2's record from its primary part of the IIF. However, since the length of the records and the relevant documents of the two terms may not be the same, $Node_2$ may read the weight of t_1 in document k ($d_{k,1}$) but not the weight of t_2 in the same document k ($d_{k,2}$). This implies that the missing information $d_{k,2}$ must be obtained from $Node_3$ through communication for the score of document k to be calculated. Although $d_{k,2}$ can also be read in by extra disk access, it turns out that communication is much cheaper because the amount of data to be transferred is usually very small.



(a) Duplicated IIF record arrangement (b) IIF record access for dynamic load balancing

Fig. 3. Duplicated IIF record arrangement and access strategy

Data duplication and load balancing strategies for the post-processing step are not very different from those used for the retrieval step. Each full document or its equivalent form is stored in both the primary and secondary nodes. Load imbalance can occur not only in faulty situation but also in normal situation in this step because the total number of documents to be processed simultaneously by the cluster nodes is very small compared to the retrieval step. We developed a simple and greedy but efficient heuristic that assigns documents to the nodes by estimating each node's crowdedness and then assigning each document to the primary or secondary node whichever has the lower crowdedness.

Table 1. Average query processing time in normal and some-nodes-faulty situations

Situation	No fault	Single fault	Double fault
SS	448 ms	565 ms	563 ms
SD	430 ms (3.9%)	492 ms (12.9%)	524 ms (7.0%)
DD	430 ms (3.9%)	468 ms (17.2%)	504 ms (10.6%)

4 Experimental Results

We have built an 8-node PC cluster system with the PCs connected via Gigabit Ethernet. The IR software on each node communicates with each other using the MVICH[4] library that is a variant of MPICH specially designed for VIA (Virtual Interface Architecture), a low-cost user-level communication interface. The corpus used for our experiments consists of two million newspaper articles. All the data are declustered and duplicated as described in the previous sections. Experiments were repeated five times with 500 queries each containing 24 terms (assuming relevance feedback), and the query processing times were averaged. For the post-processing step, we implemented a search scheme for finding in each document the sentence most relevant to the given query by calculating both the cosine similarity (between the given query and each sentence) and the relative distance between query terms within the sentence.

The overall system performance was evaluated to see the effect of our proposed strategy under both normal and faulty conditions. We implemented three different load-balancing strategies: SS, SD, and DD. SS only uses static load balancing for both the retrieval and post-processing steps. SD uses static load balancing for the retrieval step and dynamic load balancing for the post-processing step. DD uses dynamic load balancing for both steps.

Table 1 shows the average query processing times resulting from the three strategies. In case of SS, the performance was degraded severely in both single fault and double fault situations due to its excessive load concentration on the nodes next to the faulty ones. In contrast, SD and DD showed smooth and graceful performance degradation mainly due to the dynamic load balancing in the post-processing step. The percentages in parentheses are performance gains over the simplest method SS. The performance improvement of DD over SD

are realized by duplicated IIF declustering and dynamic load balancing in the retrieval step.

5 Related Works

The performance of an IR system can be improved by an efficient parallelization of the IR operations with the data appropriately partitioned and allocated to each node's disk. Random data partitioning on a Connection Machine was among the first research work in this direction [5]. Later, an IR system having one computer and many disks was simulated and the performances of term-based and document-based IIF partitioning were compared [2,6]. More recently, a term-based IIF partitioning and distribution method was proposed for PC cluster-based IR systems [3]. However, the focus of the work was on the efficiency improvement of the retrieval step. The issues on post-processing step a fault tolerance were not covered.

6 Conclusions and Future Research

We proposed a duplicated data distribution and dynamic load balancing method to provide fault tolerance and to improve load balancing for a PC cluster-based parallel IR system. For both the retrieval and post-processing steps of IR, each data partition is stored in two consecutive nodes for fault-tolerance as well as for better balancing of work loads. The proposed methods are effective in both normal and faulty operating conditions. A future direction of the work would be to extend our load balancing strategies to cover the situations where the processing nodes have different computing powers.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern information retrieval. Addison-Wesley (1999)
2. Jeong, B., Omiecinski, E: Inverted file partitioning schemes in multiple disk systems. IEEE Transactions on Parallel and Distributed Systems **6(2)** (1995) 142–153
3. Chung, S-H., Kwon, H-C., Ryu, K. R., Jang, H-K., Kim, J-H., Choi, C-A.: Information retrieval on an SCI-based PC cluster. Journal of Supercomputing **19(3)** (2001) 251–265
4. National Energy Research Scientic Computing Center: MVICH - MPI for virtual interface architecture. http://www.nersc.gov/research/ftg/mvich/index.html (1999)
5. Stanfill, C., Thau, R.: Information retrieval on the connection machine : 1 to 8192 gigabytes. Information Processing and Management **27** (1991) 285–310
6. Xi, W., Sornil, O., Luo, M., Fox, E. A.: Hybrid partitioned inverted indices for large-scale digital libraries. In Proceeding of the 6th European Conference on Research and Advanced Technology for Digital Libraries (2002) 422–431

An Efficient Conflict Detection Method for Maintaining Consistency of Mobile Database System*

Sung-Hee Kim[1], Jae-Dong Lee[2], Jae-Hong Kim[2], and Hae-Young Bae[1]

[1] Dept. of Computer Science & Engineering, Inha Univ., 402-751, Korea
laufer@dblab.inha.ac.kr, hybae@inha.ac.kr
[2] Dept. of Computer Science, Dankook Univ., 140-714, Korea
letsdoit@dku.edu
[3] School of Computer Information Engineering, Youngdong Univ., 370-701, Korea
jhong@youngdong.ac.kr

Abstract. In mobile database system, a voluntary or involuntary disconnection has happened frequently. This problem causes inconsistency. This paper has proposed *the conflict detection method* for maintaining consistency. The proposed method uses the histories of transactions consisted of the information of operations and the valid time of transactions processed in off-line. Compared with other methods, the transaction time resembles others and the conflict detection time is decreased about 17%.

1 Introduction

The concern with mobile database system has been growing by development of wireless communication technology. Mobile database system has an advantage that we can access data whenever and wherever. However, mobile computing environment has some restrictions such as limited bandwidth, insufficient resources and instability of wireless communication. Especially, a voluntary disconnection for saving of communication cost and an involuntary disconnection by wireless network error are becoming an issue newly. Therefore *disconnected operation* technique(such as *caching* or *hoarding)* has been studied for the persistent service of mobile host in disconnection. However, this technique has caused inconsistency problem. Various consistency maintenance methods used for existent distributed database system are studied for in-consistency problem, but it is not correct solution.

To solve this problem, an efficient conflict detection method is proposed for maintaining consistency of data in disconnection. The proposed method has stored and managed transaction information using extended multidimensional index. The transaction information is the operation and the valid time of transactions processed in disconnected. When reconnect with server, the transaction information of each mobile host have been sent to the server, it has been reconstructed

* This research was supported by University IT Research Center Project.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 688–693, 2004.
© Springer-Verlag Berlin Heidelberg 2004

at the server and we can detect conflicts efficiently through overlapping operation.
As performance evaluation, the proposed method can give faster response time about 17% than the conventional detection methods using *timestamp* or *version-vector* when hand-off or disconnection states.

2 Related Works

In mobile computing environments, data and information related executing operation should be sent to the mobile host for improving the availability in disconnection state. It is called *caching* or *hoarding* that it receives required data beforehand for disconnection, and it is known *disconnected operation* that it executes using only local data of mobile host.
In this time, data can be replicated in mobile host. Also, the replicated data must keep the consistency for accuracy of database. The data synchronization technology has been studied to maintain consistency of data in disconnected operation that be executed in mobile terminal in disconnected state during long time. So the database server has offered the data for disconnected operation in mobile terminal(*Publish*), mobile terminal has replicated the data which server supplies, and has used(*Subscribe*). If the modification of the data has been occurred at server or mobile terminal, the synchronization is enforced to keep consistency state. In this time, it must be confirmed whether data conflicts are occurred or not, and a conflict resolution is needed when conflicts arise. It is achieved in server that detects conflict and solves it. For the data updated by only the mobile host, it is simple matter to propagate the updates when the mobile host reconnects.
However, if the mobile host caches read-only copies of data updated by other computers, the cached data may become inconsistent. When the mobile host is connected, it can be sent *invalidation reports* that inform it of out-of-date cache entries. However, when the mobile host is disconnected, it may miss an invalidation report. A simple solution to this problem is to invalidate the entire cache on reconnection, but such an extreme solution is highly costly. If updates can occur at both the mobile host and elsewhere, detecting conflicting updates is more difficult.
It is the method using *timestamp* that it allocates timestamp to transaction processed in mobile host or server, and it checks serialization of transaction using this and detects conflict. The *version-vector method* detects inconsistencies when copies of a document are independently updated.

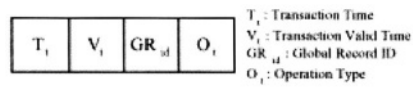


Fig. 1. Data Structure of Time and Operation Information of Transaction

3 Extension of Data Structure for Conflict Detection

3.1 History of Transaction

The proposed method for maintaining consistency in this paper offers synchronization after connection with server using special index structure that is consisted the valid time and the operation information of transaction. Figure 1 is the basic structure of stored information to be used in this synchronization.
Tt is transaction time when transaction processed relevant record cached in mobile terminal, is given automatically by system. **Vt** is valid time in the real world, and is decided by user's modification transaction. It is the last time when update operation is processed by transaction. It has **MAX_TIME** value that indicates present time while relevant record is continuously valid state. **GRid** is global record ID, it is ID value that is uniquely distinguished in server and entire mobile terminal and is different between record ID(RID) in server. **Ot** is operation information of transaction such as update, delete, insertion operation etc. Data structure in Figure 1 is stored by node of extended multi-dimensional index, and is used in collision detection at synchronization.

3.2 Additional Metadata

The following metadata should be added to do synchronization using extended data structure in database of mobile terminal. **LAST_SYNC_TIME** is time when mobile terminal does finally synchronization, and is changed by system. **RANGE_GRid** is assignment range of **GRid** that database server allocates for several mobile terminal, and is used when insertion operation happens in mobile terminal. **CLIENT_ID** is ID value to distinguish each mobile terminal, and is used to differentiate each transaction of mobile terminal at conflict detection.

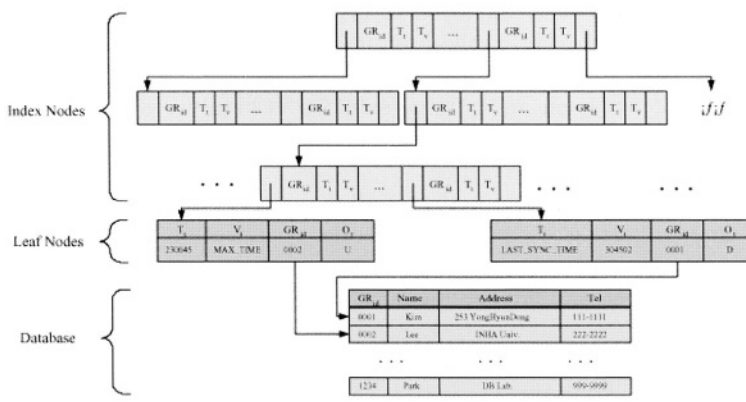


Fig. 2. The Structure of Extended Multidimensional Index

3.3 Extension of Multidimensional Index

If the transactions are achieved, transaction information of Figure 2 is inserted in multi-dimensional index, and it is changed to metadata of 3.2. Such composed index is subsequent used to detect data conflict between mobile terminals at synchronization with server. Followed figure 2 shows structure of extended multi-dimensional index.
The index node uses key that is consisted **GRid** that is global record ID, **Vt** that is valid time of transaction, and **Tt** that is transaction execution time of transaction. **Tt**, **Vt**, **GRid** and **Ot** are saved in leaf node and are composed to index. Index approaches to database where data saved in, using **GRid** of this last node.

4 Conflict Detection Method Using History of Transaction

4.1 Transaction Processing

The mobile host works two things while connected with server. One is that it manages replicated data in cache through invalidation reports sent by the server, the other is that it requests data to server when data for processing mobile transaction is not in MDBS. It inserts in cached data of MDBS when connection with server. After it sends invalidation reports to server for maintaining consistency, it commits mobile transactions. In case of disconnected operation, it inserts into cached data firstly and makes history of transaction. The history is stored and managed Transaction History Manager. In this time, new identifier of inserted record is obtained using following equation (1).

$$(\mathbf{CLIENT_ID} \ll 60) \& FindNextGRid(\mathbf{RANGE_GRid}) \quad (1)$$

First it gets next **GRid** using function *FindNextGRid* among **RANGE_GRid,** and shifts **CLIENT_ID** unit of bit using shift operation. Last we can gets identifier of new record using *Bit AND* operation them.

4.2 Conflict Detection

Before achieving synchronization works, we must confirm whether conflict occur or not, and we must all work to deal with this problem when conflict occurs. Data conflict has three kinds of collisions. The first is *Insert Conflict,* more than two terminals insert equal tuple by different value, the second is *Delete Conflict,* one terminal deletes tuple and another updates same tuple, the last is *Update Conflict,* more than two terminal update by different value to same data field.
The detection of data conflict uses index that is composed in each mobile terminal. Figure 3 shows progress that detects data conflict by disconnected operation that is processed in two terminals. Graph at left-top shows index that is composed at terminal 1, and left-bottom is index that is constructed at terminal 2.

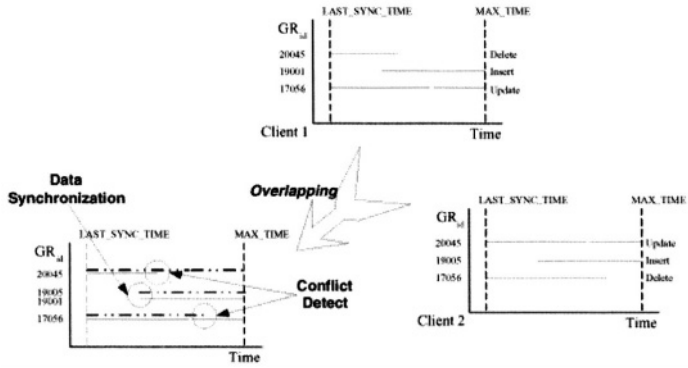


Fig. 3. Conflict Detection using Synchronization with Server

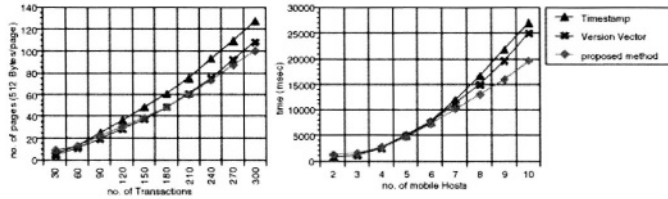


Fig. 4. Transaction Time and Conflict Detection Time

Data conflict can be detected through overlapping operation of indices that are composed in each terminal. As new node is inserted when insert, delete or update operation happens in each terminal, we can detect conflict if we examine index of terminal. If line starts or ends in duplicated part when overlapping two graphs, it is case that conflict occurs. As **GRid** 19001 and 19005 are case that different value inserted in each terminal, we must newly apply using data synchronization.

If data conflict occurs, to detect and resolve conflict we use not only passive method through selection by user, but also apply *trigger* that is rule that client or server is prior, or consistency rule.

5 Performance Evaluation

The performance evaluation of proposed method is compared with the existent method using timestamp and version vector. We used *CSIM* that is a simulation tool, and used *Visual C++* that is development tools.

In disconnection state each mobile host execute transactions. When reconnected with server, we synchronize and detect conflicts. As result of performance evaluation, we compare total time for execution of transactions and time for detecting conflicts.

In Figure 4, we have compared the transaction time and conflict detection time between the proposed method and two conventional methods. The proposed

method is resembled with other methods in transaction time. However, the difference increases from exceeding 60 because of cost for management of index structure.
According as number of mobile host increases, it shows better performance than existing methods. However, in case of few mobile hosts, it shows lower performance because of cost of overlapping operation.

6 Conclusion

This paper has focused on solving the inconsistency problem of the mobile database system. In order to resolution this problem, *the conflict detection method* has been proposed. As shown above sections, the proposed method has managed the history of transaction using the extended multidimensional index in disconnection state. Then the history of transaction has been reconstructed in the server when reconnected, it is used to detect conflicts of disconnected operations.
Finally the performance of the proposed method has been evaluated in experiments which are tested in the transaction time and the conflicts detection time. Comparing with conventional other methods using *timestamp* or *version-vector,* the transaction time has resembled others and the conflicts detection time has been reduced about 17%, therefore the performance of system is improved.

References

1. J.P.Boone, J.Pederson: Extending Enterprise Data and Applications to Mobile Devices using DB2 Everyplace. White paper. IBM. 2001
2. Daniel Barbara: Mobile Computing and Databases – A Survey. IEEE Transactions on Knowledge and Data Engineering. Vol. 11. No. 1. 1999
3. Elmasri, Navathe: Mobile Databases. Fundamentals of Database Systems. Third Edition. 886 891. Addison-Wesley. 2000
4. James J. Kistler, M. Satyanarayanan: Disconnected Operation in the Coda File System. ACM Transactions on Computer Systems. 1992
5. A.Kahol, S.Khurana: A Strategy to Manage Cache consistency in a Disconnected Distributed Environment. IEEE Transaction on Parallel and Distributed Systems. 686 700. 2001
6. Sung-Hee Kim, Hae-Young Bae: Control of Databases in Mobile Distributed Computing Environments. Proceedings of PDTSD'02. 2002
7. YinHuei Loh, Shojiro Nishio: A Hybrid Method for Concurrent Updates on Disconnected Databases in Mobile Computing Environments. Proceedings of the SAC'00. 2000
8. Ravi A. Dirckze, Le Gruenwald: A pre-serialization transaction management technique for mobile multidatabases. Proceedings of MONET'00. 2000
9. Sanjay Kumar Madria: Timestamps to Detect R-W Conflicts in Mobile Computing. Proceedings of International Workshop on Mobile Data Access in conjunction with ER'98. 1998
10. Shirish H. Phatak, B. R. Badrinath: Conflict Resolution and Reconciliation in Disconnected Databases. Proceedings of MDDS'99. 1999

Distributed Knowledge Management Based on Software Agents and Ontology*

Michal Laclavik[1], Zoltan Balogh[1], Ladislav Hluchy[1], Renata Slota[2], Krzysztof Krawczyk[3], and Mariusz Dziewierz[3]

[1] Institute of Informatics, SAS, Dubravska cesta 9, Bratislava 84237, Slovakia
laclavik.ui@savba.sk
[2] Institute of Computer Science, AGH-UST, al. Mickiewicza 30, Cracow, Poland
rena@uci.agh.edu.pl
[3] ACC CYFRONET AGH, Nawojki 11, 30-950 Cracow, Poland
krafcoo@icsr.agh.edu.pl

Abstract. In this paper we present the use of ontology for knowledge representation and handling in Software Agent Systems. Motivation has come from Pellucid IST project where we need to capture and capitalize employee's knowledge in organization. This knowledge is then presented to other employees as they work on particular tasks. The Protégé ontology editor and JADE multi-agent system is used for implementation. Ontology is usually used in intra-agent communication for agents to understand each other; we used ontology also as knowledge data model to store knowledge as instances of ontological terms into object database, thus agents can access and manipulate knowledge data directly and still stay lightweight.

1 Introduction

Motivation for this article has come from Pellucid project. Pellucid (Platform for Organizationally Mobile Public Employees) is European Project IST-2001-34519. The Pellucid System is particularly aimed to capture, record and capitalize the knowledge of current employees about their work in an organization [1].

Pellucid uses the socalled Intelligent Software Agents based on FIPA standards [2],[3] for different activities as capitalizing and data mining of knowledge, personalizing of knowledge, indexing and analyzing organizational document repositories or for integration of existing systems in organization. The ability to use ontology makes Pellucid easy to customize for different problem domains by simply changing domain ontology.

Human knowledge is based not only on facts which are true or false but also on uncertain knowledge which is true or false partially. Several methods can be used to represent such knowledge, e.g. probability measures, fuzzy logic

* This work was supported by EC Project Pellucid 5FP RTD IST-2001-34519 and Slovak Scientific Grant VEGA 2/3132/23

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 694–699, 2004.
© Springer-Verlag Berlin Heidelberg 2004

or computing with words [4]. Some methods are known to represent uncertain knowledge even in agent systems by e.g. extended FIPA-SL language; however, uncertain knowledge is still quite complicated and not understandable especially for the agents themselves. When using uncertain knowledge or knowledge where true and false facts are not strongly defined, computer systems cannot discover new facts in existing knowledge base using logical operators. This is known as a fundamental problem of contradictory knowl-edge in computer systems [5].

This is why knowledge base in Pellucid consists only of strongly true facts. Such facts are structured and defined by ontologies. By evaluation of administration application and pilot sites of the project we discovered that for administration application where business processes are well defined it is reasonable and useful to use knowledge based on facts rather than on uncertain knowledge.

Recent knowledge systems built on ontological or other models are mostly centralized solutions [6]. Pellucid uses hybrid approach, where agents can access certain knowledge directly from centralized organizational memory or they can ask specialized agents to provide them with knowledge handled only by this particular agent; e.g., in Pellucid monitoring agent has knowledge about workflow actors, activities or workflow processes since it represents interface to workflow management system; information and search agent has certain knowledge about documents and can do full text searching or content and context base searching of knowledge in documents or capitalization agents providing other agents and users with knowledge of similarity measures among documents [7], [8]. Agent based approach as created in Pellucid thus combines distributed and centralized approach which seems to be extremely useful for the area of administration application.

In this article we will devote attention mostly to usage of ontology for knowledge management in Pellucid architecture. Therefore, only two types of agents will be partially presented: Part of Personal Assistant Agent, called Query Agent, and part of Role Agent, called Answering Agent.

2 Knowledge Management Architecture

When a user wants to extract some knowledge from the information system, he/she has to create a query. Our goal is also to automatically prepare all possible questions one can ask. Such questions are created from ontology. In addition, the goal is to create such an algorithm which is able to answer questions whether such knowl-edge is presented in information system. Pellucid system has the ability to capture and record knowledge from the user. Simple actions as contacting someone in a particular workflow activity, reading or updating documents are captured. Pellucid Agents have also some data-mining capabilities for capitalization of knowledge. Data-mining algo-rithms and detail description of information capture is out of scope of this article; we believe however, that together with gathering information from information system based on pre-prepared questions they are the basis of the future type of information systems.

Information System for querying of knowledge [11], [12] consists of three main elements:
- Query Agent (QA) - Agent used by user to ask any possible questions based on used ontology. This agent includes user interface as well. There are several such agents in the system, each representing different user.
- Answering Agent (AA) - Agent which is able to answer all possible QUERY-REF ACL messages where FIPA-SL is used as content language.
- Organizational Memory (OM) - where the knowledge is stored.

2.1 Pellucid Modules

In this section we briefly describe generic modules of Pellucid. This description of functionality is needed for better understanding of described modules. Generic version of Pellucid system has three modules: Intelligent Contact Management Module, Intel-ligent Document Management Module and Critical Time Management Module.

List of contacts is presented in each organization in a form of contact database, searchable by keywords with sorting ability, etc. Intelligent Contact Management Module will provide users with intelligent contact list related to the activity, which is performed by a user in contact relation to a particular activity. Each module uses its ontology to define relations between knowledge entities such as activity, document, contact, etc., and uses common techniques to capture, capitalize and return knowledge to a user. We will explain knowledge management functionality on simplified Contact Module [12].

3 Knowledge Manipulation

Agents are manipulating with knowledge by message passing among them. FIPA defined standard called Agent Communication Language (ACL) which is used for agent communication also in our architecture. The main part of ACL message is the content of message. FIPA Semantic Language (FIPA-SL) is used as content language in Pellucid architecture [2], [3], [13]. FIPA-SL queries are also past to querying engine to return knowledge from Organizational Memory. Each agent must understand terms used in content of the message in the same way, thus these terms are defined by on-tologies.

Ontology. Ontology in Pellucid defines the meaning of the terms in used content language and the relation among these terms. The bases of the Pellucid ontology are “Event” and “InformationResource”. Each event is connected with an activity (task) in workflow, with an employee, and can be connected with “InformationResource” (Document, Contact) as well. (See Fig. 1)

Ontology will later include more generic elements, which will help extend it with domain specific ontologies, different for each application area. Pellucid does not support domain specific ontology such as ” Ontology of installation of

traffic lights" which is one pilot site of Pellucid, but we believe that by extending described extracting algorithm Pellucid will be able to query knowledge for different domains, simply by changes in domain ontology.

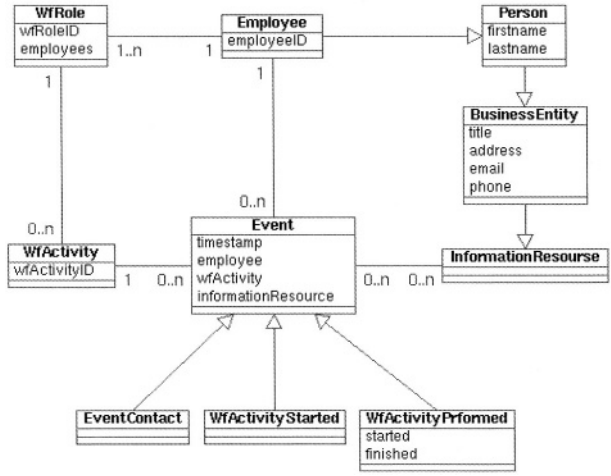


Fig. 1. UML scheme – Bases of Pellucid ontology

We developed a system which is able to work with knowledge represented by any domain specific ontology. However, ontology used in Pellucid is well suited for representing knowledge in organizations, where administration processes or workflow are presented, because it relates user activities with information resources and particular type of work.

3.1 Capturing and Extracting Knowledge

Capturing knowledge is done by user input or by simple storing of user actions such as workflow activity performed, someone contacted by email or reading of a document. Using of Workflow Management System helps Pellucid to interconnect user actions with work activities of an employee. This enables Pellucid to serve knowledge to different employee in the future when employee works on the same task (workflow activity). However, capturing of knowledge is out of the scope of this article, so we mentioned it only for better understanding of complete architecture.

Ontology as represented in JADE [14] needs to define predicates, concepts and terms. Concepts are e.g. contacts, documents, workflow activities or employees. Predicates connect concepts, e.g. EventContact connects Employee, Contact and WfActivity and it means that contact was contacted by employee in particular activity. Thus predicates are such concepts, which we can ask if it is true or false. Concept is object in ontology and term is property of the concept. Event and its child classes are treated as predicates in Pellucid ontology. We can

understand each predicate as a sentence representation in human language, e.g. for WfActivity Performed sentence is “Employees which performed current workflow activity” or for EventContact sentence is “Contacts which were contacted in current workflow activity by particular employee”. As we can see, sentences are based on Event Name and its interconnection with Employee, InformationResource and WfActivity. Our algorithm for creating questions is non-deterministic and decisions which way to go in creating questions is up to the user.

```
pQuestion = SelectPredicate();
cKnown = SelectConcept(GetAllConcepts(pQuestion));
iKnown = SelectInstance(GetAllClasses(cKnown));
cNeeded = SelectConcept(
  GetAllConcepts(pQuestion) - cKnown
);
return createQuestion(
  pQuestion, cKnown, iKnown, cNeeded
);
```

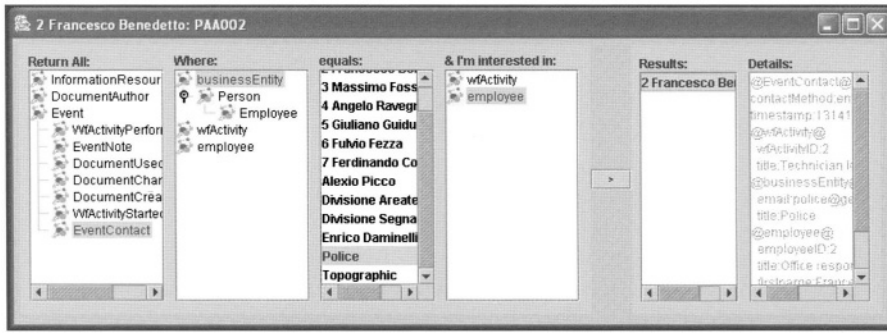


Fig. 2. Query creation GUI. Query on the picture can be read as follows: Return All EventContacts where BusinessEntity (contact contacted) is Police Department and I am interested in the employee, which made the contacts. Note that ontological elements adopt tree structure from inheritance of ontology elements.

This simplified algorithm describes how we create FIPA-SL query by using ontology and GUI for user selection of elements. The result of this algorithm is FIPA-SL query as in the example bellow and also as on the Fig. 2. Answering agent transforms this query to objects and compares object instances in OM to this query using Java reflec-tion methods. Automatic querying of InformationResources based on Event type and its connection to a workflow activity is the basis of Pellucid ontology and algorithms. By simple extending InformationResource with Document and creating several Events as docu-mentUsed or DocumentCreated extra functionality is added.

4 Conclusion

In this paper we described how agent based architecture can be used in distributed knowledge management systems. We focused on the knowledge based on true facts only, because this way we do not have to deal with contradictory knowledge and logical operations can be used for evaluating and discovery of new knowledge. It is clear that using of such knowledge is not usable for all possible knowledge management applications, but it is extremely useful for the area of applications dealing with administration processes where workflow is reasonably well defined. We believe similar approach with some modification can be and will be used in next generation of knowledge management systems in the commercial area as well.

References

1. Pellucid Consortium: Pellucid Project Technical Annex. (2001)
2. FIPA: FIPA Specification ACL Message Structure. http://www.fipa.org/. (2000)
3. FIPA: FIPA Communicative Act Repository Specification. http://www.fipa.org/. (2000)
4. Paul P. Wang (Editor): Computing with Words. ISBN: 0-471-35374-4, (2001)
5. Michael Wooldridge: Introduction to MultiAgent Systems, ISBN: 047149691X, (2002)
6. Ontoweb Consortium: Ontoweb Portal, http://www.ontoweb.org/ (2002)
7. R. Slota, K. Krawczyk, M. Dziewierz, J. Kitowski, S. Lambert: Agent paradigm for accessing document repositories in Pellucid platform, Proceedings of EuroWeb 2002 Conference, Oxford, UK, 17-18 (December 2002), pp.192-194.
8. R. Slota, M. Majewska, M. Dziewierz, K Krawczyk, M. Laclavik, Z. Balogh, L. Hluchy, J. Kitowski, S. Lambert: Ontology Assisted Access to Document Repositories for Public Sector Organizations. PPAM Conference (2003)
9. DARPA: DAML Website, http://www.daml.org/ (2002)
10. Telecom Italia Lab: JADE website, http://sharon.cselt.it/projects/jade/ (2002)
11. Pellucid Consortium: Technical Report of Pellucid Architecture, version 2003-04-15, (2003)
12. M. Laclavik, Z. Balogh, L. Hluchy, G. T. Nguyen, I. Budinska, T. T. Dang: Pellucid Agent Architecture for Administration Based Processes, IAWTIC 2003, Vienna (2003)
13. FIPA: FIPA Ontology Service Specification, http://www.fipa.org/ (2000)
14. Giovani Caire: JADE Tutorial Application-defined Content Languages and Ontology, http://jade.cselt.it/ (2002)
15. University of Amsterdam: Ontology Bean Generator for JADE, http://www.swi.psy.uva.nl/usr/aart/beangenerator/ (2002)

Ontology Assisted Access to Document Repositories in Public Sector Organizations

Renata Słota[1], Marta Majewska[2], Mariusz Dziewierz[2], Krzysztof Krawczyk[2], Michal Laclavik[3], Zoltan Balogh[3], Ladislav Hluchy[3], Jacek Kitowski[1,2], and Simon Lambert[4]

[1] Institute of Computer Science AGH-UST, al. Mickiewicza 30, Cracow, Poland
{rena, kito}@uci.agh.edu.pl
[2] ACC CYFRONET-AGH, ul.Nawojki 11, Cracow, Poland
{mmajew,aristot, krafcoo}@icsr.agh.edu.pl
[3] Institute of Informatics, SAS, Dubravska cesta 9, Bratislava 84237, Slovakia
{laclavik.ui, balogh.ui, hluchy.ui}@savba.sk
[4] CCLRC Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX11 0QX, UK
S.C.Lambert@rl.ac.uk

Abstract. This paper describes a method that combines ontology and full-text indexing to search documents in organizational repositories. The method was used in a system supporting organizational activity by formalizing, sharing and preserving operational experience and knowledge for future use. The paper is based on results obtained from EU IST Pellucid Project.

1 Introduction

Modelling business processes is a challenging task at present, due to its theoretical and practical scopes of interest. Knowledge of employees is an asset of an organization and therefore knowledge management is one of the useful approaches. Knowledge management is to capture different kind of knowledge of an organization and to organize access, sharing and reuse of that information [1, 2]. Implementation of knowledge management into the organization can result in improvement in knowledge sharing and cooperative work inside and outside the organization as well as in capitalization of past knowledge, which results in superior quality of everyday work and easy adaptation to menace situations.

A variety of techniques is used to deal with knowledge management, for example implementation of Organizational Memories (OM), generic and domain specific ontologies, Workflow Management Systems (WfMS), case-based reasoning systems and distributed, collaborative multi-agent approaches (e.g. [1]). Practical applications of the organizational knowledge management cover, amongst other problems, efficient usage of previous inter-personal contacts between the employees and of document base existing in the organization.

One of the important issues is workplace organization. When experienced workers leave a job most of their knowledge is lost for the organization. It results in reduction of organization efficiency and effectiveness. New workers need

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 700–705, 2004.
© Springer-Verlag Berlin Heidelberg 2004

some, usually stressful time to become familiar with the scope of duties at the workplace. The organization spends money on training. The problem could be diminished by computer-based information systems. The system could support the employees in searching for the required information and assist in performing tasks. The knowledge of the organization could be protected by accumulation information about each workplace and reused in the future.

In this paper an experience-based method of accessing documents in organizational repositories is described. Presented results were obtained from the EU IST Pellucid Project [3,4,5]. In Sect. 2 the state of the art is presented. In Sect. 3 two methods of search and access to documents employed in the access layer of the Pellucid platform are explained. The section ends with detailed description of query execution. Conclusions are presented in Sect. 4.

2 State of the Art

Subject matter of this paper is broadly discussed. The article [6] presents an approach for the knowledge management systems actively assisting the users. Such systems take advantage of ontologies for the description of knowledge concepts and their usage context and support ontology querying. The problem of the ontology-based searching is discussed in the article [7]. The author proposes usage of concepts' semantic for the transformation of queries (query extension, query clarification, query relaxation) and postulates generation of the set of topics for use by the user on the basis of ontologies. In the article [8] a multi-agent knowledge management system, which allows extensive access to the knowledge stored in diverse organizational repositories is described. The system uses complex ontology of repositories to enable easy navigation through the them and gathers experience for future use. DECOR is a toolkit for business process oriented knowledge management supported by the embedded WfMS [9]. DECOR uses ontologies for the description of the context of document creation and usage. Access to the documents is performed by the archive system using metadata for the description of document properties like author, version or language.

3 Access to Documents in Pellucid Platform

The Pellucid is a software agent-based platform that makes use of modern technologies such as: OM, a WfMS and ontologies [10] for gathering, capitalising, storing and sharing knowledge to support mobile employees of public sector organizations. The platform has three-tiered architecture and consists of: the interaction layer, concerned with managing the interface with the employee (end-user) and the external world; the process layer, concerned with managing tasks and workflows; and the access layer, concerned with search and retrieval of a wide range of documents. Agents of each layer collaborate with the OM. The OM stores all knowledge gathered during the system lifecycle. An employee, through WWW interface, gets general information about his/her business actions, as well as detailed knowledge and experience, accumulated in the past, about particular cases. With the help of context from the WfMS the information is matched

against user needs. Ontologies are used not only for descriptions of documents and workflow, but also for a description of repositories.

The method proposed for document searching merges two types of searches: ontology-based search and index-based search [11]. Both kinds of searches are executed separately and their results are combined together. The exploitation of ontologies and full-text indexing makes possible searching in two dimensions. The first one is concerned about meaning of documents to an organization while another gives the notion of documents' content. Ontological description of documents allows linking them to events occurring in the organization and also to annotate them. This gives ability to search for documents which are correlated in some way with organizational activities or have specific attributes described by annotations. The full-text indexing is useful when we need to search a document knowing even a small part of its content like a phrase or a single word.

3.1 Ontologies

The general definition states that ontology is a model of real world, created for future multiple uses [12]. The ontology is focused on structural dependencies rather then functional ones. Formally, the ontology is a description of ideas in a particular domain and relations between them [13]. The basis for ontology creation and development are questions in natural language, which are a part of environment and requirements analysis [14].

All ontologies in the Pellucid project are defined according to the Content Reference Model originating from JADE (Java Agent Development Framework). The mentioned model applies two main terms: a concept and a predicate. The concept is an entity with complex structure defined, with the aid of slots. The predicate is an expression, which tells something about world status and could have value true or false.

The core elements of the generic ontology are an information resource and an event. Each event is connected with an activity in the workflow process and with the employee, who generated it. An information resource is any document or contact. The following paragraphs of this section, presents one part of the generic ontology that is concerned with the document retrieval.

Document Ontology. The document ontology is shown in Fig. 1. The document is a data set, which contains information important for the organization, e.g. a text editor's file, a spreadsheet, a web page, an email or a database report. If a new document is created or an existing one is read or modified, the system notes it down. In the organizational memory instances of predicates DocumentCreated, DocumentUsed and DocumentChange are created. Those predicates are inherited from DocumentEvent predicate. Predicate DocumentAuthor identifies authors of the document – persons or units of the organization (a concept BusinessEntity). Content of the document could be a plain text, a multimedia format, binary code, a database element, etc. The predicate TextQuery is dedicated for the realization of the full-text search. The concept AbstractDocument inherits from the concept InformationResource connections to pred-

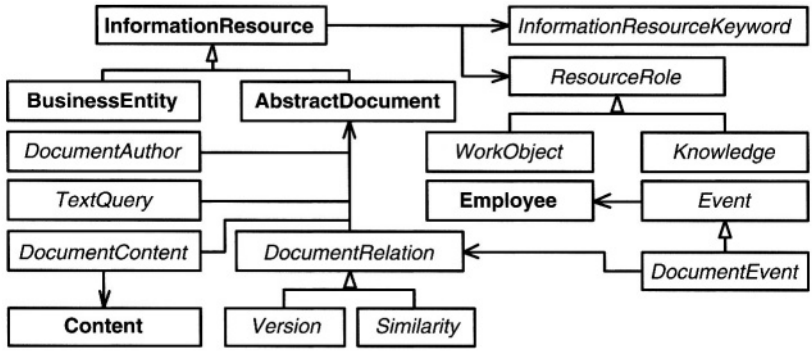


Fig. 1. The diagram of the document ontology.

icates ResourceRole and InformationResourceKeyword. The abstract predicate ResourceRole describes the status of the document in the system. First, document is currently used and modified by employees (a predicate WorkObject). Second, document is a historical object, containing fixed and unchanging knowledge (a predicate Knowledge). A predicate InformationResourceKeyword links a document with a set of attributive keywords. An abstract predicate DocumentRelation identifies a relation between two or more documents. Semantic of this predicate is described by predicates Similarity (similarity between documents) and Version (next version of a document).

3.2 Full-Text Indexing

Indexing organization documents offers another way for document search. Indexing, search and access activities are performed by: *1)* indexing engine that provides functionality of creating indexes from content of documents; *2)* search engine that uses indexes created earlier by indexing engine to locate documents; *3)* document and repository drivers that provide uniform access to documents written in different file formats and located in diverse repositories. The simplest query specifies a single term or word that is to be matched against each document in the index. More complex queries may contain nested queries joined with operators like: "and", "or", "not". Queries may also specify hints that may effects the ranking of the matched documents. Most of indexing engines require input being a plain text and therefore document drivers are used to convert text from the particular document format to the plain text form.

3.3 Search Execution

Queries are created manually by the employee or automatically by the system on purpose of support the employee with a list of recommended contacts and documents. The Query Engine, which combines two search methods – ontology-based and index-based – handles the query execution as shown in Fig. 2. The

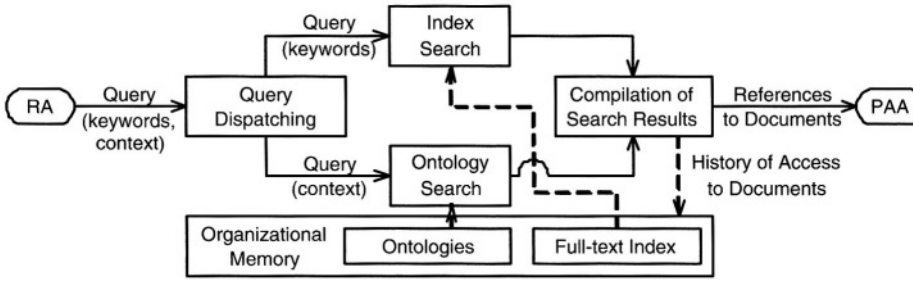


Fig. 2. The process of document search.

query coming from the employee or from the system is sent to Role Agent (RA) which adds the context from the WfMS to the query and forwards it to the access layer. Information Search Agent retrieves documents and contacts that are matching the query. The obtained list of references is returned to the agent of the interaction layer, namely Personal Assilstant Agent (PAA).

Ontology-based queries execution starts with localizing the organizational memory elements that meet requirements. The requirements are specified by predicates that connect groups of concepts and templates of concepts. A simple query consists of a single predicate only, while a complex – of many predicates linked with logical operators. Index-based queries are built with the help of the virtual predicate TextQuery. Instances of this predicate are not kept in the organizational memory, but they are dynamically generated, as results of queries of the searching engine.

Consider a sample question in the natural language: "Find all employees, who work in the activity Budget control". The query written in FIPA Semantic Language looks like:

```
(Event :employee (Employee)
       :wfActivity (WfActivity :title "Budget control"))
```

The query uses concepts of Employee and WfActivity that are connected by the predicate Event. The concept Employee in the form shown in the example will match any employee while the concept WfActivity will match only these activities which have title "Budget control".

4 Conclusions

In this paper the method of document retrieval and repository access based on combination of document indexing and ontologies has been proposed. Separated usage of those two methods is inefficient – the searching based on full-text indexes returns a broad set of answers, often inconsistent with the needs, while the searching based on ontologies is not able to answer all questions, because the ontologies do not qualify all information contained in the text of document. Postulated searching is more exact, allows getting precise answers for the asked

questions through linking a query context with information got from ontologies and documents. The method is being implemented into the Pellucid project.

Acknowledgments. Thanks are due to European Commission and participants of the Pellucid project, IST-2001-34519. AGH-UST grant is also acknowledged.

References

1. Dieng-Kuntz, R., Matta, N., "Knowledge Management and Organizational Memories", Kluwer Academic Publishers, Boston, 2002.
2. Davenport, T.H. and Prusak, L., "Working Knowledge: How Organizations Manage what They Know", Harvard Business School Press, 1998.
3. Pellucid – A Platform for Organisationally Mobile Public Employees, EU IST-2001-34519 Project, 2002, http://www.sadiel.es/Europa/pellucid/.
4. Kitowski, J., Lambert, S., Słota, R., Krawczyk, K., Dziewierz, M., "Pellucid project and its multilayered agent-based architecture", Proc. of PIONIER – Polish Optical Internet, Polish Conference, April 23-24, 2002, Poznań, 2002, Instytut Informatyki Politechniki Poznańskiej, pp. 221-233.
5. Lambert, S., Stringa, S., Vianno, G., Kitowski, J., Słota, R., Krawczyk, K., Dziewierz, M., Delaitre, S., Oroz, M.B., Gomez, A.C., Hluchy, L., Balogh, Z., laclavik, M., Fassone, M., Contursi, V., "Knowledge management for organisationally mobile public employees", accepted for 4th Working Conf. on Knowledge Management in Electronic Government, May 26-28, 2003, Rhodes, Greece.
6. Abecker, A., Bernardi A., Hinkelmann K., Kühn O., and Sintek M., "Towards a Technology for Organizational Memories", IEEE Intelligent Systems, 1998.
7. McGuinness D.L., "Ontological Issues for Knowledge-Enhanced Search", Frontiers in Artificial Intelligence and Applications, IOS-Press, Washington, DC, 1998.
8. Sintek, M., B. Tschaitschian, A. Abecker, A. Bernardi, and H.-J. Müller, "Using Ontologies for Advanced Information Access", http://www.dfki.uni-kl.de/~aabecker/Postscript/MSiAA-Pakem2000-header.pdf.
9. Abecker A., Bernardi A., Dioudis S., Elst L., Herterich R., Houy C., Legal M., Mentzas G., Müller S., "Workflow-Embedded Organizational Memory Access: The DECOR Project", http://imu.iccs.ntua.gr/Papers/C47-IJCAI-DECOR.pdf.
10. Laclavik, M., Balogh, A., Hluchy, L., Słota, R., Krawczyk, K. and Dziewierz, M., "Distributed Knowledge Management based on Software Agents and Ontology", 5th Int. PPAM 2003 Conf., Sept. 7-10, 2003, Częstochowa, Poland, submitted.
11. Słota, R., Krawczyk, K., Dziewierz, M., Majewska, M., Kitowski, J., and Lambert, S., "Optimisation of access to document repositories in EU Pellucid project", presented at PIONIER 2003 National Conf. – Polish Optical Internet, April 9-11, 2003, Poznań, Poland, accepted for publication in Conf. Proc. (in Polish).
12. Staab, S., Studer, R., Schnurr, H.-P., Sure, Y., "Knowledge Processes and Ontologies", IEEE Intelligent Systems, Jan./Feb. 2001, pp. 3-10.
13. Mike Uschold, Michael Gruninger, "Ontologies: Principles, Methods and Applications", Knowledge Engineering Review, 11(2), June 1996, pp. 93-137.
14. IST Project IST-2000-29243 Onto Web, "Deliverable 1.4: A survey on methodologies for developing, maintaining, evaluating and reengineering ontologies", http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/D1.4-v1.0.pdf.

Simulations of Granular Cohesion Dynamics on Rough Surfaces

Jacek S. Leszczynski

Czestochowa University of Technology, Institute of Mathematics & Computer Science, ul. Dabrowskiego 73, 42-200 Czestochowa, Poland
{jale,cmariusz}@k2.pcz.czest.pl

Abstract. We consider the complex problem of how to perform simulations particle motions taking into account cohesion between particles. We will focus on the molecular dynamics models. Particularly, we will analyse what happens in cohesive materials during multiparticle contacts where the surfaces of contacting particles are rough.

1 Introduction

The dynamics of granular materials has generated much interest in natural phenomena, physics and engineering communities. Discrete models [11] can reflect the collision process more realistic. Molecular dynamics models [11] leads to an assumption of an expression of the repulsive force acting between a pair of contacting particles. In this paper we will concentrate on the molecular dynamics models. Consequently we will analyse the fractional interaction law [4] of the repulsive force in order to simulate multiparticle contacts.

2 Modelling of Particle Motion

We consider a set of spherical particles moving under optional extortion. Taking into account an individual particle, we can characterise the particle trough its radius r_i, mass m_i, inertia moment $\mathcal{I}_i$, position $\mathbf{x}_i$, linear speed $\dot{\mathbf{x}}_i$ and angular velocity $\boldsymbol{\omega}_i$, for $i = 1, \ldots, np$. We can describe motion of an individual particle colliding with neighbouring particles as

$$\begin{cases} m_i \cdot \ddot{\mathbf{x}}_i + \sum\limits_{j(i),\, j(i)\neq i} \mathbf{Q}_{j(i)} = \sum\limits_{l} \mathbf{F}_l \\ \mathcal{I}_i \cdot \dot{\boldsymbol{\omega}}_i + \sum\limits_{j(i),\, j(i)\neq i} \mathbf{s}_{j(i)} \times \mathbf{Q}_{j(i)} = \sum\limits_{l} \mathbf{M}_l \end{cases} . \tag{1}$$

Above formula is devoted to motion of the particle i which collides with neighbouring particles indexed by the natural function $j(i)$ $(j(i) \neq i)$, where $\mathbf{F}_l$ denotes an optional force, $\mathbf{M}_l$, is an optional torque, $\mathbf{Q}_{j(i)}$ is a repulsive force and $\mathbf{s}_{j(i)}$ indicates the branch vector connecting the mass centre of particle i with the point of application of the repulsive force $\mathbf{Q}_{j(i)}$.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 706–713, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2.1 Coordinate System and Definition of the Overlap

The form of repulsive force is defined in the molecular dynamics models [11]. In these models particles during a contact virtually overlap. We define the overlap of two particles experiencing a contact in the following form

$$\left\|\boldsymbol{\zeta}_{j(i)}\right\| = r_{j(i)} + r_i - \left\|\mathbf{x}_{j(i)} - \mathbf{x}_i\right\|. \tag{2}$$

In the case $\left\|\boldsymbol{\zeta}_{j(i)}\right\| = 0$ one can find a time $t^*_{j(i)}$ when a collision begins.

A normal unit vector $\mathbf{e}_{\zeta\, j(i)}$ that connects the particles' centres of mass reads

$$\mathbf{e}_{\zeta\, j(i)} = \frac{\mathbf{x}_{j(i)} - \mathbf{x}_i}{\left\|\mathbf{x}_{j(i)} - \mathbf{x}_i\right\|} = \left[e^x_{\zeta\, j(i)}, e^y_{\zeta\, j(i)}, e^z_{\zeta\, j(i)}\right], \tag{3}$$

Following on from above considerations we need to define tangential unit vectors which operates on a plane being tangent to the normal component (3). Thus, we have

$$\mathbf{e}_{\eta\, j(i)} = \left[e^y_{\zeta\, j(i)}, -e^x_{\zeta\, j(i)}, 0\right] \cdot \frac{\left\|\mathbf{x}_{j(i)} - \mathbf{x}_i\right\|}{\left\|\mathbf{x}_{j(i)} - \mathbf{x}_i\right\|_{x,y}}, \qquad \mathbf{e}_{\xi\, j(i)} = \mathbf{e}_{\eta\, j(i)} \times \mathbf{e}_{\zeta\, j(i)}. \tag{4}$$

Taking into account above expressions we introduce vectors of the overlap (2) as

$$\boldsymbol{\zeta}_{j(i)} = \left\|\boldsymbol{\zeta}_{j(i)}\right\| \cdot \mathbf{e}_{\zeta\, j(i)} = \left(r_{j(i)} - r_i\right) \cdot \mathbf{e}_{\zeta\, j(i)} - \left(\mathbf{x}_{j(i)} - \mathbf{x}_i\right), \tag{5}$$

$$\boldsymbol{\eta}_{j(i)} = \left\|\boldsymbol{\zeta}_{j(i)}\right\| \cdot \mathbf{e}_{\eta\, j(i)}, \quad \boldsymbol{\xi}_{j(i)} = \left\|\boldsymbol{\zeta}_{j(i)}\right\| \cdot \mathbf{e}_{\xi\, j(i)}. \tag{6}$$

2.2 Fractional Interaction Law for the Repulsive Force

We neglect here a description of basic interaction laws which one can find in [1, 3,12]. Several studies [4,6] connected with the fractional interaction law detect that this law operates on rough surfaces of contacting particles. According to [4] we have the normal force in the following form

$$\mathbf{Q}_{\zeta_{j(i)}} = c^{\alpha}_{j(i)} \cdot k^{1-\alpha}_{j(i)} \cdot {}_{t^*_{j(i)}}D^{\alpha}_{t_{j(i)}}\left(\boldsymbol{\zeta}_{j(i)}\right), \tag{7}$$

where $k_{j(i)}$, $c_{j(i)}$ are spring and damping coefficients, $t^*_{j(i)}$ is a time when a collision begins, $t_{j(i)}$ is a time when a collision ends, ${}_{t^*_{j(i)}}D^{\alpha}_{t_{j(i)}}\left(\boldsymbol{\zeta}_{j(i)}\right)$ denotes the differential operator of fractional order α. According to [9] we introduce a definition of this operator

$${}_{t^*}D^{\alpha}_t f(t) = \frac{1}{\Gamma(n-\alpha)} \cdot \frac{d^n}{dt^n} \int_{t^*}^{t} f(\tau) \cdot (t-\tau)^{n-\alpha-1}\, d\tau =$$

$$\sum_{l=0}^{n-1} \frac{(t-t^*)^{l-\alpha}}{\Gamma(l-\alpha+1)} \cdot f^{(l)}(t^*) + {}^{C}_{t^*}D^{\alpha}_t f(t), \tag{8}$$

where the sum represents initial conditions and ${}_{t^*}^{C}D_t^{\alpha} f(t)$ is the Caputo fractional derivative

$$ {}_{t^*}^{C}D_t^{\alpha} f(t) = \begin{cases} \frac{1}{\Gamma(n-\alpha)} \cdot \int\limits_{t^*}^{t} \frac{\frac{d^n f(\tau)}{d\tau^n}}{(t-\tau)^{\alpha-n+1}} d\tau, & n-1 < \alpha < n \\ \frac{d^n}{d(t-t^*)^n} f(t), & \alpha = n \end{cases}, \tag{9} $$

where $n = [\alpha] + 1$ and $[\cdot]$ denotes an integer part of a real number. It should be noted in [4] that in multiparticle contacts we need to take into account the sum of forces (7) running over the all contacting particles. As explained in [4] the parameter α represents the degree of conversion of impact energy into viscoelasticity of the material. In analysis of the behaviour of dynamics of an "object" composed with several contacting particles we assume that the parameter α is a sum of two independent components. The first one reflects the conversion degree between a binary contact and we denoted this as $\alpha_{j(i)}$. The second is the induced degree of conversion which depends on impact energy. In this general case the parameter α should take a form

$$ \alpha = \alpha_{j(i)} + \alpha', \tag{10} $$

where α' is the induced degree of conversion and $\alpha_{j(i)}$ is the conversion degree between one binary contact.

Here we introduce the normal force of a particle i acting on a particle $j(i)$ as

$$ \mathbf{N}_{\zeta\, j(i)} = \max\left[0, \left(\left(\mathbf{Q}_{\zeta\, j(i)} + \sum_{l} \left(\mathbf{F}_l \cdot \mathbf{e}_{\zeta\, j(i)}\right) \mathbf{e}_{\zeta\, j(i)}\right) \cdot \mathbf{e}_{\zeta\, j(i)}\right) \mathbf{e}_{\zeta\, j(i)}\right]. \tag{11} $$

We define the frictional force which operates in the tangential plane as

$$ \mathbf{T}_{j(i)} = -\mu \cdot \mathbf{N}_{\zeta\, j(i)} \cdot \operatorname{sign}\left(\mathbf{u}_{j(i)}^{t}\right), \tag{12} $$

where $\mathbf{u}_{j(i)}^{t}$ is the velocity component being tangent to the normal unit vector (3). According to above considerations we introduce here the tangential forces as

$$ \mathbf{Q}_{\eta_{j(i)}} = \left(\mathbf{T}_{j(i)} \cdot \mathbf{e}_{\eta\, j(i)}\right) \mathbf{e}_{\eta\, j(i)}, \tag{13} $$

$$ \mathbf{Q}_{\xi_{j(i)}} = \left(\mathbf{T}_{j(i)} \cdot \mathbf{e}_{\xi\, j(i)}\right) \mathbf{e}_{\xi\, j(i)}. \tag{14} $$

3 Solution Procedure

For the contact detection and administration for the interaction law one can find interesting procedures in literature [5,11].

3.1 Analytical Solution for a Binary Contact

Throughout this section we shall use a simplified case when a two-particle collide. We also neglect optional forces as well as tangential forces. The system of two colliding particles in 1D is presented as

$$\begin{cases} m_1 \cdot \ddot{x}_1 + c^\alpha \cdot k^{1-\alpha} \, {}_{t^*}D_t^\alpha (\zeta) = 0 \\ \\ m_2 \cdot \ddot{x}_2 - c^\alpha \cdot k^{1-\alpha} \, {}_{t^*}D_t^\alpha (\zeta) = 0 \end{cases}, \tag{15}$$

where ζ equals $\zeta = r_1 + r_2 + x_1 - x_2$. We introduce relative coordinates as:

$$\begin{cases} w = \frac{m_1 \cdot x_1 + m_2 \cdot x_2}{m_1 + m_2} \\ \\ v = x_1 - x_2 \end{cases}. \tag{16}$$

After mathematical calculations we show the system (15) in the relative coordinates (16) as

$$\begin{cases} \ddot{w} = 0 \\ \\ \ddot{v} + c^\alpha \cdot k^{1-\alpha} \cdot \left(\frac{1}{m_1} + \frac{1}{m_2} \right) \cdot \left[{}_{t^*}D_t^\alpha (v) + (r_1 + r_2) \cdot {}_{t^*}D_t^\alpha (1) \right] = 0 \end{cases} \tag{17}$$

We also need to assume initial conditions and for the begin of contact $t^* = 0$ we have $x_1(0) = r_1$, $x_2(0) = r_2$, $\dot{x}_1(0) = u_1$, $\dot{x}_2(0) = u_2$. On the base of the theory of fractional calculus [9] we found analytical solution of Eqn (17) in the following form

$$\begin{cases} w(t) = \frac{m_1 \cdot u_1 + m_2 \cdot u_2}{m_1 + m_2} \cdot t + \frac{m_1 \cdot r_1 + m_2 \cdot m_2}{m_1 + m_2} \\ \\ v(t) = (u_1 - u_2) \cdot t \cdot E_{2-\alpha,2} \left(-A \cdot t^{2-\alpha} \right) \\ \qquad +2 \cdot r_1 \cdot E_{2-\alpha,1} \left(-A \cdot t^{2-\alpha} \right) - (r_1 + r_2) \end{cases}, \tag{18}$$

where

$$A = c^\alpha \cdot k^{1-\alpha} \cdot \left(\frac{1}{m_1} + \frac{1}{m_2} \right), \tag{19}$$

and $E_{\beta,\gamma}(\rho)$ is the Mittag-Leffler function [9] defined as

$$E_{\beta,\gamma} (\rho) = \sum_{l=0}^{\infty} \frac{\rho^l}{\Gamma (l \cdot \beta + \gamma)}. \tag{20}$$

For the relative coordinate $v(t)$ in Eqn (18) we can add factor $(r_1 + r_2)$ and then we obtain a variation in time of the overlap for two colliding particles.

3.2 Numerical Solution

In order to analyse the dynamics of multiparticle contacts in the general form we need to construct a computational procedure that to perform simulations.

Therefore we divide time of calculations into $0 < t_0 < \ldots < t_k < \ldots < T$, where T indicates the total time in which the calculations are considered. For this we can introduce the time step as $h_k = t_k - t_{k-1}$. We also introduce the total number of particles *np* and for an individual particle we assume its physical, geometrical and additional properties, and initial conditions $\mathbf{x}_i^0$, $\dot{\mathbf{x}}_i^0$, where the superscript 0 refers time $t = 0$. The first stage of our calculations is an algorithm for collision detection. The algorithm described in [5] detects a begin of collision between a pair of neighbouring particles and of course the collision end. Next we consider formula (1) neglecting the contact force for an individual particle which does not experience any contact or formula (1) describing motion of a particle experiencing several binary contacts with neighbouring particles. If a contact occurs we use the fractional interaction law throughout formulae (7), (13) and (14). In this case Eqn (1) becomes the ordinary differential equation with a mixture of operators. The fractional derivative (9) existed in Eqn (1) needs to be represented by its numerical form. In this point of our considerations we introduce a time $t^*_{j(i)}$ when a collision begins and a time $t_{j(i)}$ when the collision ends. These times result from the algorithm described in [5].

The general idea of how to solve numerically ordinary differential equations of fractional order is shown in [7]. We sightly modify this idea in order to adapt it for an ordinary differential equation with integer and fractional derivatives. Note that the fractional derivative is defined in the time interval $\left\langle t^*_{j(i)}, t_{j(i)} \right\rangle$ and $t_c = t_{j(i)} - t^*_{j(i)}$ denotes a duration of collision in time. The local index m is the total number of divisions of the collisional time t_c into time steps. Note that m varies over the overlap path (2) and therefore this index results from direct calculations. According to fractional calculus [9] we present here the numerical form for the fractional derivative (9) as

$$ {}^{C}_{t_0}D^{\alpha}_{t_m} f(t) = \frac{1}{\Gamma(n-\alpha+1)} \sum_{l=0}^{m-1} B_l \cdot \left[(t_m - t_l)^{n-\alpha} - (t_m - t_{l+1})^{n-\alpha} \right], \quad (21) $$

where

$$ B_l = f^{(n)}(t_l), \quad (22) $$

and $n = [\alpha] + 1$, $[\cdot]$ indicates an integer part of a real number. Above scheme assumes that the function $f(t)$ has a constant value in the time interval $t_l < t < t_{l+1}$. In this paper we improve formula (21) in order to assume a linear form of the function $f(t)$ in the time interval $t_l < t < t_{l+1}$. Thus we have

$$ \begin{aligned} {}^{C}_{t_0}D^{\alpha}_{t_m} f(t) = \frac{1}{\Gamma(n-\alpha)} \sum_{l=0}^{m-1} \Bigg\{ & \frac{A_l}{n-\alpha+1} \cdot \left[(t_m - t_{l+1})^{n-\alpha+1} - (t_m - t_l)^{n-\alpha+1} \right] \\ & - \frac{A_l \cdot t_m + B_l}{n-\alpha} \cdot \left[(t_m - t_{l+1})^{n-\alpha} - (t_m - t_l)^{n-\alpha} \right] \Bigg\}, \end{aligned} \quad (23) $$

where

$$ A_l = \frac{f^{(n)}(t_{l+1}) - f^{(n)}(t_l)}{t_{l+1} - t_l}, \qquad B_l = f^{(n)}(t_l) - A_l \cdot \frac{t_{l+1} + t_l}{2}. \quad (24) $$

Formula (23) is more precise in numerical calculations of the fractional derivative than formula (21). Diethelm et al [2] proposed numerical scheme involving formula (23) for the predictor-corrector method. Insteed of this method we use typical Runge-Kutta method in which dicrete form (23) is used. Note that Eqn (23) becomes Eqn (21) when $A_l = 0$.

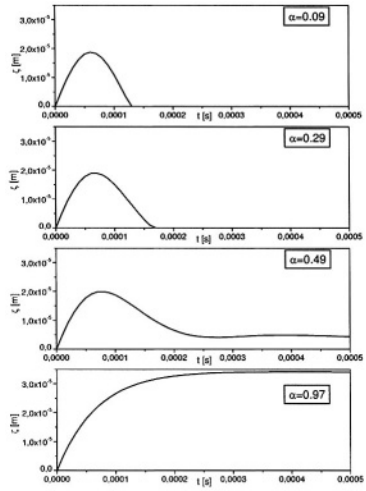


Fig. 1. The overlap over time for a binary contact and for different values of α.

4 Results and Analysis

First example shows analytical solution (18) where the factor $(r_1 + r_2)$ was added. The solution represents the overlap for a two particle contact. We assume spherical particles with their characteristic parameters as $r_1 = r_2 = 0.0015\,m$, $\rho_1 = \rho_2 = 1000\,kg/m^3$, $x_1^0 - x_2^0 = r_1 + r_2$ means that in time $t = 0\,s$ a contact occurs, $\dot{\mathbf{x}}_1 = 1\,m/s$, $\dot{\mathbf{x}}_1 = 0.5\,m/s$. During a contact we determine the following parameters $k = 5000\,N/m$, $c = 0.1\,kg/s$. Fig. 1 presents the overlap (2) over time for a two particle contact, for different values of the parameter α. For different values of the parameter α we can observe that the contact time t_c between a pair of colliding particles is finite being dependent on α. When α increases from 0.09 to 0.29 we noticed a projection of the contact time from $t_c = 1.12 \cdot 10^{-4}s$ to $t_c = 1.43 \cdot 10^{-4}s$. For higher values of the parameter α ($\alpha = 0.49$, $\alpha = 0.97$) notices infinitesimally long the contact time t_c. Therefore we can observe permanent cohesion in a two particle contact. This means that two particles stay clustered over time of calculations. It can be shown that the maximal penetration depth in the overlap increases when the conversion degree α is increased. In higher values of the conversion degree α we cannot find the maximal penetration depth in comparison to lower values of α. This can be typical behaviour for contacting materials which characterise roughness of their

contacting surfaces. On the other hand, if rough contacting surfaces may occur, strong dissipation of impact energy dominates during a collision. With regard to physical sense of the conversion degree α presented in [4] we are able to note that this parameter characterises the conversion of the impact energy into viscoelasticity of contacting surfaces where the surfaces are rough. Additional interesting case occurs for $\alpha = 0.49$. Analysing the overlap over time for $\alpha = 0.49$ one can notice two stages of variations of this overlap. The first stage is typical as for lower values of the conversion degree α, when the maximal penetration depth occurs. This means that during a contact elastic forces dominate and the overlap decreases over time after the maximal penetration depth. Nevertheless, the overlap does not decrease to zero and tends toward to a small value. This means that there is no enough energy created by the elastic forces that to break this contact and therefore viscous forces dominates.

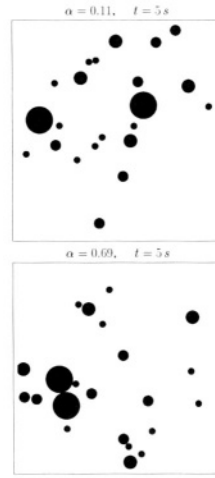


Fig. 2. Granular cohesion dynamics being dependent on α.

Simulations of particle motions in a box for different values of the parameter α shown in Fig. 2.

We assumed the same initial conditions and parameters of a structure for both states characterised by $\alpha = 0.11$ and $\alpha = 0.69$. This figure presents positions of particles in time of calculations $t = 5\,s$. For a low value of the conversion degree α ($\alpha = 0.11$) binary contacts can be observed. Therefore we cannot see particle contacts after $t = 5\,s$ because during binary contacts the contact time is very low and equals to $t_c = 1.15 \cdot 10^{-4}\,s$. In opposite to previous situation, when the parameter α increases to $\alpha = 0.69$ we noticed particles cohesion. In this case increases the contact time between particles and particles stay clustered through some part of the time of calculations.

5 Concluding Remarks

In this paper we analysed of how the fractional interaction law operates during multiparticle contacts. We extended this law introducing a definition of the repulsive force in the tangential direction. It should be noted that the fractional interaction law is defined under the fractional derivatives, where the derivatives accumulate the whole history of the particle overlap over time in weighted form. This is a great advantage of this fractional law in order to perform simulations of multiparticle collisions in arbitrary from as well as granular cohesion dynamics. In opposite to basic interaction laws, which operate on smooth contacting surfaces of a two-particle collision, the fractional interaction law operates in the rough surfaces in topological sense.

Acknowledgment. This work was supported by the State Committee for Scientific Research (KBN) under the grant 4 T10B 049 25.

References

1. Cundall P.A. and Strack O.D.L.: A discrete numerical model for granular assemblies, Geotechnique **29** (1979), pp. 47-65
2. Diethelm K., Ford N.J. and Freed A.D., A Predictor-Corrector Approach for the Numerical Solution of Fractional Differential Equations, Nonlinear Dynamics **29** (2002), pp. 3-22
3. Kuwabara G., Kono K., Restitution coefficient in a collision between two spheres, Jap. J. Appl. Phys. **26** Part 1 (1987), pp. 1230-1233
4. Leszczynski J.S.: A discrete model of a two-particle contact applied to cohesive granular materials, Granular Matter **5**(2) (2003), pp. 91-98
5. Leszczynski J.S.: Computer simulations of multiparticle - contacts dynamics, Lecture Notes in Computer Science **2328** (2003), pp. 105-114
6. Leszczynski J.S.: The calculation of a normal force between multiparticle contacts using fractional operators, Ed. K.J. Bathe, Computational Fluid and Solid Mechanics 2003, Elsevier Science (2003), pp. 2043-2047.
7. Leszczynski J.S., Ciesielski M.: A numerical method for solution of ordinary differential equations of fractional order, Lecture Notes in Computer Science **2328** (2002), pp. 695-702
8. Luding S., Cl'ement E., Blumen A., Rajchenbach J., and Duran J.: Anomalous energy dissipation in molecular dynamics simulations of grains, Physical Review E **50** (1994), pp. 4113-4122
9. Oldham K.B., Spanier J.: The Fractional Calculus. Theory and Applications of Differentiation and Integration to Arbitrary Order, Academic Press, New York (1974)
10. Pournin L., Liebling Th.M.: Molecular dynamics force models for better control of energy dissipation in numerical simulations of dense granular media, Physical Review E **65** (2001), pp. 011302-1-011302-7
11. Rappaport D.C.: The Art of Molecular Dynamics Simulation, Cambridge Univ. Press, Cambridge (1995)
12. Walton O.R., Braun R.L.: Viscosity, granular - temperature and stress calculations for shearing assemblies of inelastic frictional disks, J. Rheol. **30** (1986), pp. 949-980

Adaptive Noise Reduction in Microarray Images Based on the Center-Weighted Vector Medians

Rastislav Lukac[1], Bogdan Smolka[2]*, Andrzej Swierniak[2],
Konstantinos N. Plataniotis[3], and Anastasios N. Venetsanopoulos[3]

[1] Slovak Image Processing Center,
Jarkova 343, 049 25 Dobsina, Slovak Republic
lukacr@ieee.org
[2] Department of Automatic Control,
Silesian University of Technology, Akademicka 16 Str., 44-101 Gliwice, Poland
bsmolka@ia.polsl.gliwice.pl
[3] The Edward S. Rogers Sr. Dept. of Electrical and Computer Engineering,
University of Toronto, 10 King's College Road, Toronto, Canada
{kostas, anv}@dsp.utoronto.ca

Abstract. In this paper, we provide a new noise reduction method for the enhancement of the images of gene chips. We demonstrate that the new technique is capable of reducing outliers present in microarray images while preserving the spot edges. This paper contains the comparisons of the proposed technique with the standard schemes used in color image filtering, as well as examples of their efficiency when these methods are applied to typical microarray images.

1 Introduction

This paper focuses on a new nonlinear vector filtering scheme taking the advantage of center-weighted vector median filters and optimal filtering situation. The proposed method improves the signal detail preservation capability of the standard vector filtering schemes used for multichannel image filtering and provides the adaptive alternation between the identity operation and the robust vector median filtering. In addition, the computational complexity of the proposed method is still acceptable, which allows its application for the enhancement of the cDNA microarray images [1],[3]. Using the new filtering scheme, it is possible to achieve excellent balance between the signal-detail preservation and the noise attenuation. These properties of the proposed method were tested for a wide range of multichannel image signals such as phantom (artificial) images and real microarray chips. Using the phantom images, we can test the efficiency of the methods depending on the intensity of impulsive noise corruption, which is frequently introduced into the microarray images. We also provide a short description of the microarray cDNA images.

* B. Smolka is supported by the KBN grant 4T11F01824.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 714–721, 2004.
© Springer-Verlag Berlin Heidelberg 2004

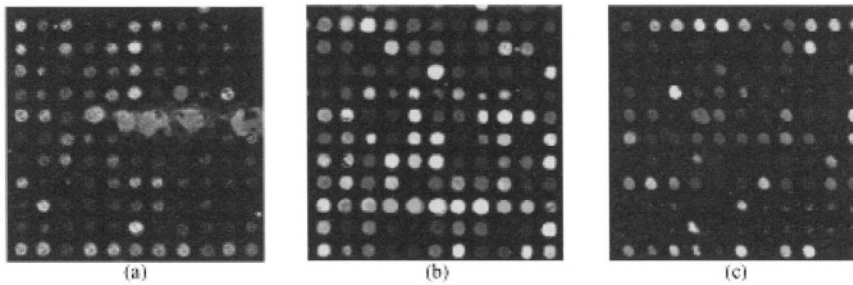


Fig. 1. Real cDNA miccroarray images.

2 Vector Median Filter

Concerning the multichannel image filtering [10], standard color images represent the vector-valued image signals, in which each image point can be considered as a vector of three components associated with intensities of color channels consisting of red (R), green (G) and blue (B). Thus, it is necessary to consider the correlation that exists between color channels and to apply the vector processing. If the existing correlation is not taken into account and color channels are processed independently, then the filtering operation is applied componentwise. In general, componentwise (marginal) approaches produce new vector samples, i.e. color artifacts, caused by composition of reordered channel samples.

Vector filters represent a natural approach to the noise removal in multichannel images, since these filters utilize the correlation between color channels. Therefore, the vector methods represent optimal and attractive approaches for studying and processing of noisy color images [2],[8],[10],[12].

The most popular nonlinear, multichannel (vector) filters are based on the ordering of vectors in a predefined sliding window. Designed to perform low-pass filtering, vector filtering schemes output the lowest ranked vector according to a specific ordering technique [10].

Let $y(x) : Z^l \rightarrow Z^m$ represent a multichannel image, where l is an image dimension and m denotes a number of channels. Let $W = \{\mathbf{x}_i \in Z^l; i = 1, 2..., N\}$ represent a filter window of a finite size *N*, where $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ is a set of noisy samples centered around the central sample $\mathbf{x}_{(N+1)/2}$.

Let us consider input sample $\mathbf{x}_i$, for $i = 1, 2, ..., N$, associated with the distance measure L_i given by

$$L_i = \sum_{j=1}^{N} \|\mathbf{x}_i - \mathbf{x}_j\|_\gamma, \text{ for } i = 1, 2, ..., N \tag{1}$$

where

$$\|\mathbf{x}_i - \mathbf{x}_j\|_\gamma = \left(\sum_{k=1}^{m} |x_{ik} - x_{jk}|^\gamma \right)^{\frac{1}{\gamma}} \tag{2}$$

characterizes the generalized Minkowski distance [10] between two multichannel samples $\mathbf{x}_i$ and $\mathbf{x}_j$. Notation γ characterizes the used norm, m is the dimension of vectors and x_{ik} is the k-th element of the sample $\mathbf{x}_i$. Note that the well-known Euclidean distance is associated with $\gamma = 2$.

Assuming that the ordering of $L_{(i)}$ implies the same ordering of the input set $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ results in the ordered set $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, ..., \mathbf{x}_{(N)}$, where $\mathbf{x}_{(i)}$ is associated with $L_{(i)}$. The vector median output [2] is defined as the lowest order-statistics $\mathbf{x}_{(1)}$. It is evident that the VMF output is restricted to the dynamic range of the input samples and thus, it can never cause new outliers.

Because the VMF minimizes the aggregated distance to other multichannel samples inside the filter window, it can be equivalently defined by

$$\sum_{j=1}^{N} \|\mathbf{y}_{VMF} - \mathbf{x}_j\|_\gamma \leq \sum_{j=1}^{N} \|\mathbf{x}_i - \mathbf{x}_j\|_\gamma, \text{ for } i = 1, 2, ..., N \tag{3}$$

where $\mathbf{y}_{VMF} = \mathbf{x}_{(1)} \in \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ denotes the VMF output.

3 Proposed Method

Let $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ be an input set determined by a filter window and N represent a window size. Let us assume that $w_1, w_2, ..., w_N$ represent a set of nonnegative integer weights so that each weight w_i, for $i = 1, 2, ..., N$, is associated with the input sample $\mathbf{x}_i$. Thus, the weighted vector distance is defined as follows [13]:

$$J_i = \sum_{j=1}^{N} w_j \|\mathbf{x}_i - \mathbf{x}_j\|_\gamma, \text{ for } i = 1, 2, ..., N \tag{4}$$

The sample $\mathbf{x}_{(1)} \in \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ associated with minimal aggregated weighted distance $J_{(1)} \in \{J_1, J_2, ..., J_N\}$, is the sample that minimizes the sum of weighted vector distances.

The sample $\mathbf{x}_{(1)}$ associated with the minimum aggregated weighted distance $J_{(1)}$ represents the output of the weighted vector median filter (WVMF) [13]. WVMF is also defined by

$$\sum_{j=1}^{N} w_j \|\mathbf{y}_{WVMF} - \mathbf{x}_j\|_\gamma \leq \sum_{j=1}^{N} w_j \|\mathbf{x}_i - \mathbf{x}_j\|_\gamma, \text{ for } i = 1, 2, ..., N \tag{5}$$

It is clear that in the dependence on the weightcoefficients $w_1, w_2, ..., w_N$ the WVMFs can perform a wide range of smoothing operations so that the optimal weight vector may be practically found for each filtering problem.

Because both VMF and WVMF will perform the smoothing operation with the fixed amount of smoothing, these filters can fail in situation with different statistical properties especially caused by a low degree of the observed noise, when they introduce too much smoothing into the image. Note the blurring introduced by a filter may be more objectionable than original noise.

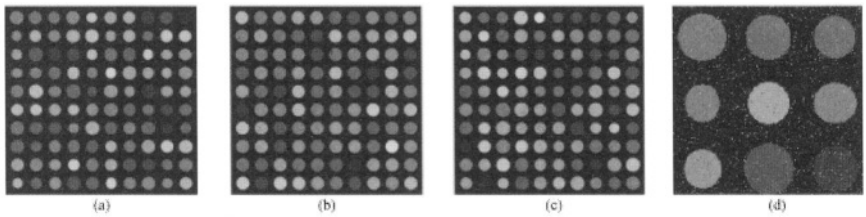


Fig. 2. cDNA phantom (artificial) test images: (a) cDNA1, (b) cDNA2, (c) cDNA3, (d) detailed view on the test image cDNA1 corrupted by 5% impulsive noise.

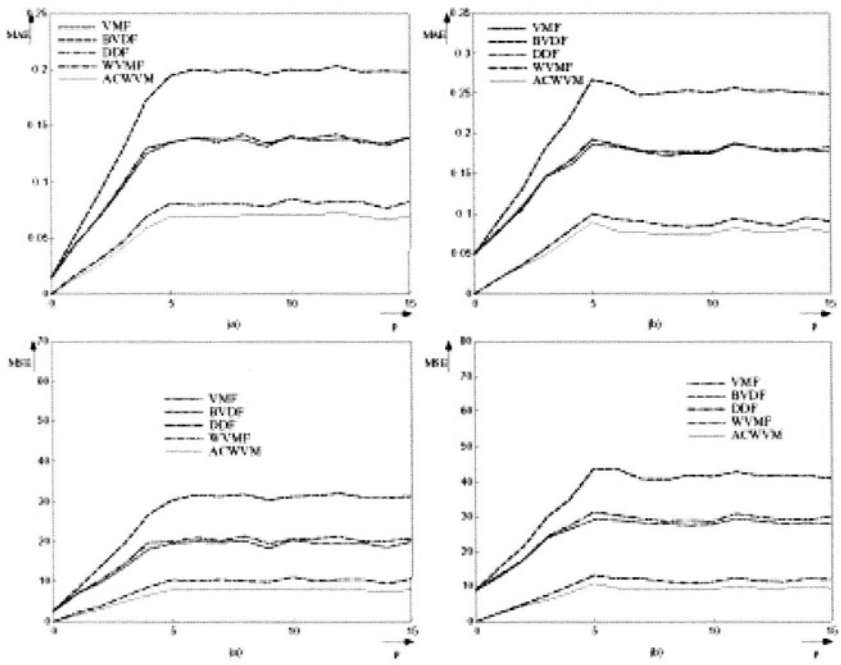


Fig. 3. Dependence of the MAE and MSE criteria on the impulsive noise probability $p_{\mathbf{v}}$: (a) cDNA1 test image, (b) cDNA3 test image.

Now, we provide the adaptive center weighted vector median filter (ACWVM). Let us assume the weight vector given by

$$w_j = \begin{cases} N - 2k + 2 \text{ for} j = (N+1)/2 \\ 1 \qquad\qquad\quad \text{otherwise} \end{cases} \tag{6}$$

i.e. the weight vector of nonnegative integer weights, where only the central weight $w_{(N+1)/2}$ associated with the central sample $\mathbf{x}_{(N+1)/2}$ can be alternated, whereas other weights associated with the neighboring samples remain equal to one. Note that $k = 1, 2, ..., (N+1)/2$ is a smoothing parameter. If the smoothing

parameter k is equal to one, then the center-weighted vector median (CWVM) [8] is equivalent to the identity operation and no smoothing will be provided. In the case of $k = (N + 1)/2$, the maximum amount of the smoothing will be performed and the CWVM filter is equivalent to WVMF. By varying the smoothing parameter k between one and $(N + 1)/2$, it is possible to achieve the best balance between the noise suppression and the signal-detail preservation.

In the case of outliers, bit errors and color artifacts (all above-mentioned events can affect the cDNA microarray images), it is desirable to filter only corrupted samples, whereas the desired image features are invariant to the filtering operation [8]. For that reason, the ACWVM is based on dividing the samples into two classes, namely corrupted samples and noise-free samples. The central sample $\mathbf{x}_{(N+1)/2}$ is examined according to the following rule:

$$\begin{array}{lll} \text{IF } Val \geq Tol & \text{THEN} & \mathbf{x}_{(N+1)/2} \text{ is impulse} \\ & \text{ELSE} & \mathbf{x}_{(N+1)/2} \text{ is noise} - \text{free} \end{array} \tag{7}$$

where *Tol* is the threshold parameter and *Val* is the operation value given by

$$Val = \sum_{k=\lambda}^{\lambda+2} \left\| \mathbf{y}_k - \mathbf{x}_{(N+1)/2} \right\|_{\gamma} \tag{8}$$

where $\mathbf{x}_{(N+1)/2}$ is the central sample of the input set W and $\mathbf{y}_k$ is the output of center-weighted median filter with the smoothing parameter k. Note that the optimal values of *Tol* and λ was found as 80 and 2, respectively. If the operation value *Val* is greater than or equal to the threshold value *Tol,* then the central input sample $\mathbf{x}_{(N+1)/2}$ is probably corrupted and it will be processed by VMF with a robust noise attenuation capability. Otherwise, the central sample $\mathbf{x}_{(N+1)/2}$ is noise-free and it should be invariant against a filtering operation.

Table 1. Results achieved using the phantom images corrupted by 5% impulsive noise.

Image	cDNA1		cDNA2		cDNA3	
Method	MAE	MSE	MAE	MSE	MAE	MSE
Noisy	3.325	525.2	3.231	502.5	3.673	583.2
VMF	0.135	19.4	0.152	21.9	0.186	29.0
BVDF	0.194	30.4	0.224	35.0	0.267	43.7
DDF	0.135	19 9	0.160	23.7	0.193	31.2
WVMF	0.081	10.4	0.085	10.6	0.100	13.1
ACWVM	0.069	8.0	0.070	7.7	0.089	10.9

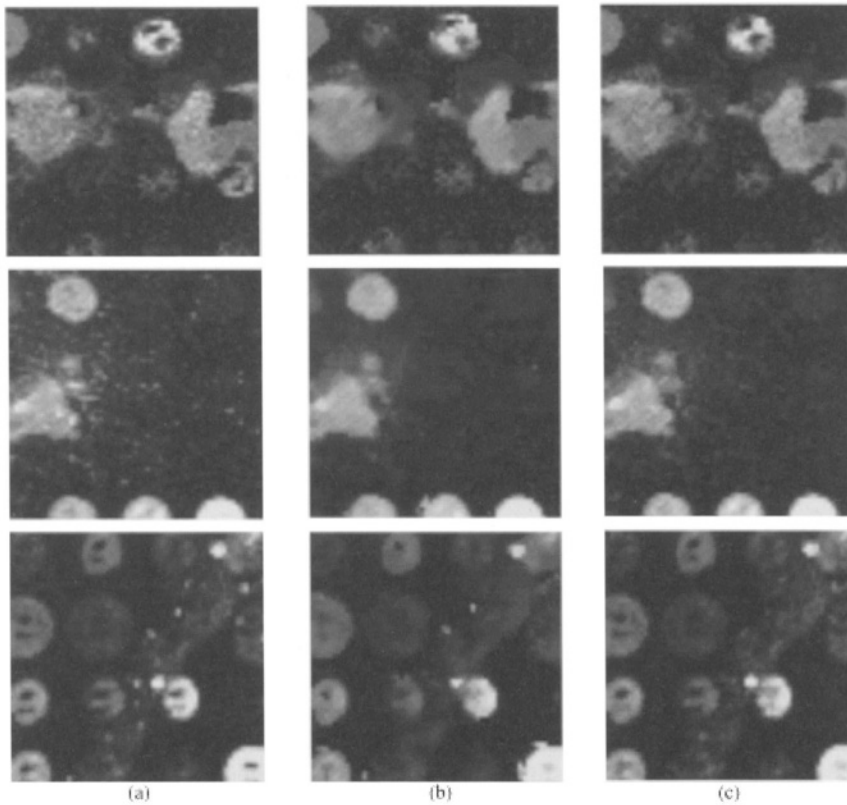


Fig. 4. Enhanced images obtained using real cDNA images: (a) observed image, (b) VMF output, (c) ACWVM output.

4 Experimental Results

The cDNA microarray [3],[6],[14] is a popular and effective method for simultaneous assaying the expression of large numbers of genes and is perfectly suited for the comparison of gene expression in different populations of cells. A cDNA microarray is a collection of spots containing DNA, deposited on the surface of a glass slide. Each of the spots contains multiple copies of a single DNA sequence. The spots occupy a small fraction of the image area and they have to be individually located and isolated from the image background prior to the estimation of its mean intensity. The fluorescent intensities for each of the two dyes are measured separately, producing a two channel image. The image is false colored using red and green for each image components, which represent the light intensity emitted by the two fluorescent dyes. So the microarray images look like a collection of green, red and yellow spots of different hue, saturation and intensity. The intensities provided by the array image can be quantified by measuring the

average or integrated intensities of the spots. The major sources of uncertainty in spot finding and measuring the gene expression are variable spot sizes and positions, variation of the image background and various image artifacts. Spots vary significantly in size and position within their vignettes despite the use of precise robotic tools to lay them out onto the slide. Additionally the natural fluorescence of the glass slide and non-specifically bounded DNA or dye molecules add a substantial noise floor to the microarray image along with discrete image artifacts such as highly fluorescent dust particles, unattached dye, salt deposits from evaporated solvents, fibers and various airborne debris. So, the task of microarray image enhancement and is of great importance, as it enables correct block and spot segmentation.

In order to compare the performance of used filtering schemes, we used some artificial images (Fig.2a-c) degraded by impulsive noise (Fig.2d) and also the natural microarray images shown in Fig.1. By using the artificial images, the undistorted, original, images are available and the obtained results can be evaluated objectively (Tab.1 and Fig.3) via the mean absolute error (MAE) and the mean square error. In the case of the natural chip images depicted in Fig.1, we can compare only the subjective results, in the form of visual assessment of the filter outputs depicted in Fig. 4.

As can be seen the proposed method significantly outperforms the commonly used multichannel image processing noise reduction techniques such as VMF [2], basic vector directional filter (BVDF) [12], directional distance filter (DDF) [7] and non-adaptive WVMF [13] with weight coefficients [1, 2,1,4, 5,4,1, 2,1]. This is also confirmed by its performance on real images shown in Fig.4, in which the noise component was successfully removed, while preserving the texture and sharpness of the spot edges.

5 Conclusion

In this work a novel algorithm of the noise reduction in microarray chip images has been presented. During the filtering process the outliers affected the spots are being detected and removed while the edges remain well preserved. The proposed technique can serve as an efficient low-processing tool for microarray image enhancement, which can enable better spots localization and the estimation of their intensity.

Future research will focus on the segmentation and the spot localization, where contour smoothing algorithms of [4], [5] can play a key role. Combined with the enhancement method introduced in this paper it is expected that image segmentation techniques will bring another improvement on the cDNA microrarray image analysis.

References

[1] Ajay, N., Tokuyasu, T., Snijders, A., Segraves, R., Albertson, D., Pinkel, D.: Fully automatic quantification of microarray image Data. Genome Research **12** (2002) 325–332

[2] Astola, J., Haavisto, P., Neuvo, Y.: Vector median filters. Proceedings of the IEEE **78** (1990) 678–689
[3] Dopazo, J.: Microarray data processing and analysis. In Microarray Data Analysis II, ed. SM Lin and KF Johnson, Kluwer Academic, 2002, 43–63
[4] Glasa, J.: On derivatives estimation of smoothed digital curves. Computer and Artificial Intelligence **19** (2000) 235–249
[5] Glasa, J., Podhajecky, G.: On feasible digital contour smoothing for biomedical image analysis. Proc. Embec'02, (2002) 840–841
[6] Hsiao, L., Jensen, R., Yoshida, T., Clark, K., Blumenstock, J., Gullans, S.: Correcting for signal saturation errors in the analysis of microarray data. Biotechniques **32** (2002) 330–336
[7] Karakos, D.G., Trahanias, P.E.: Generalized multichannel image-filtering structure. IEEE Trans. Image Processing **6** (1997) 1038–1045
[8] Lukac, R.: Vector LUM smoothers as impulse detector for color images. Proc. ECCTD '01 **3** (2001) 137–140
[9] Pitas, I., Venetsanopoulos, A.N.: Order statistics in digital image processing. Proceedings of the IEEE **80** (1992) 1892–1919
[10] Plataniotis, K.N., Venetsanopoulos, A.N.: Color image processing and applications. Springer Verlag (2000)
[11] Schena, M., Shalon D., Davis, R.W., Brown, P.O.: Quantitative monitoring of gene expression patterns with a complimentary DNA microarray. Science **270** (1995) 467–470
[12] Trahanias, P.E., Karakos, D., Venetsanopoulos, A.N.: Directional processing of color images: theory and experimental results. IEEE Trans. Image Processing **5** (1996) 868–881
[13] Viero, T., Oistamo, K., Neuvo, Y.: Three-dimensional median related filters for color image sequence filtering, IEEE Trans. Circ. Syst. Video Tech **4** (1994) 129–142.
[14] Yang, Y., Buckley, M., Dudoit, S., Speed, T.: Comparison of methods for image analysis on cDNA microarray data. Jour. Computational and Graphic Statistics **11** (2002) 108–136

Implementation Aspects of a Recovery-Based Error Estimator in Finite Element Analysis

Arkadiusz Nagórka and Norbert Sczygiol

Częstochowa University of Technology,
Institute of Computer and Information Sciences,
Dąbrowskiego 73, 42–200 Częstochowa, Poland
{nagorka, sczygiol}@icis.pcz.pl
http://icis.pcz.pl

Abstract. The paper is devoted to the use of error estimators based on gradient recovery in finite element computations, where the resulting error estimates can be used as the basis for *hp*-adaptive mesh refinement. Due to high complexity of adaptive numerical software we have decided to take advantage of the object-oriented paradigm of software development. We discuss our implementation of the Zienkiewicz-Zhu error estimator and of selected gradient recovery techniques (averaging and superconvergent patch recovery).

1 Introduction

Approximate solutions of problems described by partial differential equations, obtained e.g. by means of the finite element method (FEM), are inaccurate. The error of the solution depends on discretization parameters such as mesh density and interpolation order. However, the finer the discretization is, the more degrees of freedom (unknowns) are to be determined and the simulations are more costly in terms of computation time and memory requirements. Often unacceptably high cost of computations can be substantially reduced by introducing *adaptivity* to the method, so that the mesh is refined only in places that actually need it. The decision on whether the refinement is needed or not is best based upon accuracy of the approximate solution at hand in form of a posteriori *error estimates.* Hence the error estimation is crucial part of adaptive finite element software.

Users of finite element software are often more interested in derivatives of the solution (heat flux, strain, stress) than in the solution (temperature, displacement) itself. Unfortunately, these quantities, being the functions of the approximate gradient, are less accurate than the solution due to differentiation of polynomial approximation. Moreover, the normal component of the gradient is generally discontinuous across element boundaries. For this reason it is often postprocessed (smoothed) before being presented to the user. It has been observed that under some circumstances the smoothed gradient is superior to the original one in the sense of much better agreement with the unknown exact derivatives. Natural approach is to utilize the difference between the smoothed

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 722–729, 2004.
© Springer-Verlag Berlin Heidelberg 2004

(recovered) derivatives and the original derivatives as a measure of discretization error, which has led to recovery-based error estimation techniques. According to many authors, error estimators of this kind seem to be the most robust [1] and their use is advocated in practical computations.

In subsequent sections a recovery-based error estimator along with selected gradient recovery techniques, which are implemented in our in-house adaptive finite element code, are presented with emphasis put on implementation aspects.

2 The Zienkiewicz-Zhu Error Estimator and Gradient Recovery Techniques

We are interested in the error of approximate solution measured in the *energy norm.* For the heat conduction problem the norm of the temperature error $e = T - T_{hp}$, where T_{hp} is a FEM approximation, can be written as

$$\|e\|_E^2 = \int_\Omega \lambda^{-1} \left(\boldsymbol{q} - \boldsymbol{q}_{hp}\right) \cdot \left(\boldsymbol{q} - \boldsymbol{q}_{hp}\right) \, dx \, dy, \tag{1}$$

where $\boldsymbol{q} = -\lambda \nabla T$ is the heat flux. For the linear elasticity problem, where displacement $\boldsymbol{u}$ is the primary unknown and the stress $\boldsymbol{\sigma} = \frac{1}{2} \boldsymbol{C} \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right)$, the energy norm of the error reads

$$\|\boldsymbol{e}\|_E^2 = \int_\Omega \left(\boldsymbol{\sigma} - \boldsymbol{\sigma}_{hp}\right) \boldsymbol{C}^{-1} \left(\boldsymbol{\sigma} - \boldsymbol{\sigma}_{hp}\right) \, dx \, dy. \tag{2}$$

In general the error norm squared is the integral of a function of $\boldsymbol{g} - \boldsymbol{g}_{hp}$, where $\boldsymbol{g}$ is a quantity related to the gradient of the solution, such as $\boldsymbol{q}$ or $\boldsymbol{\sigma}$, and in which the components can be weighted by some material parameters, such as λ or $\boldsymbol{C}^{-1}$ for instance.

Although the approximate gradient $\boldsymbol{g}_{hp}$ is discontinuous across element boundaries, there exist some points inside finite elements where it is exceptionally accurate *(superconvergent)* [2]. These superior values can be used to reconstruct (recover) the continuous field $\boldsymbol{g}^*$, which often turns out to be very close to the unknown $\boldsymbol{g}$. Midpoints of one-dimensional or barycenters of triangular first-order elements are examples of such optimal points. The recovered gradient $\boldsymbol{g}^*$ can be substituted for $\boldsymbol{g}$ in (1) or in (2), which results in the *Zienkiewicz-Zhu* error estimator [3].

The recovered gradient is interpolated in the element interior using element shape functions and stored nodal values. The latter ones are determined using one of the following recovery techniques:

1. Averaging – a method that gives the best results on meshes consisting of elements with linear interpolation – where values of the original gradient are sampled in some points (nodes, adjacent Gauss points) and a possibly weighted average is taken, e.g.

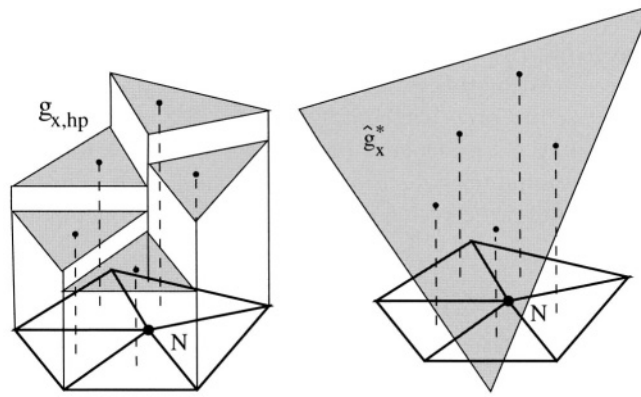


Fig. 1. Superconvergent recovery of a component of the gradient on a patch of elements

- nodal average (in the node N)

$$\boldsymbol{g}^*(\boldsymbol{x}_N) = \frac{1}{N_e}\sum_{e=1}^{N_e} \boldsymbol{g}_{hp}^{(e)}(\boldsymbol{x}_N), \tag{3}$$

- Gauss point or optimal point average

$$\boldsymbol{g}^*(\boldsymbol{x}_N) = \sum_{e=1}^{N_e} \frac{\boldsymbol{g}_{hp}^{(e)}\left(\boldsymbol{x}_c^{(e)}\right)}{d_{c,N}^{(e)}} \Big/ \sum_{e=1}^{N_e} \frac{1}{d_{c,N}^{(e)}}, \tag{4}$$

where $d_{c,N}^{(e)}$ is the distance from the node N to the barycenter of the element e.

2. Superconvergent patch recovery (SPR) [4] where the least-squares method is used to fit a higher-order polynomial $\hat{\boldsymbol{g}}^*(\boldsymbol{x}) = \boldsymbol{P}(\boldsymbol{x})\,\boldsymbol{a}$, where $\boldsymbol{P}$ is a matrix of monomial terms and $\boldsymbol{a}$ is the vector of coefficients, to the values of the original gradient $\boldsymbol{g}_{hp}$ sampled in optimal points located inside a patch of elements that share the given node N (Fig. 1).
The polynomial $\hat{\boldsymbol{g}}^*$ is then evaluated in all nodes of the patch so that the recovered gradient can be interpolated inside each element using its shape functions.
The local problem on a patch reads: find $\boldsymbol{a}$ minimizing

$$\Pi(\boldsymbol{a}) = \sum_{e=1}^{N_e}\Bigg\{ \sum_{p=1}^{n_p(e)} \left(\boldsymbol{P}\boldsymbol{a} - \boldsymbol{g}_{hp}\left(\boldsymbol{x}_p\right)\right)\cdot\left(\boldsymbol{P}\boldsymbol{a} - \boldsymbol{g}_{hp}\left(\boldsymbol{x}_p\right)\right) + \phi_r \left\|r^*\right\|^2_{L_2(\Omega^e)} + \phi_R \left\|R^*\right\|^2_{L_2(\partial\Omega^e\cap\Gamma_N)} \Bigg\} \tag{5}$$

where $n_p(e)$ is the number of sampling points in an element and $\boldsymbol{x}_p$ denotes the coordinates of the sampling point. The quantities r^* and R^* are the

interior and boundary residuals arising from the fact that the recovered gradient fails to satisfy the PDE being solved and natural boundary conditions. Minimization of (6) yields a system of linear algebraic equations which has to be solved for $\boldsymbol{a}$.

3 An Object Model of Recovery-Based Error Estimation

Adaptivity in FEM involves interaction between a posteriori error estimation, mesh refinement, iterative solution etc. and introduces considerable complexity into numerical software in terms of both code size and sophisticated data structures. In consequence, efficient implementation of such codes becomes a challenge. Moreover, the bulk of adaptive finite element programs is application independent. In order to cope with software complexity and to facilitate code reuse we decided to take advantage of the object-oriented paradigm of software development. Reusable code has been put into a C++ class library which makes it easier to develop new programs (solvers) and provides the user with finite elements of various shapes and basis functions, numerical integration, equation solvers, meshes with refinement support, error estimation, file I/O etc.

The simplified UML diagram of classes involved in recovery-based error estimation is outlined in Fig. 2. The `ZZ_ErrorEstimator` class is designed to be decoupled from gradient recovery techniques and from specific application area. It uses separate objects responsible for interpolation of finite element gradient (a subclass of `FeGradField`), for gradient recovery (`AveragedGradField` or a subclass of `SPRGradField`) and for the particular problem definition (`EnergyNormIntegrand` or its descendant).

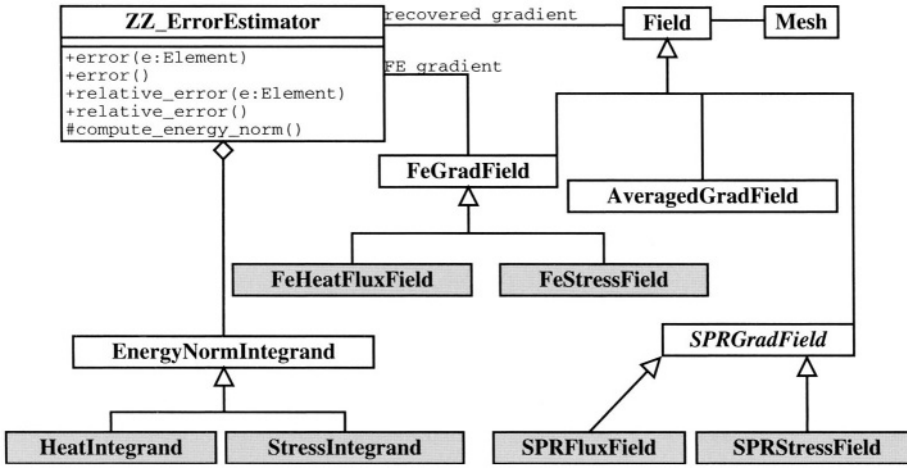


Fig. 2. An object model of recovery-based error estimation

In this and subsequent diagrams boxes with gray background denote classes specific to the problem being solved (heat transfer, elasticity, diffusion, electrostatics etc.) and having usually trivial implementation, whereas the others represent library classes. For instance, the `EnergyNormIntegrand` class defaults to $(\boldsymbol{g}^* - \boldsymbol{g}_{hp}) \cdot (\boldsymbol{g}^* - \boldsymbol{g}_{hp})$ but it is possible to tailor the behaviour to the problem at hand, e.g. for Eq. (2) in the `StressIntegrand` class where the components are multiplied by elements of the $\boldsymbol{C}^{-1}$ material tensor.

The `ZZ_ErrorEstimator` class provides the user with methods computing global or local error which is either absolute or relative to the norm of the solution. The helper function `compute_energy_norm` takes care of all the details concerning automatic selection of quadrature rule for numerical integration. The decision is based on actual polynomial degree of the element at hand and the information is fetched using accessor functions in library classes.

The constructor of the `ZZ_ErrorEstimator` class takes objects representing $\boldsymbol{g}^*$ and $\boldsymbol{g}_{hp}$ as arguments. Example use of the class is presented in the code excerpt below

```
FeHeatFluxField q_hp(mesh);
SPRFluxField q_rec(mesh, q_hp, 0.0, 0.0, false);
q_rec.do_recovery();
// AveragedGradField q_rec(mesh, q_hp, SC_POINTS);
// q_rec.do_averaging();
HeatIntegrand integrand;
ZZ_ErrorEstimator estimator(q_rec, q_hp, integrand);
for (int i = 0; i < NE; i++) cout << estimator.error(i) << '\n';
```

The design outlined in Fig. 2, although overly complicated at first glance, is very flexible. It is possible to estimate the error on meshes of elements of various types, even unknown at the time of compilation of the library, to add new gradient recovery techniques and to account for the specific application details. The latter is done by merely defining a few derived classes with usually trivial implementation.

Figure 3 depicts the object model of gradient recovery using averaging techniques. In the diagram only class members that are relevant to gradient recovery are shown. The work is being done mainly by the `AveragedGradField` class, which can perform nodal, point or weighted point averaging. After the recovery nodal values of $\boldsymbol{g}^*$ are stored for quick access and for later use during interpolation of the recovered gradient inside elements, executed by the `get_value` method.

Details of the problem at hand are hidden in an associated application-dependent (supplied by the programmer) descendant of this class, for example in the `FeStressField` class that computes the components of the stress tensor in a quite complicated way.

The superconvergent patch recovery of the gradient is implemented by the `SPRGradField` and `SPRElementPatch` classes (enclosed by the dashed rectangle in the diagram shown in Fig. 4). These classes are abstract and some functionality

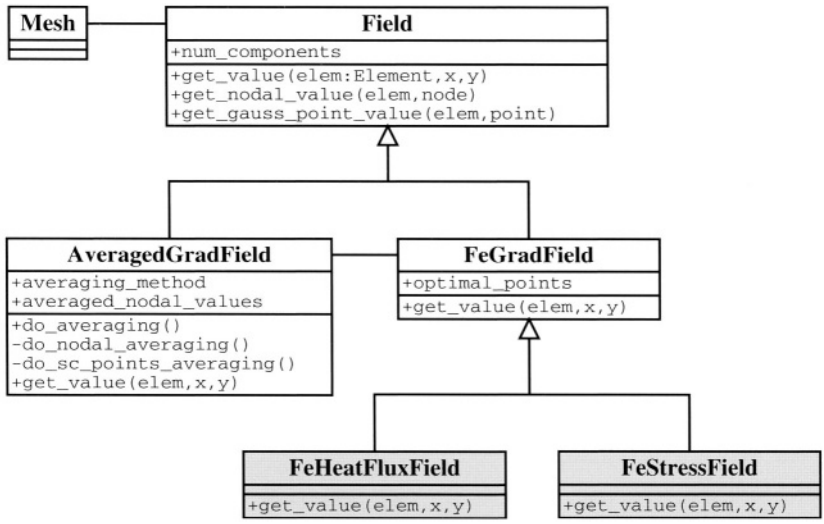


Fig. 3. Relationships between classes involved in averaging of the approximate gradient

must be provided by application-specific subclasses, e.g. by `SPRFluxPatch` and `SPRFluxField` for heat transfer problems (implementation of the latter is trivial).

The `SPRElementPatch` class is responsible for all of the work related to the recovery process illustrated in Fig. 1. It fits a higher-order polynomial to gradient values sampled in optimal points, whose locations for individual elements on the patch are fetched from the associated `FeGradField`. It is generally assumed in this implementation that the optimal points coincide with appropriate Gauss integration points [2,5], with a few exceptions for lower-order triangles.

Application-specific patch objects are created in the `create_patch` factory method defined in a corresponding subclass of `SPRGradField`, e.g. in `SPRFluxField` for `SPRFluxPatch`. The recovery process is performed on the patch associated with the node whose index is given during construction of the object. The method `move_to_node` lets one associate the patch with another node without the need for costly destruction and creation of the object.

The fitting process starts with determination of the degree of polynomial representation of the recovered gradient on the patch. The method `determine_degree` defaults to evaluation of number of optimal points on the patch. The sum is compared to the number of coefficients needed to represent a complete polynomial of degree `e.PolynomialDegree()` (or maximal degree in case of a patch containing elements of varying order). If the number is smaller the degree is decremented successively.

Next a system of linear equations for the polynomial coefficients is built in the `solve_local_problem` method of `SPRElementPatch`. The system matrix and right hand side vector consist of various terms and are built using the

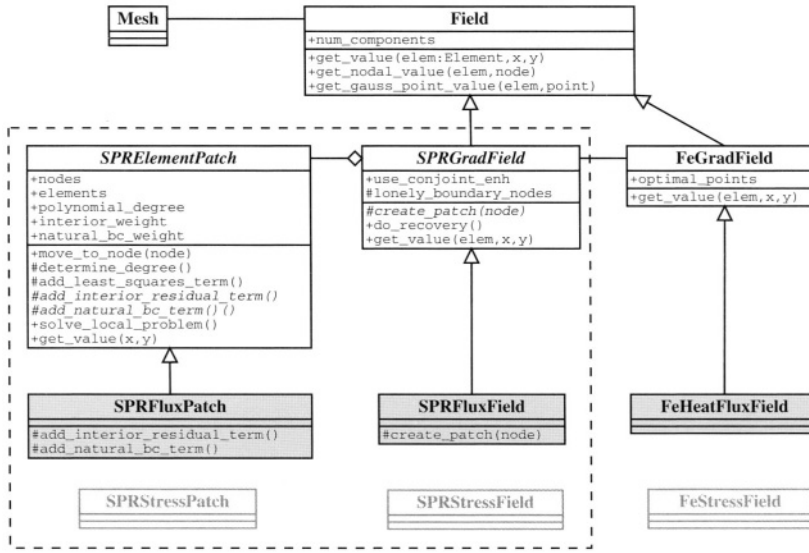


Fig. 4. Relationships between classes involved in superconvergent patch recovery

`add_least_squares_term`, `add_interior_residual_term` and `add_natural_bc_term` methods, respectively. These methods correspond to the components of the sum in (6). While the least squares term is application independent and is implemented in `SPRElementPatch`, the other terms are not. Hence the need for a problem-specific derived class that provides the implementation of the other two abstract methods, e.g. `SPRFluxPatch`. It is also possible to turn off accounting for residual terms by setting `interior_weight` and `natural_bc_weight` members to zero, which leads to decoupled systems of equations for individual components of the gradient.

The linear system is solved using the LU factorization. The polynomial coefficients obtained as the result can be used for local interpolation on the patch in the `SPRElementPatch::get_value` member function. This method is used by the `SPRGradField` to evaluate the polynomial $\hat{\boldsymbol{g}}^*$ in patch nodes. The nodal values are stored and can be later used for interpolation of the recovered gradient in the interior of an element provided by the `SPRGradField::get_value` method.

The problem of interpolation of the recovered gradient $\boldsymbol{g}^*$ inside individual higher-order finite elements is complicated if degrees of freedom have another meaning than just values of the interpolated quantity in the nodes. For example, the use of hierarchical basis functions would require the solution of local interpolation problems. In our implementation a much more efficient approach is used that involves storing values of $\boldsymbol{g}^*$ sampled in interpolation points and employing usual higher-order Lagrange shape functions. Care must be taken for interpolation points shared between adjacent elements. Associative containers are used with point coordinates (ordered with a tolerance) as the key and sampled gradi-

ents as the data. Alternative approach would involve generation of a temporary mesh of multi-node Lagrange-type finite elements and utilize it directly.

For meshes of linear elements the so-called conjoint interpolant enhancements [6] can yield even better results. The `SPRGradField: :get_value` method can use this technique if required.

It might happen that a boundary element patch has insufficient number of sampling points to fit even a linear polynomial. In such cases the recovery process is abandoned. The nodal value has already been or will be evaluated from adjacent patches. Once existence of such a node is detected its index is stored in the `lonely_boundary_nodes` set owned by the `SPRGradField`.

4 Conclusion

Object-oriented paradigm of software development allowed us to cope with increasing complexity of finite element programs. A finite element class library has been created which makes it easier to develop new solvers for specific applications. The Zienkiewicz-Zhu error estimator and derivative recovery techniques serve as the basis for our implementation of algorithms of adaptive mesh refinement and selection of approximation order discussed in [7].

Acknowledgments. This work is sponsored by the Polish State Committee for Scientific Research under Grant No. 4 T08B 032 24.

References

1. I. Babuška. Validation of a posteriori error estimators by numerical approach. *Int. J. Num. Meth. Engng.,* 37:1073–1123, 1994.
2. J. Barlow. Optimal stress locations in finite element models. *Int. J. Num. Meth. Engng.,* 10:243–251, 1976.
3. O.C. Zienkiewicz and J.Z. Zhu. A simple error estimator and adaptive procedure for practical engineering analysis. *Int. J. Num. Meth. Engng.,* 24:337–357, 1987.
4. O.C. Zienkiewicz and J.Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. Part I. The recovery technique. *Int. J. Num. Meth. Engng.,* 33:1331–1364, 1992.
5. R.J. Mackinnon and G.F. Carey. Superconvergent derivatives: a Taylor series analysis. *Int. J. Num. Meth. Engng.,* 28:489–509, 1989.
6. T. Blacker and T. Belytschko. Superconvergent patch recovery with equilibrium and conjoint interpolant enhancements. *Int. J. Num. Meth. Engng.,* 37:517–536, 1994.
7. N. Sczygiol and A. Nagórka. An *hp*-adaptive finite element method for the solidification problem based on explicit residual error estimates. In *Proceedings of Computer Methods in Mechanics,* Wisla, Poland, 3–6 June, 2003.

Optimization Using Nimrod/O and Its Application to Robust Mechanical Design

Tom Peachey[1], David Abramson[1], Andrew Lewis[2], Donny Kurniawan[1], and Rhys Jones[3]

[1] School of Computer Science and Software Engineering, Monash University, Melbourne, Australia
[2] Div. of Information Services, Griffith University, Brisbane, Australia
[3] Department of Mechanical Engineering, Monash University, Melbourne, Australia
tcp@csse.monash.edu.au

Abstract. We describe the Nimrod/O distributed optimization tool and its application to a problem in mechanical design. The problem is to determine the shape for a hole in a thin plate under load that gives optimal life in the presence of flaws. The experiment reveals two distinct design strategies for optimizing this life. Nimrod/O is able to find both of these rapidly due to its inherent parallelism.

1 Nimrod/O

Advances in computing science and engineering modeling have enabled design engineers to use computational models instead of real world prototypes in many situations. Such an approach is usually faster and cheaper and hence allows the user to explore various design scenarios. In particular the user may search through combinations of design parameters in order to achieve a design that is optimal in some sense. Similarly, scientific research is increasingly using computational models. Often the aim is to determine those model parameters that produce the best fit to real world data, so again the model is used repeatedly to achieve some optimum.

Such models typically require computational times of minutes or hours, even on a high end processor. We have produced a number of tools that facilitate concurrent execution of these models. The tools Nimrod and EnFuzion [1] allow the user to specify a range of values for each design parameter; then all combinations of these values are generated and the resulting jobs are farmed to a cluster of processors. Nimrod/G [2] is a version of Nimrod that utilises resources on a global computational grid. These tools may be used for design optimization by exploring all possibilities in a search space. A more efficient search is provided by Nimrod/O [3], [4], [5] which offers a variety of standard optimization procedures to explore the parameter space.

When a Nimrod/O procedure requires values of the objective function it passes the parameters to Nimrod or EnFuzion for evaluation against the computational model. The model is run on an appropriate platform and the objective

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 730–737, 2004.
© Springer-Verlag Berlin Heidelberg 2004

function value is extracted from the model output. A cache is superimposed between Nimrod/O and the back-end to reduce the number of calculations required if the same parameter values are requested more than once. A persistent database is attached to the cache to support restart if Nimrod/O is terminated prematurely. The Nimrod/O implementation of the optimization procedures uses concurrent evaluations where possible. In addition it allows separate optimizations to be run concurrently.

To perform a Nimrod/O optimization the user needs to prepare a simple text "schedule" file. This specifies the parameters of the search space, any constraints imposed on the solution, how to run the computational model to obtain objective values and the optimization method(s) to apply. Assuming the model correctly executes on the remote nodes, no further development is required.

2 Computation of Fatigue Life for Shape Optimization

In this paper we discuss the application of Nimrod/O to the optimal design of mechanical components. The failure of such components is usually due to slow growth of a pre-existing crack followed by a sudden fracture [6]. A common mechanism for crack growth is fatigue due to cyclical loading. As a load is applied the high stress at the crack tip causes plastic deformation which produces an irreversible growth in the crack. This growth is small (typically $10^{-7} - 10^{-3}$ mm) but repeated cycles of loading may extend the crack to a stage where fracture occurs. The problem is very important industrially, because failures of large structures like trains, automobiles, oil drilling rigs and mining equipment can cost millions of dollars, and cause death and injury to personnel.

The computational model that we used is described in [7]. In this approach, finite element techniques are used to compute the stress field throughout the component for a given applied load in the absence of cracks. Then cracks are assumed to occur at critical boundaries of the component and a recent modification [8] of the finite element alternating method [9] is applied to compute the stress intensity factor at the crack positions. Finally the Paris growth law [10] is used to compute the number of loading cycles required for the cracks to grow from its given initial size to a given final size. The number of cycles at the worst (least cycles) crack is taken as the fatigue life. Because a fine numerical mesh is required at the crack tips, computation times are typically of the order of minutes even for small two-dimensional models.

For the purposes of the stress analysis and subsequent fracture mechanics the shape profile is defined by control points. The finite element analysis proceeds by fitting a spline through the points to generate further boundary points. For optimization purposes the control points may be considered the state variables in the search space. For example if a curve in two-dimensions is specified by 50 movable points then the optimization has 100 independent variables. Optimization in a space of such high dimension is notoriously difficult. A further difficulty is how to impose desired constraints; especially difficult are constraints on the slope of the boundary.

Despite these difficulties some success has been achieved using a "biological algorithm" which moves each control point normal to the boundary a distance based on the assessment of that point. Earlier work assessed points on the basis of stress [11] or stress intensity factor [12],[13],[14]; more recently [7] fatigue life has been used.

An alternative approach, which is proposed in this paper, is to use a family of curves or surfaces specified by some parameters and to generate the control points from the curve. Finding the optimum shape involves determining the best parameters. Since the number of parameters is small, optimization is more efficient. However for this approach to find the true optimum the family of curves must include that optimum. Construction of a suitable family of curves can be a difficult design problem.

3 The Test Problem

The test problem is an aluminium alloy plate of thickness 1.5mm with dimensions 320mm by 600mm as shown in Figure 1. A remote stress field of 100MPa is applied in the vertical direction. The plate contains a hole of width 20mm which is 10mm from the left edge. As the geometry is symmetrical about the horizontal mid plane only the top half is modeled. This simple test problem is characteristic of a number of real designs found in practice. For example, the plate between the rear seats and the trunk of a passenger car, typically has a section removed to allow long items to project from the trunk into the cabin. In general, the hole needs to be as large as possible without compromising the mechanical stability of the plate, which is an important part of the structure of the car. In another setting, the same basic design can be found in the fuel tanks of high performance jet fighters. In this case a mechanical stiffener is placed in the fuel tank, which is in turn held in the wing, to increase the stability of the wing structure. However, a hole must be placed in the stiffener to allow fuel to move freely. Again, the hole needs to be large enough to allow fuel to move, but not too large that the effect of the stiffener is compromised.

The aim of the experiment is to determine the hole shape for optimal fatigue life in the presence of cracks distributed around the hole profile. The cracks have an initial length of 1mm and are deemed to have failed when they reach a length of 5mm.

4 Parameterization

Theory predicts that optimal hole profile must be free of sharp corners and symmetric about the horizontal mid plane. A suitable family of smooth curves, symmetrical about the point (p, q), are the "generalized ellipses",

$$\frac{|x-p|^t}{a^t} + \frac{|y-q|^t}{b^t} = 1 \tag{1}$$

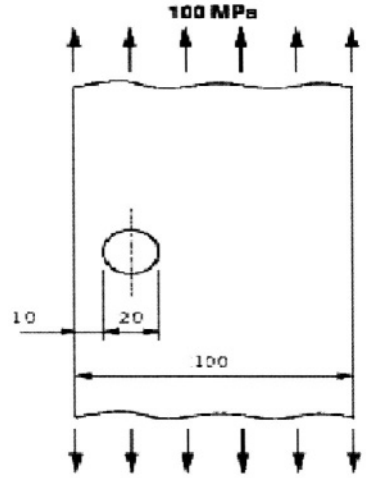


Fig. 1. Test problem for hole profile optimization

These are closed curves of width $2a$ height $2b$ where t controls the curvature at the shoulders of the curves. We will call this the "tightness". Figure 2a shows some of these with $a = 10$, $b = 20$ and a variety of values of t. More generally, the curves

$$\frac{|x-p|^t \exp(-t\beta(y-q))}{a^t} + \frac{|y-q|^t}{b^t} = 1 \tag{2}$$

also have width $2a$ and height $2b$ at the vertical axis. The inclusion of the exponential factor allows for asymmetry about that axis. The parameter β, called the "bias" here, controls the slope at the top. Figure 2b shows samples with $p = 0$, $q = 0$, $a = 10$, $b = 20$, $t = 5$ and various β.

For the plate model we employed $a = 10$, $p = 20$, and $q = 0$ giving a hole of width 20mm which is 10mm from the left boundary. Thus the variables b, t and β are the optimization parameters. The search space used was the domain $5 \leq b \leq 35$, $2 \leq t \leq 9$, $-0.02 \leq \beta \leq 0.02$.

5 Results

5.1 Experiment 1

Experiments were performed on a Compaq SC cluster, consisting of 20 Alpha EV68 processors connected by a Quadrics switch. Nine independent optimizations were performed starting from various initial parameters evenly spread through the search space. Each optimization used the simplex search algorithm [15]. The results are shown in Table 1.

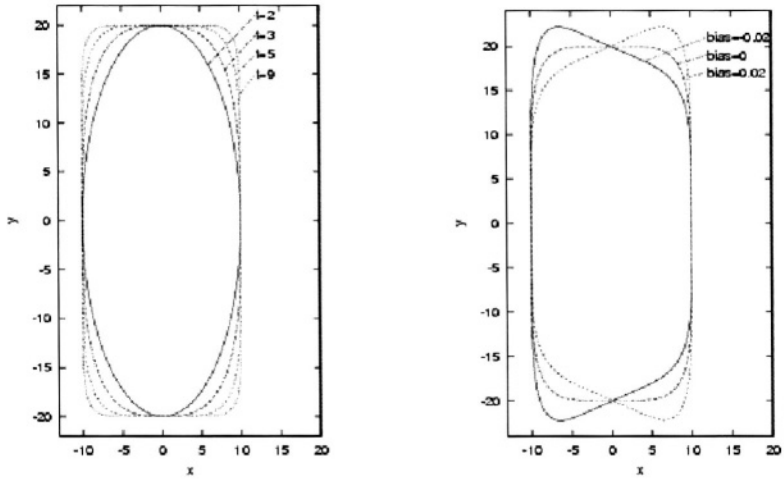


Fig. 2. Parametrized families (a) Equation (1), (b) Equation (2).

The "parameters" referred are the triples (b, t, β). As well as the final parameters and the optimum achieved, Nimrod/O reports statistics on the search process. The "Itrs" column gives the number of iterations of the simplex search. The "Evals" column contains two numbers; the second (in parentheses) is the number of evaluations required by the algorithm. However some evaluations may be duplicates, in which case the result is obtained from a cache rather than recomputed. The first number gives the number of unique evaluations. Since jobs are computed concurrently in batches, the number of batches gives a better indication of the running time of an optimization. The "Batches" column gives the number of batches requested by the algorithm and the number actually calculated. Because all nine optimizations were performed in parallel and batches averaged about 4 jobs, the concurrency was approximately 36 when compared with a serial optimizer performing the same searches.

Eight of the searches achieved an optimum with b at or close to the upper limit of 35. This agrees with previous work [13] on this problem using a control point algorithm that found the height of the hole grows to its upper bound. However one search (the first) revealed a local optimum in another part of the search space.

5.2 Experiment 2

So as to obtain a better understanding of the search space we used Nimrod to survey the search space evaluating the life at all combinations of $b = 5, 35, (5)$, $t = 2, 9, (0.5)$, $\beta = -0.020, 0.020, (0.002)$. Maximum stress on the hole profile was also computed in each case.

For each of these values of the bias β, rubber sheet representations were produced for both the life and the stress as functions of b and t. Further, isosurface

Table 1. Results of 9 simplex searches.

Starting Parameters	Optimal Parameters	Optim. Life	Itrs.	Evals.	Batches
(10,2,0)	(8.72,2.000,-0.00032)	3719	23	88 (104)	26 (26)
(10,5.5,0)	(35.00,4.390,-0.01687)	5335	19	76 (76)	19 (19)
(10,9,0)	(35.00,8.986,-0.00104)	5341	17	65 (68)	17 (17)
(20,2,0)	(33.86,4.078,-0.01904)	5361	18	70 (72)	18 (18)
(20,5.5,0)	(33.89,4.523,-0.01569)	5282	11	44 (44)	11 (11)
(20,9,0)	(35.00,7.963, 0.00148)	5316	12	48 (48)	12 (12)
(30,2,0)	(35.00,8.780, 0.00047)	5342	16	62 (64)	16 (16)
(30,5.5,0)	(35.00,8.300, 0.00000)	5342	27	31 (32)	8 (8)
(30,9,0)	(35.00,8.767, 0.00267)	5339	8	20 (20)	5 (5)

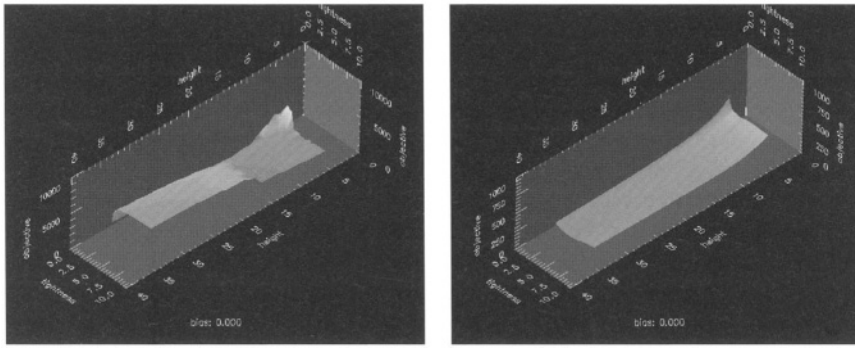


Fig. 3. Rubber sheet representations for (a) life (b) stress against b and t for $\beta = 0$.

representations were produced for life and stress in terms of the three variable (b, t, β). These pictures are available at [16]. They show that the bias has only a minor effect on stress and life. For the case $\beta = 0$, the rubber sheets for life and stress are shown in Figures 3a and b respectively.

Figure 3a shows two regions of the search space that give high values for life. One is around $b = 35.0$, $t = 9.0$ (we call this region 1) and the other near $b = 5.0$, $t = 2.0$ (region 2). This explains why Nimrod/O searches found two distinct local optima. The fact that these optimal regions are on the search space boundaries suggest that extending the boundaries will allow better optima.

We expect that shapes with long life will have low stress. Figure 3b shows that this is true for region 1 but not for region 2. The latter case seems anomalous in that maximum life corresponds to maximum stress. This matter is resolved by visualizations of the stress field for shapes taken from these regions. Figure 4a shows the hole shape and associated stress field for the shape $b = 35.0$, $t = 4.0$ and $\beta = -0.020$, which gave the global optimum in region 1, and Figure 4b for shape $b = 5.0$, $t = 2.0$ and $\beta = -0.004$, the local optimum in region 2. The latter reveals that, although the shape gives high stress on the hole boundary, the stress

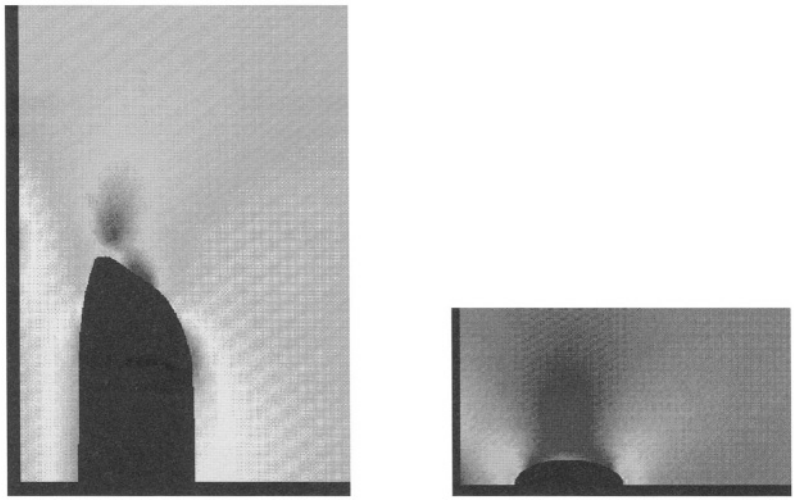


Fig. 4. Stress fields for hole shape (a) globally optimal (b) locally optimal.

falls off rapidly when moving into the body. Hence cracks on this boundary will initially grow quickly but this is balanced by very slow subsequent growth.

6 Conclusion

For the test geometry both long angular shapes and short rounded shapes produced high fatigue life, but only the former corresponds to low stress on the hole boundary. These results confirm that optimization for fatigue life may produce very different results from traditional optimization based on boundary stress. However, the current practice in industry is to only design for stress minimization. As a result of this work it would appear that to achieve optimal life it may be best to use a fracture/fatigue rather than a stress based optimization process. This has clear implications for both safety and economics of production.

For this problem the search space contained multiple local optima; this fact was revealed by performing multiple searches from various starting points. Nimrod/O facilitated this by performing these searches in parallel. To understand these results a full survey of the search space was required. Here Nimrod was an appropriate tool.

References

1. Abramson D., Sosic R., Giddy J. and Hall B.: Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations. The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.

2. Buyya R., Abramson D., Giddy J.: Nimrod/G: an architecture for a resource Management and Scheduling system in a Global Computational Grid, 4th International Conference on High-Performance computing in the Asia-Pacific Region (HPC Asia), Beijing, May 2000, 283–289
3. Abramson D., Lewis, A. and Peachey, T., Nimrod/O: A Tool for Automatic Design Optimization, The 4th International Conference on Algorithms & Architectures for Parallel Processing (ICA3PP 2000), Hong Kong, December 2000.
4. Abramson D., Lewis, A. and Peachey, T. Case Studies in Automatic Design Optimisation using the P-BFGS Algorithm, 2001 High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference, Seattle, Washington (USA), April 2001, 104–109.
5. Abramson D, Lewis A, Peachey T, Fletcher, C., An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics, SuperComputing 2001, Denver, Nov 2001.
6. Broek D., The practical use of fracture mechanics, Kluwer Academic Publishers, 1988.
7. Jones R., Peng D., Chaperon P., Tan M., Abramsos D. and Peachey T. Structural Optimization with Damage Tolerance Constraints, J. Theoretical and Applied Fracture Mechanics, (in press).
8. Jones R. and D. Peng, A simple method for computing the stress intensity factors for cracks at notches, Engng. Failure Analysis **9** (2002) 683–702.
9. Nishioka, T. and Atluri, S.N., Analytical Solution for Embedded Elliptical Cracks, and Finite Element Alternating Method for Elliptical Surface Cracks, Subject to Arbitrary Loadings, Engng. Fracture Mechanics, **17** (1983) 247–268.
10. Paris P. C. and Erdogan F. A critical analysis of crack propogation laws, Trans. ASME, J. Basic Engng., **85** (1963) 528–534.
11. Mattheck C. and S. Burkhardt, A new method of structural shape optimization based on biological growth, Int. J. Fatigue, **12** (1990) 185–190.
12. Chaperon P., Jones R., Heller M., Pitt S. and Rose F., A methodology for structural optimisation with damage tolerance constraints, Journal of Engng. Failure Analysis **7** (2000) 281–300.
13. Jones R., Chaperon P., and Sawyer J. P. G., Structural optimisation with damage tolerance constraints, Ageing Studies Lifetime Extension of Materials, , L. G. Mallinson (ed.), Kluwer Academic Press (2001) 601–608.
14. Jones R., Chaperon P. and Heller M., Structural optimisation with fracturestrength constraints, J. of Engng. Fracture Mechanics, **69** (2002) 45–67.
15. Nelder, J. A. and Mead R., A simplex method for function minimization, The Computer Journal, **7** (1965) 308–313.
16. http://www.csse.monash.edu.au/ donny/mc2/tcp/paper/

Object Oriented Implementation of Modelling Bi-phase Gas-Particle Flows

Roman Wyrzykowski[1], Sebastian Pluta[1], and Jacek Leszczynski[2]

[1] Institute of Computer and Information Sciences,
Czestochowa University of Technology
{roman,pluta}@icis.pcz.pl
[2] Institute of Mathematic and Computer Science,
Czestochowa University of Technology

Abstract. This paper deals with the problem of modeling bi–phase gas-particle flows taking into account the basic mathematical model, numerical methods, problem geometry and object oriented model of an application used for modeling. The object-oriented approach is used to build an efficient software package based on DEM and MP-PIC methods.

1 Formulation of the Problem

Two-phase gas-particle flows are applied in many industrial domains, especially in chemical and processing engineering, coal combustion, powder production technology, environmental engineering, petrochemical industry, pharmaceutical industry and others. Empirical and half-empirical correlations between individual state variables, presented in the literature, do not include all possible cases of behaviour of such a flow. This is the reason for building the computer models of the two-phase gas-particle flows [3].

The computer simulations of particle movement, on basis of dual continua theory with the use of Navier-Stokes equations, were possible to apply in the eighties, due to progress in computer performance [9]. An assumption that gas and ensemble of particles are two mutually penetrating continuum gives constraint description of the solid phase movement, relying on its average parameters, e.g. average diameter of particles. That is because every loose material is characterised by diversification of particle shapes in accordance to a certain gas particle distribution. Collisions between particles, which modify their velocity vector, are also neglected in this case. Full application of the kinetic theory of gases [7] to the description of these flows is also impossible because of the lack of the correct collision term in the kinetic equation.

The application of Boltzman equation is limited only to the ensemble of particles with identical geometrical sizes in spite of certain analogy between gas molecules behaviour and particle movements. According to the dual continuum theory the interactions between particles are taken into consideration in the application of the kinetic theory of gases. These interactions are reduced to the simplest forms: particles collide centrally and they have identical sizes. The later

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 738–745, 2004.
© Springer-Verlag Berlin Heidelberg 2004

papers in this field [6] gave only certain generalization of consideration for the kinetic theory of gases applying to the two-phase gas-particles flows.

Unfortunately, the introduction of real quantities, such as distribution of particle shapes, particle weight distribution, their surface roughness, etc., substantially restrict the application of the above Euler-Euler approach to the description of particle movements in these flows.

Application of Euler's description to gas movement and Lagrange's desricption to particle movements is an alternative to the mentioned above approach.

As the first Tanaka and Tsui [12], using high-performance computers, investigated the simulations of two-phase flows describing movement of every particle introduced into installation and interacted with gas flow. The interactions between particles were replaced by the classic model of spring-damper used in an oscillation theory. The method of solving the equation of state was called DEM (Distinct Element Method). Binary interactions of particles were assumed here. It was also assumed that particles are balls with known diameters and that mass of a considered particle is unchangeable during collision. The friction forces between colliding particles are also neglected. Despite the use of supercomputers, only movement of small number of particles and only in the part of installation was possible to analyse. A significant restriction was e.g. allocation of state variables for particles in computer memory. In spite of the above, this approach was still developed, especially in the analysis of local effects, e.g. collisions of particles–wall [10].

The deriviation of solutions for the flow-state equation became an essential problem in Euler-Lagrange approach, as opposed to computer models describing studied flow with the use of Euler-Euler approach. Therefore, recently worked out MP-PIC [Multiphase Particle in Cell] method [2] is an important achievement. The above method is based on the PIC [Particle in Cell] method proposed by Hockney [5]. The MP-PIC method is characterised by Euler-Lagrange description taking into account different particle characteristics such as particle sizes, density of loose material.

This paper deals with the problem how to build a software package for modelling two-phase gas-loose material flows. This package enables modelling of such flows by the use of DEM and MP-PIC methods. The diagram (Fig. 1) presents key elements of building a software package for numerical modelling of physical phenomena. We start from the determination of mathematical model and then definition of specific numerical methods of model realisation takes place. The implementation of the determined methods includes three key mutually connected factors:

1. computer algorithms,
2. low-level data structures,
3. techniques of software engineering.

The presentation of this paper is organised in the following way. A mathematical model, applied for modelling of two-phase gas-loose material flows, is described in Section 2. Geometry of a considered problem is the key element of this model. It is an initial point to the definition of basic data structures (Section

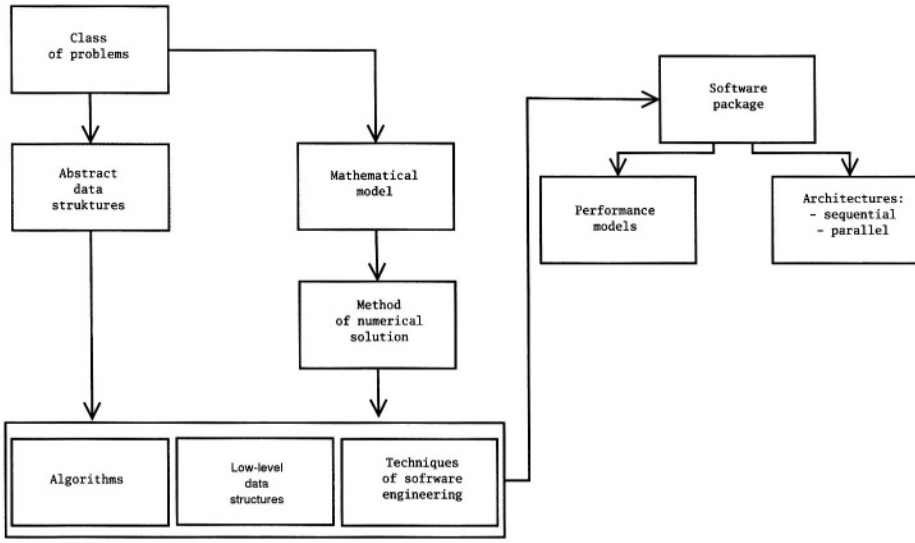


Fig. 1. Key elements of software packages for numerical modelling of physical phenomena

3), used in the target software package, which is built using the object-oriented approach. These structures are the basis of realization of numerical methods formulated in Section 4 for implementing the model. Section 5 includes conclusions and further works.

2 Mathematical Model

The mathematical model for solving the problem includes the following equations:

1. Equations of gas movements:

   a) equation of gas continuity

$$\frac{\partial \varepsilon_g \rho_g}{\partial t} + \frac{\partial \varepsilon_g \rho_g U_{gj}}{\partial x_j} = 0. \tag{1}$$

   b) equation of gas momentum exchange (Navier-Stokes equation)

$$\frac{\partial \varepsilon_g \rho_g U_{gj}}{\partial t} + \frac{\partial \varepsilon_g \rho_g U_{gi} U_{gj}}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\varepsilon_g \mu_g \frac{\partial U_{gi}}{\partial x_j}\right) + \tag{2}$$
$$+\frac{F_{mgi}}{V} + \frac{C_v \beta}{lz} \sum_{k=1}^{lz} \left(\frac{dx_i}{dt} - U_{gi}\right)$$

where:
ρ_g - gas density
U_g - velocity of gas
ε - cell porosity:

$$\varepsilon = 1 - C_v, \tag{3}$$

C_v - particle concentration in the cell

$$C_v = \sum_{l=1}^{lz} \frac{V_l}{V}, \tag{4}$$

lz - number of particles, located currently in the given cell
V_l - particle volume

$$F_{mgi} = \begin{cases} i = 1\ 0 \\ i = 2\ (m_p - m_g)g \\ i = 3\ 0 \end{cases}, \tag{5}$$

$$\beta = \begin{cases} \varepsilon \leq 0.8\ 150\frac{(1-\varepsilon)^2}{(\varepsilon\varphi D_p)^2}\mu_g + 1.75\frac{(1-\varepsilon)\rho_g \left|\boldsymbol{U}_g - \frac{d\boldsymbol{x}}{dt}\right|}{\varepsilon\varphi D_p} \\ \varepsilon > 0.8\ 0.75\frac{(1-\varepsilon)Cd(Re)}{\varphi D_p}\left|\boldsymbol{U}_g - \frac{d\boldsymbol{x}}{dt}\right|\varepsilon^{-2.65} \end{cases}, \tag{6}$$

$$Cd(Re) = \begin{cases} Re \leq 1000\ \frac{24}{Re}\left(1 + 0.15Re^{0.687}\right) \\ Re > 1000\ 0.44 \end{cases}, \tag{7}$$

$$Re = \frac{\varepsilon\rho_g D_p \left|\boldsymbol{U}_g - \frac{d\boldsymbol{x}}{dt}\right|}{\mu_g} \tag{8}$$

$$\left|\boldsymbol{U}_g - \frac{d\boldsymbol{x}}{dt}\right| = \sqrt{\left(U_{g_1} - \frac{dx_1}{dt}\right)^2 + \left(U_{g_2} - \frac{dx_2}{dt}\right)^2 + \left(U_{g_3} - \frac{dx_3}{dt}\right)^2} \tag{9}$$

φ - sphericality of the particle

c) equation of ideal gas

$$\frac{p}{\rho_g^{\kappa}} = idem. \tag{10}$$

2. Equation of motion of the solid phase
   a) progressive motion equation for a single grain without interaction with other grains

$$(m_p + a_p m_g)\frac{d^2 x_i}{dt^2} = \beta c_v V\left(U_{gi} - \frac{dx_i}{dt}\right) - F_{mgi} \tag{11}$$

where

$$m_p = \rho_p \frac{\pi D_p^3}{6}, \tag{12}$$

$$m_g = \varepsilon\rho_g V, \tag{13}$$

V - cell volume
a_p - coefficient of gas uplift pressure,

b) progressive motion equation for a single grain including interaction with other grains

$$(m_p + a_p m_g)\frac{d^2 x_i}{dt^2} = \beta c_v V\left(U_{gi} - \frac{dx_i}{dt}\right) - F_{mgi} +$$
$$- \sum_{j,j\neq i}\left(c_n\left(\frac{dx_i}{dt} - \frac{dx_j}{dt}\right) + k_n\big((r_i + r_j)e_{\zeta_j} - (x_j - x_i)\big)\right) +$$
$$- \sum_{j,j\neq i}\left(c_t\left(\frac{dx_i}{dt} - \frac{dx_j}{dt}\right) + k_t\big((r_i + r_j)e_{\eta_j} - (x_j - x_i)\big)\right) + \qquad (14)$$
$$- \sum_{j,j\neq i}\left(c_t\left(\frac{dx_i}{dt} - \frac{dx_j}{dt}\right) + k_t\big((r_i + r_j)e_{\xi_j} - (x_j - x_i)\big)\right)$$

The distinction between motionless cells, connected with gas movement, and movable cells, connected with particles, is the basic assumption of our computation methodology. It is illustrated in Fig. 2. One certain type of the rectangular mesh was shown here. In reality, cells can be set up as a result of an arbitrary space discretization.

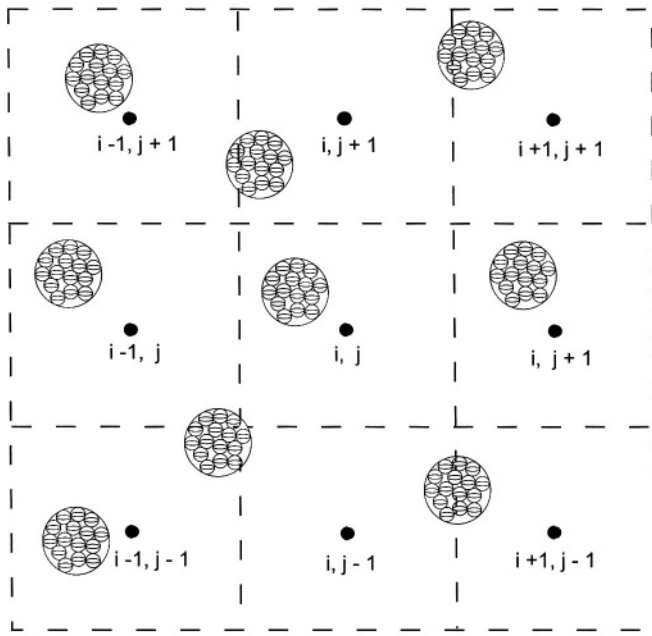


Fig. 2. The scheme of solving methodology for two-phase flow problem

3 Data Structures

The choice of adequate data structures is of great importance for the efficient implementation of the presented model. The nature of problem requires two separate structures - for gas and loose phases.

3.1 Data Structures for Gas

The gas phase is inseparably connected with the description of a control volume mesh. Parameters describing gas state are assigned to every control volume. So velocity of the gas phase is averaged inside the volume for every control volume. The following parameters describe every control volume:

1. connected with the mesh:
   a) centre of the cell,
   b) index of the cell in the mesh,
   c) area of contact surface with individual neighbouring cells,
   d) the distance between the centre of the cell and centres of neighbours,
   e) type of the cell (active, border, outflow , source),
2. connected with gas parameters:
   a) average velocity of gas in the cell,
   b) ε filling ration of the cell.

The software implementation of this problem contains TCV class describing control volume and 3D array of TCV objects, representing mesh of control volumes (Fig. 3).

3.2 Data Structures for Loose Phase

The loose phase is made up of an arbitrary amount of particles. They are assembled into bigger structures on the basis of size grade determing their properties. A single particle will be called “basic particle” and a group of particles - “substitute particle”. The substitute particle is described by:

1. position of particle centre,
2. current velocity,
3. index of control volume in which the particle is actually located,
4. D_p - substitute diameter of an ensemble of basic particles,
5. d_p - average diameter of basic particle,
6. ϱ - density of loose material,
7. N - the number of basic particles in the substitute particle,
8. N_k - the number of substitute particles.

The container of list type is applied to store the parameters of particle state, making adding and removal of elements relatively easy.

In general case, considerable part of data (d_p, D_p, ϱ, N and N_k) are identical for large quantity of particles. For example, if only the particles of one material

are in the system, then ϱ is constant for all particles. To decrease the memory complexity of software, the interval scale was applied for representation of the above mentioned parameters. The container, storing particles, sorts elements automatically in such a way that arranges elements with identical values of repeated parameters next to each other. As a result, given values of parameters and ranges of its occurrence arc stored in the container.

The identification of the parameter value for a specific particle takes place by checking its range. This mechanism is transparent for an user of the class describing the mentioned container. One can save even up to 80% of internal storage due to application of this mechanism, but at the cost of decrease in code performance.

The diagram of classes corresponding to these data structures is shown in Fig. 3.

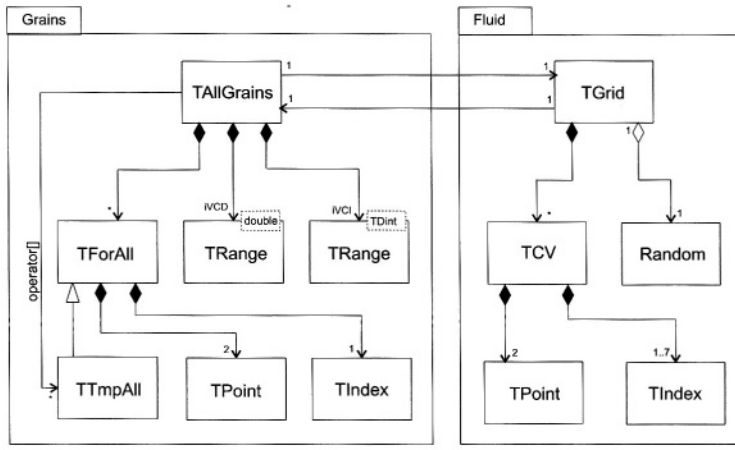


Fig. 3. Diagram of classes in UML illustrating data structures.

The TAllGrains class is a container which included list of TForAll class objects and adequate number of TRange template class objects describing parameters in the above presented method. The TForAll class describes those particle parameters whose values are different for different particles (velocity and position), while the TTmpAll class contains all data describing particle and is a result of invocation of array operator for the TAllGrains class.

4 Conclusions

The problems of numerical implementation of modelling bi–phase flows with the application of DEM and MP-PIC methods has been presented in the paper. These methods are base for building the software package using the object-oriented approach. The current status of software implementation includes: com-

putations of gas velocity, identification of particle collisions as well as solution of equation (11) and (15) with the use of Fehldberg method.

The nearest plans for this implementation include selection of suitable time step in case of particle collisions. The parallelization of computation [1] is foreseen in the prospect.

References

1. Andrews G. R., *Multithreaded, parallel, and distributed programming,* Addison Wesley, 2000.
2. Andrews M. J., O'Rourke P. J., *The multiphase particle-in-cell (MP–PIC) method for dense particulate flows,* Int. J. Multiphase Flow, 22, 1996.
3. Bis Z., *Circulation fluidisation of polidispersed mixtures,* Wydawnictwo Politechniki Czestochowskiej, Czestochowa 1999 (in Polish).
4. Coad P., Yourdon E., *Object-oriented analysis,* Oficyna Wydawnicza Read Me, Warszawa 1994 (in Polish).
5. Hockney R. W., Eastwood J. W., *Computer simulation using particles,* McGraw-Hill Inc., 1981.
6. Jenkins J. T., Mancini F., *Balance laws and constitutive relations for plane flows of a dense, binary mixture of smooth, nearly elastic, circular disks,* J. of Applied Mechanics, vol. 54, 1987.
7. Jenkins J. T., Savage S. B., *A theory for the rapid flow of identical, smooth nearly elastic spherical particles,* J. Fluid Mech., no. 130, 1989.
8. Krupowicz A., *Numerical methods for boundary problems with ordinary differentia equations,* PWN, Warszawa 1986 (in Polish).
9. Lyczkowski R. W., Gidaspow D., Solbrig C. W., *Multiphase flow models for nuclear, fossil and biomass energy production,* Advances in Transport Processes, New York, Wiley-Estern, 1982.
10. Sommerfeld M., *Modelling of Particle–wall collisions in confined gas–particle flows,* Int. J. Multiphase Flow, vol. 18, no. 6, 1992.
11. Subieta K., *Object-oriented approach to software design and data bases,* Akademicka Oficyna Wydawnicza PLJ, Warszawa 1998 (in Polish).
12. Tsuji Y., Kawaguchi T., Tanaka T., *Discrete particle simulation of two dimensional fluidized bed,* Powder Technology, 77, 1993.

Mathematical Linguistics Model for Medical Diagnostics of Organ of Hearing in Neonates

Mariusz Flasiński*[1], Elżbieta Reroń[2], Janusz Jurek[1], Piotr Wójtowicz[1], and Krzysztof Atłasiewicz[1]

[1] Chair of Applied Computer Science,
Institute of Computer Science Jagiellonian University
Nawojki 11, 30-072 Cracow, Poland
[2] Otolaryngological Clinic, Jagiellonian University Medical College
ul. Śniadeckich 2, 31-501 Cracow, Poland

Abstract. A mathematical linguistics model for constructing diagnostic expert system for evaluating of organ of hearing in neonates in electric response audiometry is presented in the paper. A model allows one to describe audiometry signals and recognise them with the help of parser. The formal language primitives, the meta-scheme of the rewriting system, and the DPLL(k) grammar are defined. The paper contains also a discussion of the whole system software architecture and its future extensions.

1 Introduction

Syntactic pattern recognition methods have been used for monitoring of signals describing physiology and pathology in medicine since the seventies [5]. The most popular application area of syntactic pattern recognition methods in medicine is ECG and EEG analysis [10,7,6].

At the same time the application of expert systems supporting medical diagnostics is common, especially in United States of America, Japan, and the countries of Western Europe. However, in the second case expert systems operate on preprocessed data delivered by the medical personnel, like facts resulting from medical observations or possible hypotheses about the diagnosis. The are practically no scientific reports on such expert systems that monitor signals describing continuous observational characteristics, and are able to recognise discrete observational facts being the base for formulating a diagnosis. The lack of such expert systems is caused by the fact that the expert systems being used in medicine are rule-based systems, which operate only on the discrete information (even if the information is fuzzy). In order to implement an expert system able to operate on continuous data, the model of "intelligent" transformation of continuous information to discrete one is needed. The model can be established on the basis of syntactic pattern recognition systems.

* Corresponding author.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 746–753, 2004.
© Springer-Verlag Berlin Heidelberg 2004

In the early nineties the research concerning a construction of such expert systems supporting CTG (cardiotocograph) diagnostics started in Institute of Computer Science, Jagiellonian University, Cracow, Poland. As a result of the research an expert system has been design and implemented. The expert system received two signals: the FHR (fetal heart rate) signal, and the TG signal describing the behaviour of a uterus (frequency, duration and intensity of contractions). The task of the system was defined as simultaneous monitoring of both signals and analysing of the signals with the help of two independent syntax analysers being the modules of the system. The goal of the analysis was to early warn medical personnel in situations when fetal health or life was in danger, especially during a birth. Since very good results of the research have been achieved, a new research project has been started: this time the research teams from Chair of Applied Computer Science, Jagiellonian University, and Otolaryngological Clinic, Jagiellonian University Medical College have begun investigation on possibilities of the application of syntactic pattern recognition methods in an expert system for evaluating of organ of hearing in neonates in electric response audiometry (E.R.A.) [8,9]. The results of the research are presented in the paper.

Let us stress that the novelty of our results consists in construction of such an expert system in medicine that is able to monitor signals describing *continuous* observational characteristics, and to recognise *discrete* observational facts for inferencing and formulating a diagnosis. It has become possible thanks to the use of advanced syntactic pattern recognition methods for recognising audiometry signals in time-series.

2 The Model of a Diagnostic System Based on Formal Languages

As far as a software architecture is considered, the system (in its first version) is based on the blackboard approach. The approach has been successfully verified in previous project of constructing real-time expert system, conducted by the research team from Chair of Applied Computer Science, JU, in Deutsches Elektronen-Synchrotron in Hamburg, Germany [1]. It is planned that a new version of the system will be based on multi-agent architecture, designed in the context of embedding syntactic pattern recognition algorithms in diagnostic expert systems [3].

Considering the aspect of application of artificial intelligence and pattern recognition methods, the model of the system is based on a two-level scheme. On the first level E.R.A. signals are treated as the sentences of a formal language. Both correct signal structures (i.e. without indications of pathology) and incorrect ones (eg. indicating Pierre-Robin syndrome or Down's syndrome [8] — see: Figure 1) are described by a formal grammar constructed for the language.

As it has been shown by the preliminary analysis of brainstem electric responses audiometry, the application of computationally efficient regular grammars for the analysis will not give us satisfactory results since the grammars are too weak in the sense of their generative (discriminative) power. On the

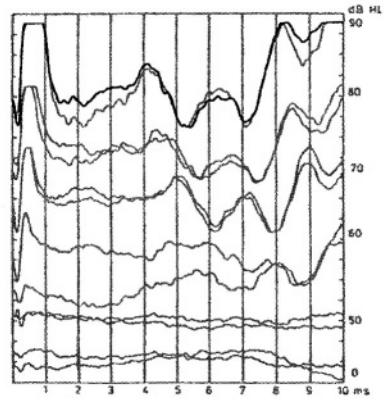


Fig. 1. Brainstem electric responses audiometry of a child with Down's syndrome: the left ear [8] X-axe represents time (in ms), y-axe represents intensity of signals (in dB HL)

other hand, it has been decided not to use Chomsky's grammars having the adequate generative power (i.e. context-sensitive grammars), because of non-polynomial computational complexity of automata performing syntax analysis of languages generated by these grammars. Therefore, so-called dynamically programmed LL(k) grammars (i.e. DPLL(k) grammars) introduced to the literature by Flasiński in 1996, and so-called programmed automata [4] have been chosen for the application in the system. DPLL(k) grammars are characterised by big descriptive power (they are stronger than context-free grammars) and at the same time the programmed automata able to analyse DPLL(k) languages are efficient.

Let us introduce a few basic definitions corresponding to DPLL(k) grammars [2].

Definition 1. A *dynamically programmed context-free grammar* is a quintuple

$$G = (V, \Sigma, O, P, S)$$

where V is a finite, nonempty alphabet; $\Sigma \subset V$ is a finite, nonempty set of terminal symbols (with N we denote a set of nonterminal symbols $N = V \setminus \Sigma$); O is a set of operations on a tape : *add, read, move;* $S \in N$ is the starting symbol; P is a finite set of productions of the form :

$$p_i = (\mu_i, L_i, R_i, A_i, DCL_i)$$

in which $\mu_i : \bigcup DCL_k \longrightarrow \{TRUE, FALSE\}$ is the predicate of applicability of the production p_i; $L_i \in N$ and $R_i \in V^*$ are left- and right-hand sides of p_i respectively; a pair (L_i, R_i) will be called a core of p_i (we assume that for each two various productions p_i, p_j from P, the core of p_i is different from the core of p_j, i.e. either $L_i \neq L_j$ or $R_i \neq R_j$); A_i is the sequence of actions of a type

add, move $\in O$ performed over $\bigcup DCL_k$; DCL_i is a derivation control tape for p_i. □

A derivation for dynamically programmed grammars is defined in the following way. Apart from testing whether L_i occurs in a sentential form derived, we check the predicate of applicability of a production p_i. The predicate is defined with use of $read(x)$ operation which is responsible for reading a current position of a head of a derivation control tape of a production x. If it is true, then we replace L_i with R_i, and then we perform the sequence of actions over derivation control tapes: $add(x, y)$ action is responsible for adding an index of a production y to a derivation control tape of a production x; $move(x)$ action is responsible for moving a head of a derivation control tape of a production x.

Definition 2. Let $G = (V, \Sigma, O, P, S)$ be a dynamically programmed context-free grammar. The grammar G is called a *Dynamically Programmed LL(k) grammar,* DPLL(k) grammar, if the following two conditions are fulfilled.

1. Let $w \in \Sigma^*$, $A \in N = V \setminus \Sigma$, $x, y, \alpha, \beta, \gamma \in V^*$. Then, for every two left-hand side derivations in G:
$$S \xrightarrow{*} wA\alpha \longrightarrow w\beta\alpha \xrightarrow[\text{core}]{*} wx$$
$$S \xrightarrow{*} wA\alpha \longrightarrow w\gamma\alpha \xrightarrow[\text{core}]{*} wy$$
such, that: $First_k(x) = First_k(y)$, the following condition holds: $\beta = \gamma$.
2. For a grammar G there exists a certain number ξ such, that for any left-hand side derivation $S \xrightarrow{*} wA\alpha \xrightarrow{\pi} w\beta\alpha$ (where $w \in \Sigma^*$, $A \in N$, $\alpha, \beta \in V^*$) fulfilling a condition: $|\pi| \geq \xi$, the first symbol of $\beta\alpha$ is the terminal one.

(The following notations have been used: $\xrightarrow[G]{}$ denotes a single derivation step in G; $\xrightarrow[G]{*}$ denotes the transitive and reflexive closure of $\xrightarrow[G]{}$; $\xrightarrow[G]{\pi}$ denotes such a derivation in G, which consists in application of succeeding productions from π, where π is a string of indices of productions; $\xrightarrow[G_{\text{core}}]{}$ denotes such a single derivation step in G that consists in an application of only a production core as a rewriting rule; $\xrightarrow[G\ \text{core}]{*}$ denotes the transitive and reflexive closure of $\xrightarrow[G\ \text{core}]{}$; $First_k(x)$ is a simple extension of a $First_k(x)$ notation used for $LL(k)$ grammars.) □

The algorithm of the parser for DPLL(k) grammars (DPLL(k) parser) has been described in [2]. We will not present it in the paper, but let us notice that the algorithm exactly reflects the way how the derivation in the grammar is performed. The algorithm uses top-down approach during the derivation. FIFO (first-in, first-out) queues are used to simulate all *DCL* tapes. Before application of a production the algorithm checks its predicate of applicability. After application of a production (i.e. after application of the re-writing rule) the algorithm performs *actions* on the queues, adding or removing elements, and in this way it *dynamically programs* future steps of the derivation.

As it has been described at the beginning of this section, DPLL(k) grammars have been chosen for the application in the system as a base of its first level

(responsible for analysis of E.R.A. signals as the sentences of a formal language). This choice has been successfully verified in practice. The generative power of DPLL(k) grammars have appeared to be satisfactory even in very sophisticated cases of brainstem electric responses audiometry signals.

On the second level of the system, the result of syntax analysis of E.R.A. signals should be put together with other parameters (of a discrete type) like mother's diseases (eg. bleeding, diabetes) [8]. This action takes place in the inference module constructed according to a rule-based approach. Let us consider the following (simplified) rule:

```
IF
Pierre-Robin_syndrome_detected AND
pregnancy_without_pathology
THEN
type_of_hearing_damage :=
acoustic_duct_damage
```

Indeed, if a Pierre-Robin syndrome is detected and on the other hand there were no pathologies during pregnancy, we should assume that the type of hearing damage is an acoustic duct damage — and this knowledge is reflected in the system by the rule above.

The inference module has been prepared with the help of the expert system building shell *RTworks* from *Talarian* [11]. Now, at the present stage of system development, we conduct the research into formalisation of all necessary medical knowledge in the form of rules written in *RTworks.* A description (structure) of medical facts is to be defined as classes and objects (the *RTworks* shell allows one to implement the domain knowledge with the OO methodology).

3 The Functionality of the Subsystem Responsible for Syntactic Analysis of E.R.A. Signals

The functionality of the subsystem responsible for syntactic analysis of E.R.A. signals is based on the classic scheme and it consists of the following phases:

- smoothing of a pattern,
- segmentation of a pattern and extraction of primitives,
- syntax analysis (parsing) resulting in a classification of a pattern.

The set of primitives has been defined as $\{S, P, F, N\}$ — see: Figure 2. The primitives are characterised by the following conditions:

$$\begin{aligned} S: &\quad l \leq len, \\ P: &\quad (l > len) \wedge (a > slope), \\ F: &\quad (l > len) \wedge (|a| < slope), \\ N: &\quad (l > len) \wedge (a < -slope), \end{aligned}$$

where *len* and *slope* are border parameters for length and slope angle (correspondingly).

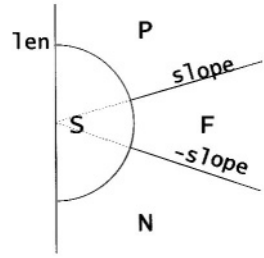


Fig. 2. The definition of primitives

The string grammar is constructed according to the following meta-scheme (the definition of a DPLL(k) grammar, presented in previous section, is complicated — in order to demonstrate the DPLL(k) grammar constructed for E.R.A. analysis, we will use simplified notation omitting all "programming features"):

$$G = (V_T, V_N, P, S),$$

where:

$$V_T = \{S, P, F, N\},$$
$$V_N = \{hill, dale, pos, neg, flat, S\},$$
$$S = V_N,$$

and P consists of the following productions:

$$hill \longrightarrow pos\ neg \mid pos\ short\ neg$$
$$dale \longrightarrow neg\ pos \mid neg\ short\ pos$$
$$pos \longrightarrow P \mid pos\ P \mid pos\ short\ P$$
$$neg \longrightarrow N \mid neg\ N \mid neg\ short\ N$$
$$flat \longrightarrow F \mid flat\ F \mid flat\ short\ F$$
$$short \longrightarrow S$$

An exemplary result of the analysis of brainstem electric responses audiometry signals in case of a child with Down's syndrome is presented in Figure 3 (compare: Figure 1).

4 Concluding Remarks

In the paper we have presented the recent results of the research into construction of a diagnostic expert system for evaluating of organ of hearing in neonates in electric response audiometry. The novelty of the expert system consists in use of advanced syntactic pattern recognition methods for recognising audiometry signals in time-series. As it has been mentioned in the introduction there are no reports on such expert systems in medicine that monitor signals describing continuous observational characteristics, and are able to recognise discrete observational facts being the base for formulating a diagnosis.

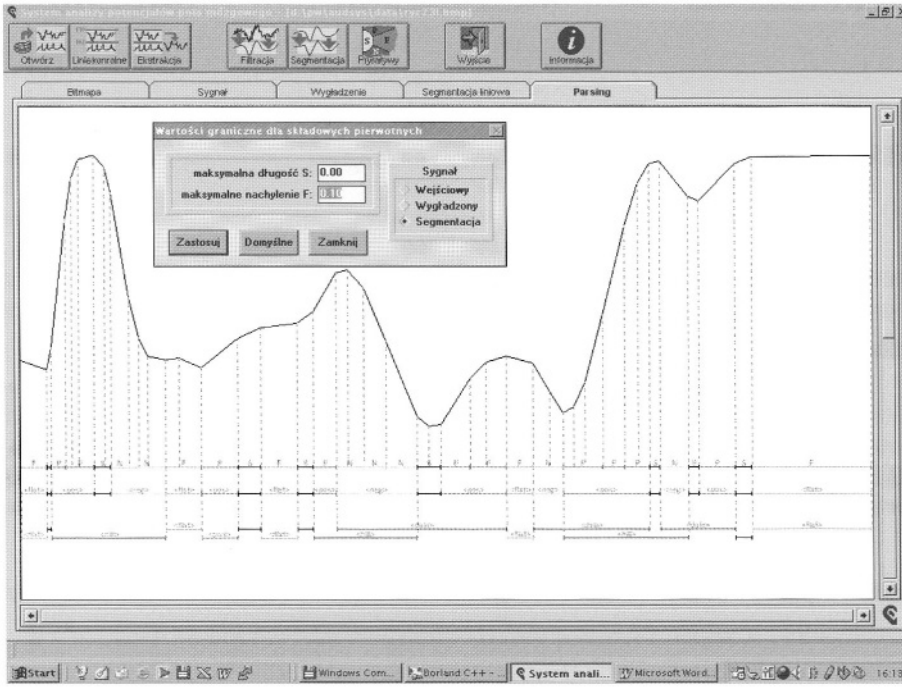


Fig. 3. The syntax analysis of brainstem electric responses audiometry signals in case of a child with Down's syndrome: the left ear. X-axe represents time (in ms), y-axe represents intensity of signals (in dB HL)

The syntactic pattern recognition model has been based on DPLL(k) grammars, DPLL(k) grammars are characterised by big descriptive power (they are stronger than context-free grammars) and at the same time a parser for DPLL(k) languages is efficient (of linear computational complexity). E.R.A. signals are interpreted by the syntactic pattern recognition module as words in a DPLL(k) language, and the words can be recognised by the DPLL(k) parser.

The model has been successfully verified in practice. Syntactic pattern recognition module has been able to identify even very sophisticated symptoms in E.R.A. However, some work is still to be done. Now, we do the research into constructing the second, rule-based, level of the system. It is not enough to provide a recognition of symptoms in E.R.A. The formalisation of the medical knowledge concerning evaluation of organ of hearing (on the base of the symptoms) is needed to deliver a really helpful tool for medical personnel. Our first experiences show that this can be done by the application of rule-based approach. The discussion of the final results of the research will be a subject of future reports.

References

1. Behrens, U., Flasiński, M., Hagge, L., Jurek, J., Ohrenberg, K.: Recent developments of the ZEUS expert system ZEX, IEEE Trans. Nucl. Sci. **NS-43** (1996), 65–68
2. Flasiński, M., Jurek, J.: Dynamically Programmed Automata for Quasi Context Sensitive Languages as a Tool for Inference Support in Pattern Recognition-Based Real-Time Control Expert Systems. Pattern Recognition, Vol. 32 (1999) no. **4**, 671–690, Elsevier Science
3. Flasiński M.: Automata-Based Multi-Agent Model as a Tool for Constructing Real-Time Intelligent Control Systems. Lecture Notes in Artificial Intelligence **2296** (2002), 103–110, Springer Verlag
4. Flasiński M.: The Programmed Grammars and Automata as Tools for a Construction of Analytic Expert Systems. Archives of Control Sciences **40** (1995), 5–35
5. Fu K.S.: Syntactic Pattern Recognition and Applications, Prentice Hall, Englewood Cliffs, 1982
6. Koski A., Juhola M., Meriste M.: Syntactic recognition of ECG signals by attributed finite automata. Pattern Recognition, Vol. 28 (1995) no. **12,** 1927–1940, Elsevier Science
7. Pietka E.: Feature extraction in computerized approach to the ECG analysis. Pattern Recognition, Vol. 24 (1991) no. **2**, 139–146, Elsevier Science
8. Reroń E.: Badania kliniczne i elektrofizjologiczne narzadu słuchu u noworodków, Rozprawa Habilitacyjna, Akademia Medyczna im. M. Kopernika w Krakowie, 1990
9. Sekula J., Reroń E.: Evaluating of organ of hearing in neonates in impedance audiometry and E.R.A., Revisita Italiana di Otolaryngologia Audiologia e Foniatria **4** (1984), 477–480
10. Skordolakis E.: Syntactic ECG processing: A review. Pattern Recognition **19** (1986), 305–313, Elsevier Science
11. Talarian Corporation, Mountain View, USA: RTworks v 2.1 Users Manual, December 1992.

Parallelization in an Algorithm of Multi-dimensional Nonconvex Optimization: An Application to Insurance Network Design*

Arkady Kryazhimskiy[1] and Vyacheslav Maksimov[2]

[1] V.A. Steklov Institute of Mathematics, Russian Academy of Sciences,
8 Gubkin Str., 117966 Moscow, Russia
kryazhim@aha.ru
[2] Institute of Mathematics and Mechanics,
Ural Branch of the Russian Academy of Sciences,
16 S.Kovalevskaya Str., 620219 Ekaterinburg, Russia
maksimov@imm.uran.ru

Abstract. The paper deals with the area of management of large-scale economic and environmental systems. A nonconvex multi-dimensional optimization problem related to the issue of optimal design of a network of insurance companies is analyzed. In order to avoid dramatic increase in optimization time, a general optimization algorithm is specified and presented as a series of elementary parallel operations. The optimal number of parallel processors is estimated.

1 Introduction

Management of large-scale economic and environmental systems is often concerned with the necessity to solve highly complex global optimization problems whose complexity is primarily due to nonconvexity and multi-dimensionality. There are numerous types of nonconvexity and every particular type of it requires a strictly specialized global optimization technique; this opens up a broad area for mathematical analysis, naturally focusing on nonconvexities arising in applications. Multi-dimensionality implies the danger of a dramatic increase in optimization time; in this context, specification of general optimization algorithms and, particularly, the selection of parallelizable fragments in them become especially important. In the present paper we analyze a nonconvex multi-dimensional optimization problem related to the issue of optimal design of a network of insurance companies. We consider a successive global optimization algorithm suggested in [1,2] and specify it so that each step of the algorithm is decomposed into a sequence of explicit formulas and finite optimization procedures.

* This work was supported by the International Institute for Applied Systems Analysis; by RFBR, project #03-01-00737; and by the Program on Basic Research of the Department of Mathematics of the Russian Acad. Sci., the project on Methods of System Optimization and the Program of supporting leading scientific schools of Russia, project #1846.2003.1.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 754–761, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Finally, we present the algorithm as a series of elementary parallel operations and estimate the number of parallel processors.

2 Insurance Network Optimization Problem

The underlying model is the following [1]. Companies $1,\ldots,M$ insure regional property against natural catastrophes (say, earthquakes). The region is split into cells numbered 1,..., N; d_i denotes the cost of property in cell i and c_{ij} the transaction cost for company j insuring property in cell i. Variable parameters are the premium for a unit of property, p, and the insurance distribution matrix, $X = (x_{ij})$; here x_{ij} is the share of property in cell i which is insured by company j. Assuming that all property in each cell is insured, we have

$$x_{ij} \geq 0 \quad (i = 1,\ldots,N,\ j = 1,\ldots,M), \quad \sum_{j=1}^{M} x_{ij} = 1 \quad (i = 1,\ldots,N). \tag{1}$$

If a catastrophe damages cells covering a set $I \subset \{1,\ldots,N\}$, the loss of company j is found as

$$L_{Ij}(p,X) = \sum_{i\in I} d_i x_{ij} + \sum_{i=1}^{N} c_{ij}x_{ij} - \sum_{i=1}^{N} p d_i x_{ij}. \tag{2}$$

Remembering that for strong catastrophes there is usually no reliable statistics [3,4], we adopt a non-stochastic scenario-based approach. Namely, we treat any set of damaged cells, I, as a *catastrophe scenario* and assume that a set $\mathcal{I}$ of *admissible* catastrophe scenarios is fixed. Then

$$L_{Ij}(p,X) \leq L_j^0 \quad (j = 1,\ldots,M,\ I \in \mathcal{I}) \tag{3}$$

gives a criterion for the survival of all companies under any admissible catastrophe scenario; here L_j^0 is a critical value for the loss of company j. In what follows $\mathcal{X}$ denotes the set of all insurance distribution matrices $X = (x_{ij})$ defined by (1). The *insurance network optimization problem* consists in minimizing premium p across all $(p,X) \in [0,\infty) \times \mathcal{X}$ satisfying the survival criterion (3).

The nonconvexity of the inequality constrains (3) implies that the insurance network optimization problem is nonconvex. The number of the constrains is $M|\mathcal{I}|$where $|\mathcal{I}|$ is the number of admissible catastrophe scenarios, i. e., elements of $\mathcal{I}$. The dimension of the problem is $NM + 1$, which is large, since the number of regional cells, N, is large; the number of companies, M, is relatively small, however, any new company increases the problem's dimension by N. The number of constrains (3), $M|\mathcal{I}|$, is also a strong factor of multi-dimensionality, since the number of admissible catastrophe scenarios, $|\mathcal{I}|$, is, generally, large. Besides, generally, $|\mathcal{I}|$ can be much larger than N.

The nonconvexity and multi-dimensionality of the insurance network optimization problem complicate the design of fast algorithms of numerical approximation to its global solution, whereas such fast algorithms should obviously be

a core of a decision support software allowing the user promptly estimate and compare optimal solutions for different sets of data (coalitions of insurance companies, critical values of losses, collections of admissible catastrophe scenarios, etc.)

3 Solution Algorithm

As stated in [1,2], the following algorithm produces a sequence (p^k, X^k) converging to the solution set of the insurance network optimization problem:

Algorithm A

Step 0. One chooses an arbitrary $X^0 \in \mathcal{X}$ and sets $p^0 = 0$.

Step $k+1$. Given $p^k \geq 0$ and $X^k \in \mathcal{X}$, one computes p^{k+1} and X^{k+1} by finding sequentially

$$s_{Ij}^k = \max\{0, L_{Ij}(p^k, X^k) - L_j^0\} \quad (I \in \mathcal{I},\ j = 1, \ldots, M), \tag{4}$$

$$f^k(p, X) = \sum_{j=1}^{M} \sum_{I \in \mathcal{I}} s_{Ij}^k (L_{Ij}(p, X) - L_j^0), \tag{5}$$

$$p^{k+1} = \min \left\{ p \geq p^k : \min_{X \in \mathcal{X}} f^k(p, X) \leq 0 \right\}, \tag{6}$$

$$X_*^{k+1} = \arg\min\{f^k(p^{k+1}, X) : X \in \mathcal{X}\}, \tag{7}$$

$$\tau^{k+1} = \arg\min \left\{ \sum_{j=1}^{M} \sum_{I \in \mathcal{I}} \max\{0, L_{Ij}(p^{k+1}, X^k + \tau(X_*^{k+1} - X^k)) - L_j^0\}^2 : \right. \tag{8}$$

$$\left. 0 \leq \tau \leq 1 \right\},$$

$$X^{k+1} = X^k + \tau^{k+1}(X_*^{k+1} - X^k); \tag{9}$$

if $p^{k+1} = p^k$ (clearly, $p^{k+1} \geq p^k$), the algorithms stops and gives out (p^k, X^k) being an exact solution to the insurance network optimization problem.

A software package named *Insurer* was created on the basis of this algorithm by the Institute of Mathematics and Mechanics, Ural Branch of the Russian Academy of Sciences, jointly with the Risk Modeling and Society Project of the International Institute for Applied Systems Analysis [5]. A brief flowchart of the software is shown in Fig. 1. The structure of the algorithm is such that the most time-consuming are problems (6) and (7). Below, we suggest a specification of solution algorithm for these problems, as well as a parallel (multi-processor) version of it.

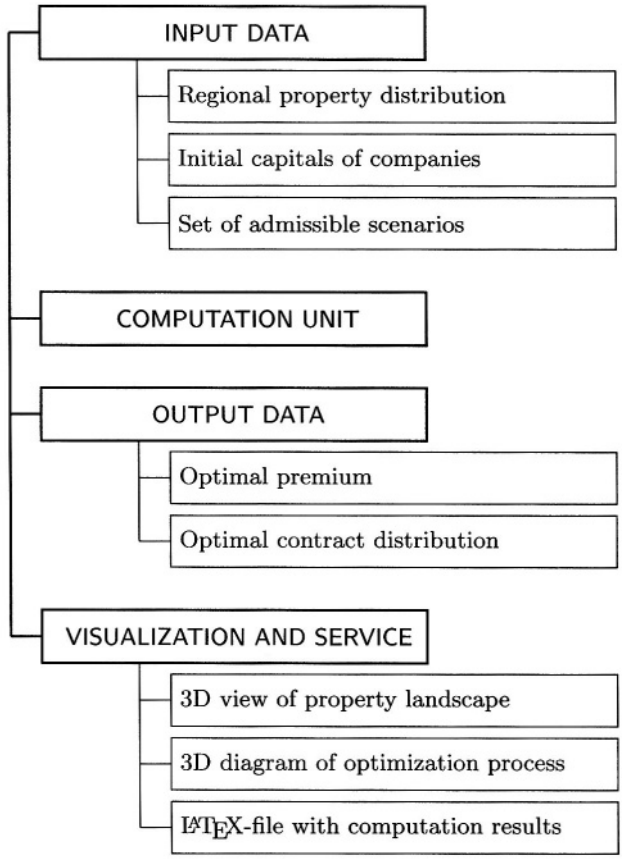


Fig. 1. A brief flowchart of the *Insurer* software package

3.1 Specification of Solution Algorithm

Let us provide an explicit description of the solution to problem (6) that is obviously central in step $k+1$. Using (5) and (2), we get

$$f^k(p,X)=\sum_{j=1}^{M}\sum_{i=1}^{N}(a_{ij}^k-b_{ij}^k p)x_{ij}-q^k=\sum_{i=1}^{N}\sum_{j=1}^{M}(a_{ij}^k-b_{ij}^k p)x_{ij}-q^k$$

where

$$a_{ij}^k=\sum_{I\in\mathcal{I}_i}s_{Ij}^k d_i+\sum_{I\in\mathcal{I}}s_{Ij}^k c_{ij},\quad \mathcal{I}_i=\{I\in\mathcal{I}: i\in I\},\quad b_{ij}^k=\sum_{I\in\mathcal{I}}s_{Ij}^k d_i, \tag{10}$$

$$q^k=N\sum_{j=1}^{M}\sum_{I\in\mathcal{I}}s_{Ij}^k L_j^0. \tag{11}$$

Taking into account (1), we find that

$$\min_{X\in\mathcal{X}} f^k(p,X)=\sum_{i=1}^{N}\min_{j=1,\dots,M}(a_{ij}^k-b_{ij}^k p)-q^k=f^k(p,X^k(p)) \tag{12}$$

where

$$X^k(p)=(x_{ij}^k(p)),\quad x_{ij}^k(p)=\begin{cases}0 \text{ if } j\neq j_i^k(p),\\ 1 \text{ if } j=j_i^k(p),\end{cases} \tag{13}$$

$$j_i^k(p)=\arg\min\{a_{ij}^k-b_{ij}^k p: j=1,\dots,M\}.$$

Therefore, (6) is specified as

$$p^{k+1}=\min\{p\geq p^k: g^k(p)-q^k\leq 0\} \tag{14}$$

where

$$g^k(p)=\sum_{i=1}^{N} g_i^k(p), \tag{15}$$

$$g_i^k(p)=\min_{j=1,\dots,M}(a_{ij}^k-b_{ij}^k p)=a_{ij_i^k(p)}^k-b_{ij_i^k(p)}^k p. \tag{16}$$

Introducing

$$\alpha_{ilj}^k=\frac{a_{il}^k-a_{ij}^k}{b_{il}^k-b_{ij}^k}, \tag{17}$$

$$J_{ij}^{k-}=\{l=1,\dots,M: b_{il}^k<b_{ij}^k\},\quad \beta_{ij}^k=\begin{cases}\max\{\alpha_{ilj}^k: j\in J_{ij}^{k-}\} & \text{if } J_{ij}^{k-}\neq\emptyset,\\ -\infty & \text{if } J_{ij}^{k-}=\emptyset,\end{cases} \tag{18}$$

$$J_{ij}^{k+}=\{l=1,\dots,M: b_{il}^k>b_{ij}^k\},\quad \gamma_{ij}^k=\begin{cases}\min\{\alpha_{ilj}^k: j\in J_{ij}^{k-}\} & \text{if } J_{ij}^{k+}\neq\emptyset,\\ -\infty & \text{if } J_{ij}^{k+}=\emptyset,\end{cases} \tag{19}$$

$$J_{ij}^{k0}=\{l=1,\dots,M: b_{il}^k=b_{ij}^k\},\quad \delta_{ij}^k=\begin{cases}\min\{a_{ij}^k: j\in J_{ij}^{k0}\} & \text{if } J_{ij}^{k0}\neq\emptyset,\\ \infty & \text{if } J_{ij}^{k0}=\emptyset,\end{cases} \tag{20}$$

we easily state that for every $j=1,\dots,M$ and every $p\geq 0$ one has $j=j_i^k(p)$ or, equivalently (see (16)),

$$g_i^k(p)=a_{ij}^k-b_{ij}^k p \tag{21}$$

if and only if $\beta_{ij}^k\leq p\leq\gamma_{ij}^k$ and $a_{ij}^k\leq\delta_{ij}^k$. Then, setting

$$J_i^k=\{j=1,\dots,M: a_{ij}^k\leq\delta_{ij}^k,\ \beta_{ij}^k\leq\gamma_{ij}^k,\ \gamma_{ij}^k\geq p^k\}, \tag{22}$$

we find that for every $j\in J_i^k$ the interval

$$P_{ij}^k=[\max\{p^k,\beta_{ij}^k\},\gamma_{ij}^k] \tag{23}$$

is nonempty (if $\gamma_{ij}^k=\infty$, the latter point is not included in P_{ij}^k) and (21) holds for all $p\in P_{ij}^k$. Since for every $p\geq p^k$ (21) holds for some j (which necessarily lies in

J_i^k), we conclude that $[p^k, \infty) = \cup_{j \in J_i^k} P_{ij}^k$. Therefore, one can find $j_{i1}^k, \ldots, j_{im_i^k}^k \in J_i^k$ and

$$p_{i1}^k \in P_{ij_1^k}^k \cap P_{ij_2^k}^k,\ p_{i2}^k \in P_{ij_2^k}^k \cap P_{ij_3^k}^k,\ \ldots,\ p_{im_i^k}^k \in P_{ij_{m_i^k-1}^k}^k \cap P_{ij_{m_i^k}^k}^k \tag{24}$$

such that

$$p^k < p_{i1}^k < p_{i2}^k < \ldots < p_{im_i^k}^k, \tag{25}$$

$$j_i^k(p) = \begin{cases} j_{i1}^k & \text{if } p \in [p^k, p_{i1}^k], \\ j_{il}^k & \text{if } p \in [p_{il-1}^k, p_{il}^k] \quad (l = 2, \ldots, m_i^k - 1), \\ j_{im_i^k}^k & \text{if } p \geq p_{im_i^k}^k \end{cases} \tag{26}$$

and

$$g_i^k(p) = \begin{cases} a_{ij_{i1}^k}^k - b_{ij_{i1}^k}^k p & \text{if } p \in [p^k, p_{i1}^k], \\ a_{ij_{il}^k}^k - b_{ij_{il}^k}^k p & \text{if } p \in [p_{il-1}^k, p_{il}^k] \quad (l = 2, \ldots, m_i^k - 1), \\ a_{ij_{im_i^k}^k}^k - b_{ij_{im_i^k}^k}^k p & \text{if } p \geq p_{im_i^k}^k; \end{cases} \tag{27}$$

thus, we get an explicit piece-wise linear representation for $g_i^k(p)$; note that $m_i^k \leq M$.

Basing on this, we specify the form of $g^k(p)$ (see (15)) as follows. We order the set

$$P^k = \{p_{il}^k : l = 1, \ldots, m_i^k,\ i = 1, \ldots, N\}, \tag{28}$$

i. e., determine $p_r^k \in P^k$ where $r = 1, \ldots, m_1^k + \ldots + m_N^k = m^k$ so that

$$P^k = \{p_r^k : r = 1, \ldots, m^k\}, \quad p^k < p_1^k \leq \ldots \leq p_{m^k}^k; \tag{29}$$

note that $m^k \leq MN$ since $m_i^k \leq M$. From (15) and (27) we get the following piece-wise linear representation for $g^k(p)$:

$$g^k(p) = \begin{cases} a_1^k - b_1^k p & \text{if } p \in [p^k, p_1^k] \\ a_r^k - b_r^k p & \text{if } p \in [p_{r-1}^k, p_r^k] \quad (r = 2, \ldots, m^k - 1), \\ a_{m^k}^k - b_{m^k}^k p & \text{if } p \geq p_{m^k}^k; \end{cases} \tag{30}$$

here

$$a_r^k = \sum_{i=1}^N a_{ij_{il(i,r)}^k}^k, \quad b_r^k = \sum_{i=1}^N b_{ij_{il(i,r)}^k}^k \tag{31}$$

and $l(i, r)$ are determined by

$$l(i,1) = 1; \quad p_{il(i,r)-1}^k \leq p_{r-1}^k,\ p_{il(i,r)}^k \geq p_r^k\ (r = 2, \ldots, m^k - 1); \quad l(i, m^k) = m_i^k. \tag{32}$$

Due to (26)

$$\begin{aligned} j_{il(i,1)}^k = j_{i1}^k = j_i^k(p) \quad & \text{if } p \in [p^k, p_{i1}^k], \\ j_{il(i,r)}^k = j_i^k(p) \quad & \text{if } p \in [p_{il(i,r)-1}^k, p_{il(i,r)}^k] \quad (r = 2, \ldots, m^k - 1), \\ j_{il(i,m_i^k)}^k = j_{i,m_i^k}^k = j_i^k(p) \quad & \text{if } p \geq p_{im_i^k}^k. \end{aligned} \tag{33}$$

Note that $g^k(p)$ is continuous and decreasing in p (see (15) and recall that $g_i^k(p)$ is continuous and decreasing for every $i = 1, \dots, N$). Therefore, the solution p^{k+1} to problem (14) (or (6)) is given by

$$p^{k+1} = \begin{cases} p^k & \text{if } a_1^k - b_1^k p^k \le q^k, \\ (a_{r^k}^k - q^k)/b_{r^k}^k & \text{if } a_1^k - b_1^k p^k > q^k \end{cases} \tag{34}$$

where $r^k \in \{1, \dots, m^k\}$ is the index of the interval of the linearity of $g^k(p)$, which contains q^k; r^k is defined by

$$r^k = 1 \quad \text{if} \quad a_1^k - b_1^k p^k > q^k \quad \text{and} \quad a_1^k - b_1^k p_1^k \le q^k, \tag{35}$$

$$a_{r^k}^k - b_{r^k}^k p_{r^k-1}^k > q^k, \quad a_{r^k}^k - b_{r^k}^k p_{r^k}^k \le q^k \quad \text{if} \quad a_1^k - b_1^k p_1^k > q^k. \tag{36}$$

Recall that if $p^{k+1} = p^k$ or $a_1^k - b_1^k p^k \le q^k$, see (34)), then (p^k, X^k) is a sought solution. Suppose $p^{k+1} > p^k$ and consider problem (7). By (33) $j_{il(i,r^k)}^k = j_i^k(p^{k+1})$, therefore, observing (7), (12) and (13), we find that problem (7) is resolved by

$$X_*^{k+1} = X^k(p^{k+1}) = (x_{ij}^k(p^{k+1})), \quad x_{ij}^k(p^{k+1}) = \begin{cases} 0 \text{ if } j \ne j_{il(i,r^k)}^k, \\ 1 \text{ if } j = j_{il(i,r^k)}^k. \end{cases} \tag{37}$$

3.2 Algorithm A: Parallelization

Summarizing the above arguments, we arrive at the next specification of Algorithm A, whose major steps are decomposed into elementary operations performed in parallel.

Algorithm A (specified)

Step 0. One chooses an $X^0 \in \mathcal{X}$ and sets $p^0 = 0$.

Step k processing the approximate solution (p^k, X^k) found at step k is decomposed into steps $(k,1)$–$(k,12)$.

Step $(k,1)$. This step employs $|\mathcal{I}|M$ parallel processors indexed (I,j) ($I \in \mathcal{I}$, $j = 1, \dots, M$). Processor (I,j) computes values $L_{Ij}(p^k, X^k)$ (2) and s_{Ij}^k (4).

In steps $(k,2)$–$(k,7)$ the exact solution p^{k+1} to the key problem (6) is constructed.

Step $(k,2)$. This step prepares data for finding $g_i^k(p)$, the components of $g^k(p)$ (15). The step employs NM parallel processors indexed (i,j) ($i = 1, \dots, N$, $j = 1, \dots, M$). Processor (i,j) computes sums a_{ij}^k and b_{ij}^k (10); identifies the index sets J_{ij}^{k+}, J_{ij}^{k-} and J_{ij}^{k0} (18)–(20); computes value α_{ilj}^k (17) for every $l \in J_{ij}^{k+} \cup J_{ij}^{k-}$; and finds values β_{ij}^k, γ_{ij}^k, δ_{ij}^k(18)–(20).

Step $(k,3)$. In this step, points p_{il}^k, at which the linear representation of $g_i^k(p)$ (16) switches, are computed. The step employs N parallel processors indexed i ($i = 1, \dots, N$). Processor i identifies the index set J_i^k (22); stores the endpoints $\max\{p^k, \beta_{ij}^k\}$ and γ_{ij}^k of the (nonempty) interval P_{ij}^k (23) for every $j \in J_i^k$; and finds, sequentially, indices $j_{il}^k \in J_i^k$ and points p_{il}^k ($l = 1, \dots, m_i^k$) determined by (24) and (25).

Step $(k,4)$. The master processor forms set P^k (28) and orders it, i. e., uses (29) to determine points $p_r^k \in P^k$ $(r = 1,\ldots,m_1^k + \ldots + m_N^k = m^k \leq NM)$, at which the linear representation of $g^k(p)$ (30) is switched.

Step $(k,5)$ specifies the piece-wise linear form of $g^k(p)$ and identifies index r^k of the interval of linearity of $g^k(p)$, which contains p^{k+1}, the solution to problem (6). The step employs m^k parallel processors indexed r $(r = 1,\ldots,m^k)$. Processor r identifies index $l(i,r)$ determined by (32) for every $i = 1,\ldots,N$; finds coefficients a_r^k and b_r^k (31) in the linear representation of $g^k(p)$ (30) at its rth interval of linearity; and verifies if $r = r^k$ using criterion (35), (36) (the criterion is satisfied for the unique r).

Step $(k,6)$. The master processor computes value q^k (11) and uses (34) to find p^{k+1}, the solution to problem (6).

Step $(k,7)$ is performed by the master processor. If $p^{k+1} = p^k$, the algorithm stops and gives out (p^k, X^k) as an exact solution to the insurance network optimization problem. If $p^{k+1} > p^k$, then matrix X_*^{k+1} solving problem (7) is constructed using (37).

Step $(k,8)$. The master processor finds value τ^{k+1} (8) and matrix X^{k+1} (9).

4 Conclusion

The insurance network optimization problem is investigated in the article. This problem is formalized as a problem of nonconvex optimization of large dimensionality. An iterative algorithm is suggested for solving this problem. Special attention is paid to the decomposition of each step of the algorithm into a sequence of explicit formulas and finite optimization procedures. The algorithm is presented as a series of elementary parallell operations, and the number of the required parallel processors is estimated.

References

1. Digas, B.V., Ermoliev, Yu.M., Kryazhimskii A.V.: Guaranteed Optimization in Insurance of Catastrophic Risks, IIASA Interim Report IR-98-082, Laxenburg, Austria (1998)
2. Kryazhimskii, A.V., Paschenko, S.V.: On the Problem of Optimal Compatibility. Ill-Posed and Inverse Problems, Vol. 9, No. 3 (2001) 283–300
3. Keilis-Borok, V., Soloviev, A. (eds.): Nonlinear Dynamics of the Lithosphere and Earthquake Prediction. Springer-Verlag, Berlin Heidelberg (2002)
4. Soloviev, A., Maksimov, I., Rozenberg, V., Ermoliev, Yu.: Block Models of Lithosphere Dynamics: Approach and Algorithms. In: Wyrzykowski, *et al.* (eds.): Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, Vol. 2328. Springer-Verlag, Berlin Heidelberg (2002) 572–579
5. Baranov, S., Digas, B., Ermolieva, T., Rozenberg V., Earthquake Risk Management: a Scenario Generator. IIASA Interim Report IR-02-025, Laxenburg, Austria (2002)

Discrimination between Models of Distributed Parameter Systems Using T-optimum Experimental Design

Bartosz Kuczewski, Maciej Patan, and Dariusz Uciński

University of Zielona Góra, Institute of Control and Computation Engineering
50, Podgórna Str., 65-246, Zielona Góra, Poland
{B.Kuczewski, M.Patan, D.Ucinski}@issi.uz.zgora.pl

Abstract. The paper considers the problem of determining an optimal sensor location for discrimination between competing models of distributed parameter process. A measurement policy using stationary sensors is proposed based on a numerical scheme for computing T-optimum designs. The resulting numerical method is validated on a simulation example regarding the atmospheric pollution proliferation process.

1 Introduction

Being the initial part of modelling and simulation of many physical phenomena, the structure identification of complex process models constitutes an essential part of that methodology. Usually the choice of a particular model structure is ambiguous, since the mechanisms of many complex processes are not fully known. Detailed analysis of the considered phenomenon often leads to a simplified model of the investigated system, but occasionally several plausible models can be used. A specially designed experiment to discriminate between those rival models constitutes a good source of information about the model fit. Moreover, the very important aspects of the model level of complexity and cost of the experiment can be included into the criterion of the best model choice.

Motivations to develop some discrimination techniques come from practical problems. A representative example is the forecast of the level of groundwater or atmospheric pollutant emissions [14,12]. The phenomena of that kind are modelled with the aid of partial differential equations of advection-diffusion-reaction type. The main difficulty is that only a part of model parameters (e.g. wind velocity) belong to the set of measurable quantities. On the contrary, the diffusion coefficients or parameters occurring in chemical reactions cannot be directly measured. That inconvenience, connected with the fact that we deal with compound of a few complicated phenomena (transport, diffusion and chemical reactions in the mentioned case), significantly complicates the appropriate modelling of such systems.

The main and most important aspect of structure discrimination is the adoption of an appropriate quantitative criterion used to compare the quality of different designs. Experimental design theory for precise estimation of model

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 762–769, 2004.
© Springer-Verlag Berlin Heidelberg 2004

parameters with given structure is relatively well developed [1,8,5,15] and was applied to various kinds of systems, including processes with spatiotemporal dynamics [7,9] . However, the high level of complexity in the case of the experimental design for discrimination between models limited its application to relatively simple models of static systems. Various criteria were considered in [2,3,4,11]. Recently the criterion, called the T-optimality criterion, introduced by Atkinson and Fedorov [2,3] was generalized to discrimination between two rival multiresponse dynamic models given in the form of ordinary differential equations with observations corrupted by white noise [13,6].

In the present paper the approach based on T-optimum designs is generalized to find optimal stationary sensor locations which allow us for the most reliable discrimination between two models of a process with spatiotemporal dynamics. It is assumed that the outputs of the considered system are corrupted with Gaussian noise with zero mean and a known covariance matrix. Additionally, the assumption of the lack of correlation between outputs noise signals is made. The proposed numerical iterative procedure for computing optimum designs consists in solving a maximin problem where global optimization techniques have to be employed.

2 Design Problem in Context

Assume that $\Omega \subset \mathbb{R}^2$ represents a bounded, simply-connected domain with a suitably smooth boundary $\partial\Omega$. Consider a distributed parameter system whose mathematical model is expressed by the partial differential equation:

$$\frac{\partial y}{\partial t} = \mathcal{G}\left(x, t, y, \nabla y, \nabla^2 y\right), \quad (x,t) \in \Omega \times T, \tag{1}$$

subject to boundary and initial conditions defined as

$$\begin{cases} \mathcal{E}\left(x, t, y, \nabla y\right) = 0, & (x,t) \in \partial\Omega \times T, \\ \mathcal{F}\left(x, y, \nabla y\right) = 0, & x \in \Omega, \ t = 0 \end{cases} \tag{2}$$

where: $x = (x_1, x_2) \in \bar{\Omega}$ stands for the vector of spatial coordinates, $y(x,t) \in \mathbb{R}^s$ denotes the system state, $T \in [0, t_f]$ is the observation horizon, and $\mathcal{E}, \mathcal{F}$ and $\mathcal{G}$ denote some functions mapping their arguments into $\mathbb{R}^s$. $\mathcal{G}$ may include terms connected with input functions known *a priori.*

Assume that the observations $y_{ij} \in \mathbb{R}^s$ of the model response are described by the equation:

$$z_{ij}(t) = y(x^i, t) + \varepsilon_{ij}(t), \quad t \in T \quad i = 1, \ldots, n; \ j = 1, \ldots, r_i, \tag{3}$$

where$n$ denotes the number of measurement sensors, $x^i \in \Omega$ stands for sensor locations $(x^i \neq x^k$ for$i \neq k)$; $\varepsilon_{ij}(t)$ represents the statistical uncertainty deriving from possible measurement errors. We make the assumption that the errors $\varepsilon_{ij}(t)$ can be described by a spatially uncorrelated Gaussian stochastic process.

Moreover, the measurements can be taken repeatedly at a particular location, which is taken into account in the output equation by introducing the replication term r_i, $\sum_{i=1}^{n} r_i = N$.

Having set conditions (2), the main assumption of the proposed approach is the conformity of the structure $\mathcal{G}$ with $\mathcal{G}_1(\cdot, \tilde{p}_1)$ or $\mathcal{G}_2(\cdot, \tilde{p}_2)$, where functions $\mathcal{G}_1$ i $\mathcal{G}_2$ are given *a priori,* and $\tilde{p}_1 \in P_1 \subset \mathbb{R}^{m_1}$ and $\tilde{p}_2 \in P_2 \subset \mathbb{R}^{m_2}$ stand for vectors of unknown but *constant* parameters values (P_1 and P_2 are some compact sets). The aim of the experiment is to determine which of the models $\mathcal{G}_1$ and $\mathcal{G}_2$ is compatible with the investigated phenomenon. The proposed solution can be delineated as follows.

The least-squares estimate of parameters $\tilde{p}_\ell$ in the considered case can be expressed by the equation:

$$\hat{p}_\ell = \arg \min_{p_\ell \in P_\ell} \sum_{i=1}^{n} \sum_{j=1}^{r_i} w_i \int_T \|z_{ij}(t) - \eta_\ell(x^i, t; p_\ell)\|^2 \mathrm{d}t, \tag{4}$$

for $\ell = 1, 2$, where $w_i = r_i/N$, $\eta_\ell(x^i, t; p_\ell)$ is the solution of (1),(2) with $\mathcal{G}(\cdot) = \mathcal{G}_\ell(\cdot, p_\ell)$; $\|\cdot\|$ stands for Euclidean norm.

Without loss of generality we can assume that the first of the considered models $\mathcal{G}_1$ is 'true', i.e. it is compatible with the investigated phenomenon. This assumption entails the condition of knowledge of the underlying parameter vector $\tilde{p}_1$ (this value can be obtain as a result of a pilot experiment or some nominal values can be used if accessible). Then the problem of the best model selection reduces to maximization of the following criterion subject to weights w_i and spatial locations x^i [5]:

$$T_{12}^0(\xi_N) = \min_{p_2 \in P_2} \sum_{i=1}^{n} \sum_{j=1}^{r_i} w_i \int_T \|\eta(x^i, t) - \eta_2(x^i, t; p_2)\|^2 \, \mathrm{d}t, \tag{5}$$

where $\eta(x, t) = \eta_1(x, t; \tilde{p}_1)$, and the set of variables:

$$\xi_N \stackrel{\text{def}}{=} \begin{Bmatrix} x^1, \ldots, x^n \\ w_1, \ldots, w_n \end{Bmatrix} \tag{6}$$

is called *the N-observation normalized exact experimental design.* The quantities x^i and w_i are called the support and weights, respectively.

The criterion (5) defines some measure of discrepancy between the responses of both the models. Then the resultant optimum design have effect upon the maximum lack of fit of the second model in the sense of the sum of squared errors computed for optimum estimates of the p_2 parameter. It is intuitively clear, that the solution strongly depends on which model is assumed to be 'true', as well as on the true values of the parameters in this model ($\tilde{p}_1$ in the considered case). We wish to choose a design which is optimal for some particular value of $\tilde{p}_1$ in the hope that it is not too bad whatever the true $\tilde{p}_1$ happens to be. The dependency of the optimal solution on the model parameters is an unappealing characteristic of nonlinear experimental designs. Some solution to partially

overcome this drawback can be the use of desing methods which are 'robust' to parametric uncertainty [15].

Allowing for replicated observations at the support points x_i entails serious difficulties, as the resultant numerical analysis problem is not amenable to be solved by standard optimization techniques, particularly when N is large. This is caused by the discrete nature of the N-observation exact designs, since the weights w_i are rational numbers. To alleviate this inconvenience, the notion of the design is relaxed to all probability measures ξ over X which are absolutely continuous with respect to the Lebesgue measure and satisfy by definition the normalization condition $\int_X \xi(\mathrm{d}x) = 1$. The set of all this measures ξ is denoted as $\Xi(X)$, and the designs thus constructed are called *continuous designs.* They constitute a basis for modern experimental design theory [5,1,16]. The continuous equivalent of the criterion (5) can be then expressed in the considered case as follows:

$$T_{12}(\xi) = \min_{p_2 \in P_2} \int_X \left\{ \int_T \|\eta(x,t) - \eta_2(x,t;p_2)\|^2 \, \mathrm{d}t \right\} \xi(\mathrm{d}x) \tag{7}$$

and the design

$$\xi^\star = \arg \max_{\xi \in \Xi(X)} T_{12}(\xi) \tag{8}$$

is called locally T_{12}-optimum.

3 T_{12}-Optimality Conditions

The results presented in [2] can be easily generalized to prove the following result:

Theorem 1 *Assume that*

A1. sets X and P_2 are compact,
A2. η is a continuous function on $X \times T$,
A3. η_2 is a continuous function on $X \times T \times P_2$.

If the optimization problem (7) *possesses a unique solution $p_2^\star \in P_2$ for the design $\xi^\star$, then the necessary and sufficient condition for the T_{12}-optimality of the design $\xi^\star$ is that*

$$\int_T \|\eta(x,t) - \eta_2(x,t;p_2^\star)\|^2 \, \mathrm{d}t \le T_{12}(\xi^\star), \qquad \forall\, x \in X. \tag{9}$$

For each support point in the design $\xi^\star$ the inequality is replaced by equality in (9). *Moreover, the set of all possible optimum designs $\xi^\star$ is convex.*

4 Numerical Construction of Locally T_{12}-Optimum Designs

The proposed numerical scheme of computing locally T-optimum continuous designs constitutes a generalization of the iterative procedure introduced by

Atkinson and Fedorov [2] for the case of static systems. However, its usefulness was shown in the case of discrimination between multiresponse dynamic system models, described by ordinary differential equations [6]. The procedure can be schematically depicted as the sequence of the following steps:

Step 1. Choose an initial nonsingular design ξ_0. Set $k = 1$.

Step 2. In the k-th iteration find:

$$\hat{p}_{2k} = \arg \min_{p_2 \in P_2} \sum_{i=1}^{n} w_i \int_T \|\eta(x^i, t) - \eta_2(x^i, t; p_2)\|^2 \mathrm{d}t$$

$$x^{(k)} = \arg \max_{x \in X} \int_T \|\eta(x, t) - \eta_2(x, t; \hat{p}_{2k})\|^2 \mathrm{d}t$$

Step 3. If $\phi(x^{(k)}) - \Delta(\xi_k) \leq \epsilon$, where

$$\phi(x^{(k)}) = \int_T \|\eta(x^{(k)}, t) - \eta_2(x^{(k)}, t; \hat{p}_{2k})\|^2 \mathrm{d}t$$

$$\Delta(\xi_k) = \sum_{i=1}^{n} w_i \int_T \|\eta(x^i, t) - \eta_2(x^i, t; \hat{p}_{2k})\|^2 \mathrm{d}t$$

then $\xi^\star = \xi_k$. **STOP**. Otherwise, go to step 4

Step 4. Choose α_k from $0 \leq \alpha_k \leq 1$ and compute the convex combination of designs:

$$\xi_{k+1} = (1 - \alpha_k)\xi_k + \alpha_k \xi(x^{(k)})$$

where $\xi(x^{(k)})$ stands for the design concentrated only at one point $x^{(k)}$ with unit weight. Set $k = k + 1$ and return to step 2.

The choice of the α_k sequence is ambiguous [2] and should be detailed. The simulation presented in the next section was performed with $\alpha_k = 1/(1 + n)$, where n denotes the size of the design. Although the number of support points in the optimum design is not set *a priori,* algorithm possesses a property of finding designs minimum or close to minimum (in the sense of number of support points allowing for parameter estimation of a model and ensuring the non-singularity of a design as well). To achieve this, techniques of avoiding the clusterization phenomenon (the points in the resultant design forming 'clouds' or clusters with a small diameter in places of optimal sensor locations) have to be used [10].

The most important part of the algorithm, which directly affects the convergence of the proposed procedure, is the solution of the global optimization problem which appears in Step 2 of the scheme. It it the most computation time-consuming part of algorithm as well, since the system state is given implicitly as the solution of a particular PDE and each evaluation of the cost function for different model parameters is connected with necessity of resolving underlying PDE. Getting stuck in a local minimum usually leads to premature convergence of an algorithm or the lack of convergence at all. Therefore, an effective implementation of some global optimization method has a crucial significance into numerical realization of the presented approach. The relatively simple Adaptive Random Search method [16,6] was chosen. It is worth of noticing, that the ARS

algorithm can be parallelized, which is a very valuable property in the light of the foregoing remarks on evaluation of the cost function (especially when the dimensionality of the considered problem grows, due to an increasing number of model parameters or the complexity of the considered domain).

5 Numerical Example

In order to illustrate the properties of the proposed algorithm, a computer experiment was performed. For this purpose, the process of pollutant transport-diffusion over a given area Ω was considered. Assume that the pollutant concentration u over the time interval $T = [0,1]$ is described by the model in the form of advection-diffusion equation:

$$\frac{\partial u(x,t)}{\partial t} + \operatorname{div}(\mathbf{v}(x)u(x,t)) = \operatorname{div}(d(x)\nabla u(x,t)) \qquad \text{in } x \in \Omega, \tag{10}$$

subject to initial and boundary conditions

$$\begin{cases} u(x,0) = 100\mathrm{e}^{-100(x_1^2+x_2^2)} \text{ in } \Omega, \\ \dfrac{\partial u(x,t)}{\partial n} = 0 \qquad\qquad \text{in } \partial\Omega \times T, \end{cases} \tag{11}$$

where $\partial u/\partial n$ stands for the partial derivative of u with respect to the outward normal of boundary $\partial\Omega$. The following form of the distributed diffusion coefficient was assumed: $d(x) = 0.1 + 0.1x_1^4 + 0.1x_2^4$. The velocity of transport medium was modelled as a radial field directed outwards with the source situated at point $P = (0,0)$. The domain Ω, boundary $\partial\Omega$, contour of the initial concentration of the substance $u(x,0)$ and the gradient of the transport medium velocity are shown in Fig. 1a.

The alternative model has no advection part, so only diffusion of pollutant is considered. The model is expressed by the equation:

$$\frac{\partial u(x,t)}{\partial t} = \operatorname{div}(d_a(x)\nabla u(x,t)) \qquad \text{in } x \in \Omega \tag{12}$$

with boundary and initial conditions (11) on the same domain and the same observation horizon as in the previous model (10). The distributed diffusion coefficient of the alternative model has the form preserving symmetry: $d_a(x) = p_1 + p_2(x_1^4 + x_2^4)$. The allowed values of the alternative model parameters were fixed as $p_1, p_2 \in [0.01, 0.2]$. The program for computing optimum design was written completely in the Matlab 6.x environment aid of procedures from the PDE Toolbox (note that PDE Toolbox can not handle the advection-diffusion-reaction type PDE's directly).

The resulting optimum design includes two points and has the form

$$\xi^\star = \left\{ \begin{matrix} (-0.4002; -0.0110), & (0.0120; 0.3890) \\ 0.2097, & 0.7903 \end{matrix} \right\}$$

and the parameters of the alternative model with respect to $\xi^\star$ have the values: $\boldsymbol{p}_2^\star = (p_1, p_2) = (0.1004, 0.1048)$. The obtained sensitivity function $\phi(x) = \int_T \|\eta(x,t) - \eta_2(x,t;\hat{p}_2)\|^2 \mathrm{d}t$ defining the discrepancy between the responses of both the models for the optimum location of sensors is shown on Fig. 2b. It is worth of noticing, that the support points are located at the maxima of the $\phi(x)$ function which is consistent with the depicted theory. As can be seen in Fig. 2b, the sensitivity function is symmetric, which results from the symmetry of the problem. The function possesses four maxima, where values of the T_{12} criterion equal one another, but sensors are located only at two of them. This situation is caused by unavoidable numerical inaccuracies of computations (even the smallest possible float difference between the values of the appropriate criterion decides which of the points will be included into design during the present iteration of the algorithm).

6 Concluding Remarks

In the article a generalization of T-optimality criterion fitting to the considered case of discrimination between two models of distributed parameter system was presented. The proposed numerical algorithm was validated on the example connected with the problem of modelling the atmospheric pollution process. It should be pointed out that the high computational complexity of the task predisposes it to parallel implementation which is easy to do when using the ARS method (c.f. comments in Section 4). Proper work of the algorithm needs a significantly good FEM approximation of the solutions to the considered PDE's and this is obtained by refining the mesh, which increases the computational complexity. The optimization task, mentioned in Section 4 and

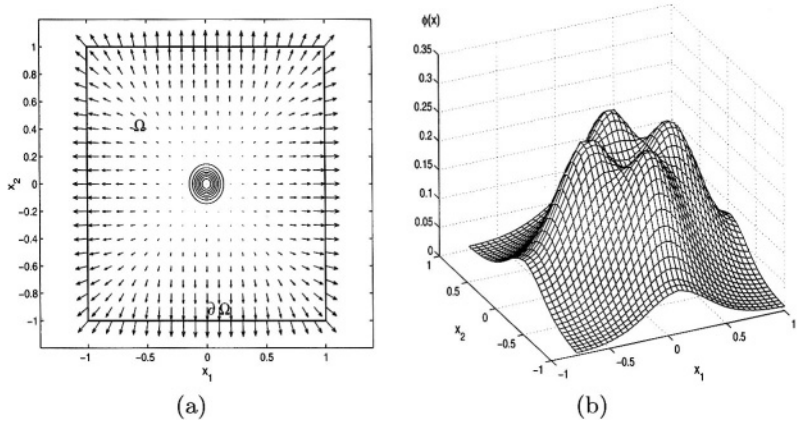


Fig. 1. Domain Ω, its boundary, initial concentration of a pollutant (contour plot) and transport medium velocity field (a) and 3D plot of the sensitive function $\phi(x)$ (b).

being crucial to convergence, additionally increases the necessary computation time. Finally, it is worth to noticing that the presented approach can be generalized to alternative measurement policies, e.g. scanning or moving sensors.

Acknowledgement. This research was supported by the State Committee for Scientific Research under Grant No. 7 T11A 023 20.

References

1. Atkinson A. C., Donev A. N.: Optimum Experimental Designs. Clarendon Press, Oxford, (1992).
2. Atkinson A. C., Fedorov V. V.: The designs of experiments for discriminating between two rival models. Biometrica, **62(1)**, (1975), 57–70.
3. Atkinson A. C., Fedorov V. V.: Optimal design: Experiments for discriminating between several models. Biometrica, **62(2)**, (1975), 289–303.
4. Burke A. L., Duever T. A., Pendilis A.: Model discrimination via designed experiments: Discriminating between the terminal and penultimate models on the basis of composition data. Macromolecules, **27**, (1994), 386–399.
5. Fedorov V. V., Hackl P.: Model-oriented Design of Experiments. Springer-Verlag, New York, (1997).
6. Kuczewski B., Uciński D.: Optimal design of measurement strategies for discrimination between multiresponse dynamic models. Proc. 8th Int. Conf. Methods and Models in Automation and Robotics, MMAR '02 , Szczecin, Poland, 2-5 Sept., 2002, **1**, (2002), 593–598.
7. Patan M., Uciński D.: Optimization of the measurement effort in parameter estimation of distributed systems. Proc. 8th Int. Conf. Methods and Models in Automation and Robotics, MMAR '02 , Szczecin, Poland, 2-5 Sept., 2002, **1**, (2002), 195-200.
8. Pukelsheim F.: Optimal Design of Experiments. Wiley, New York, (1993).
9. Rafajłowicz E.: Optimum Choice of Moving Sensor Trajectories for Distributed Parameter System Identification. International Journal of Control, **43(5)**, (1986), 1441-1451.
10. Rafajłowicz E.: Alghorithms of experimental design with implementations in MATHEMATICA. Academic Press Office PLJ, Warsaw, (1996) (in Polish).
11. Steward W. E., Shon Y., Box G. E. P.: Discrimination and goodness of fit of multiresponse mechanistic models. AIChE Journal, **44(6)**, (1998), 1404–1412.
12. Sun N. Z.: Inverse Problems in Groundwater Modelling. Theory and Applications of Transport in Porous Media. Kluwer Academic Publishers, Dordrecht, (1994).
13. Uciński D., Bogacka B.: Construction of T-Optimum Designs for Multiresponse Dynamic Models. Proceedings in Computational Statistics : 15th Symposium Held in Berlin, Berlin 2002, (2002), 267-272.
14. Uciński D., Patan M.: Optimization of the Measurement Effort in Parameter Estimation of Distributed Systems. Proc. 8th Int. Conf. Methods and Models in Automation and Robotics, MMAR '02, Szczecin, Poland, 2-5 Sept., **1**, (2002), 195-200.
15. Uciński, D.: Measurement Optimization for Parameter Estimation in Distributed Systems. Technical University Press, Zielona Góra, (1999).
16. Walter, É., Pronzato L.: Identification of Parametric Models from Experimental Data. Springer-Verlag, Berlin, (1997).

Robust Activation Strategy of Scanning Sensors via Sequential Design in Parameter Estimation of Distributed Systems

Maciej Patan and Dariusz Uciński

Institute of Control and Computation Engineering, University of Zielona Góra, ul. Pódgorna 50, 65–246 Zielona Góra, Poland
{M.Patan,D.Ucinski}@issi.uz.zgora.pl

Abstract. We study the problem of determining an activation strategy of discrete scanning sensors so as to maximize the accuracy of parameter estimation of a distributed system defined in a given multidimensional domain. Usually, when designing an identification experiment for nonlinear models, the uncertainty of nominal parameters has to be taken into account. To this end, an approach involving sequential design techniques is proposed, where there is no particular information about the parameter distribution. A scalar measure of the Fisher information matrix is used as the design criterion. The setting examined here corresponds to situations where there are many sensors and only some of them are activated during a given time interval. Additionally, the routine based on the extension of Fedorov's idea of directly constrained design measures is proposed and verified by a computer simulation regarding air pollution process.

1 Introduction

The problem of parameter estimation for distributed systems requires a properly designed process of data acquisition from various sources. This task comprises the determination of allocation for a limited number of sensors over the spatial domain in such a way as to maximize the estimation efficiency of the system parameters. Most often, the dependence between the sensor placement and the system performance is not intuitive and has counfounding nature. Due to the cost of obtaining data, constraints associated with measures and the need for increasing the efficiency of identifiers, there exists a strong necessity for developing some systematic methods.

The existing approaches to the sensor location problem concentrate mainly on the stationary (or motionless) sensor placement [8,9,11,4]. A very attractive alternative is the application of movable [6,11,10,12,8] or scanning [1,14,13] sensors, because it offers possibility to increase the degree of optimality for solutions. However, in all these situations, the process of designing an experiment generally strongly depends on the unknown parameter values which have to be estimated. The common approach is then to design the experiment for some

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 770–778, 2004.
© Springer-Verlag Berlin Heidelberg 2004

reasonable nominal value of parameters which are not known *a priori.* Since the uncertainty of this nominal values is not taken into account, the practical application of such procedure is limited to situations when the characteristics of the designs change slowly in some neighbourhood of the nominal parameter vector and which contain with high certainty the vector representing the true values of the parameters. To the best of our knowledge, in the context of scanning measurement strategies the problem has received no attention yet (for stationary and movable sensors some attempts have been made (for survey, we refer the reader to the monograph [8]), and therefore the aim of this work is to outline an approach to fill this gap to some extent.

Our basic idea is to extend the results for the so-called clusterization-free designs set forth in [5,13] in conjunction with some techniques taken from sequential design theory. The approach to constructing replication-free designs which is directly exploited here was proposed by Fedorov [3,2]. In spite of its somewhat abstract assumptions, the resulting extended multi-point correction algorithm of exchange type is very easy to implement.

2 Optimal Scanning Problem

The mathematical models of the DPS considered in this work are described by systems of n partial differential equations of the form:

$$\frac{\partial u}{\partial t} = \mathcal{G}\Big(x, t, u, \nabla u, \nabla^2 u; \theta\Big), \qquad x \in \Omega \subset \mathbb{R}^2, \quad t \in T = (0, t_f), \tag{1}$$

where Ω is some bounded simply-connected open domain with boundary Γ, $x = (x_1, x_2) \in \Omega \cup \Gamma$ is the spatial coordinate vector, T is the interval of observation and $u = u(x,t) \in \mathbb{R}^n$ denotes the system state. The system (1) is supplemented by a suitable set of boundary and initial conditions of the general form:

$$\begin{cases} \mathcal{E}\Big(x, t, u, \nabla u, \nabla^2 u; \theta\Big) = 0, & x \in \Gamma \times T, \\ u(x, 0) = u_0(x), & x \in \Omega \end{cases} \tag{2}$$

where $\mathcal{E}, \mathcal{G}$ and u_0 are some known functions which map their arguments to $\mathbb{R}^n$. In addition to that, $\mathcal{G}$ may contain terms accounting for known *a priori* forcing inputs. The vector $\theta \in \mathbb{R}^m$ represents unknown parameters which must be estimated based on the observations which are described by:

$$y^j_{k\ell} = u(x^j_k, t_k; \theta) + \varepsilon^j_{k\ell}, \; j = 1, \ldots, N, \; k = 1, \ldots, K, \; \ell = 1, \ldots, r_j, \tag{3}$$

where $t^j_k \in T$ denotes a sampling moment, y^j_{kl} is the scalar output, $x^j_k \in X$ stands for the location of the j-th sensor at time moment t_k, $X \subset \Omega$ is the set of admissible sensor locations. Let us note that in general the replications of measurements at some sensor positions are admitted and are represented as numbers of replications r_j. This fact immediately results from the assumption

of independent measurements, i.e. we adopt the common approach in which $\varepsilon_{k\ell}^j$ denotes the zero-mean, uncorrelated and white Gaussian measurement noise.

The purpose here is to choose sensor locations which maximize the expected accuracy of the parameter estimates obtained in the sense of the least-squares criterion. It can be shown that this task amounts to minimization of some scalar function Ψ of the *average Fisher Information Matrix* (FIM), which is given in the considered case by [13]:

$$M(\xi_1^N, \ldots, \xi_K^N) = \frac{1}{K} \sum_{k=1}^{K} \sum_{j=1}^{N} p_k^j g(x_k^j, t_k) g^{\mathrm{T}}(x_k^j, t_k), \tag{4}$$

where $g(x,t) = (\partial u(x,t;\theta)/\partial\theta)^{\mathrm{T}}_{\theta=\theta^0}$ stands for the so-called *sensitivity vector,* θ^0 being a prior estimate to the unknown parameter vector θ [10], $p_k^j = r_k^j/L$, $L = \sum_{j=1}^{N} r_k^j$. In (4) the following collection of variables

$$\xi_k^N = \begin{Bmatrix} x_k^1, x_k^2, \ldots, x_k^N \\ p_k^1, p_k^2, \ldots, p_k^N \end{Bmatrix}, \tag{5}$$

has been introduced, which is called the *exact design* of the experiment at moment t_k. The values p_k^j of observations performed at x_k^j can be considered as the percentage of experimental effort spent at that point. Such a problem formulation is generally accepted in optimum experimental design for DPS's, since the inverse of the FIM constitutes, up to a constant multiplier, the Cramér-Rao lower bound on the covariance matrix of any unbiased estimator of θ [15].

In the literature the most common choices for Ψ [15,3] are:

- the D-optimality (determinant) criterion: $\Psi(M) = -\log\det M$,
- the A-optimality (trace) criterion: $\Psi(M) = \operatorname{trace} M^{-1}$.

The p_k^j's are obviously rational numbers, because both r_k^j's and L are integers. Extending the definition of designs by assuming that they can be any real numbers of the unit interval [0,1] such that $\sum_{i=1}^{N} p_k^j = 1$, we may consider the designs as probability distributions over X. At this point it is possible to further generalize the concept of design to all probability measures ξ_k over X which are absolutely continuous with respect to the Lebesgue measure and satisfy by definition the condition $\int_X \xi_k(\mathrm{d}x) = 1$. Such a conceptual extension yields:

$$M(\xi_1, \ldots, \xi_K) = \frac{1}{K} \sum_{k=1}^{K} \int_X g(x_k, t_k) g^{\mathrm{T}}(x_k, t_k)\, \xi_k(\mathrm{d}x), \tag{6}$$

where the integration in (6) is to be understood in the Lebesgue-Stieltjes sense. This leads to the so-called *continuous* designs which constitute the basis of the modern theory of optimal experiments [3,15]. It turns out that such an approach drastically simplifies the design. What is more, it is also convenient to study an even more general setting by introducing a fixed measure $\xi(\mathrm{d}t)$ of observation

effort in the interval T, i.e. to consider

$$M(\xi) = \int_T \left[\int_X g(x,t) g^{\mathrm{T}}(x,t)\, \xi(\mathrm{d}x|t) \right] \xi(\mathrm{d}t) = \iint_{X \times T} g(x,t) g^{\mathrm{T}}(x,t)\, \xi(\mathrm{d}x, \mathrm{d}t), \quad (7)$$

where the measure $\xi(\cdot\,|t)$ corresponds to a spatial measure at time t satisfies

$$\int_X \xi(\mathrm{d}x|t) = 1 \quad \text{a.e. on } T, \quad \xi(\mathrm{d}t) = \int_X \xi(\mathrm{d}x, \mathrm{d}t). \quad (8)$$

This means that we focus on designs ξ being non-Cartesian product measures [7], or in other words, we interpret sensor locations at given time moments as conditional distributions.

Then we may redefine optimal designs as solutions to the optimization problem

$$\xi^\star = \arg\min_{\xi \in \Xi} \Psi[M(\xi)], \quad (9)$$

where Ξ denotes the set of all probability measures on $X \times T$ for which the measure defined by (8) is equal to the selected fixed sampling strategy regarding the time axis.

3 Sequential Designs for Scanning Strategy

From (4) it follows immediately that the values of the FIM for any design ξ depends on some initial vector of parameters θ^0 which is obviously unknown. Determination of the optimal design demand finding some bounded domain Θ, containing the true value of the parameters θ^0 and where $M(\xi;\theta)$ varies insignificantly for $\theta \in \Theta$. In case when the initial information about the parameters is missing, the only solution is to conduct an additional analysis for predetermination of the region Θ. Nevertheless, it is impossible to construct an optimal design for such an experiment without sufficient knowledge about parameters, because solutions of (9) most often depend on θ. Therefore a 'preliminary' design should be nondegenerate and has to guarantee uniqueness of the θ estimate, which is some kind of drawback, but has not great significance since finding such a design does not present any essential difficulty.

Typical practical conditions in the scanning measurement strategy make it possible to apply the sequential design technique to implement alternate experimental and estimation stages. The total time horizon is divided into adjacent intervals related to the consecutive stages of the experiment. The general scheme is shown in Fig. 1. Each stage consists of three sequential tasks: measurements are taken at the active sensors positions, then the obtained data are used for parameter estimation and, at last a locally optimal sensor placement is determined based on newly calculated parameter estimates.

With regard to its simplicity, the idea of sequential design is relatively easy in implementation and could be used in conjunction with a wide class of design

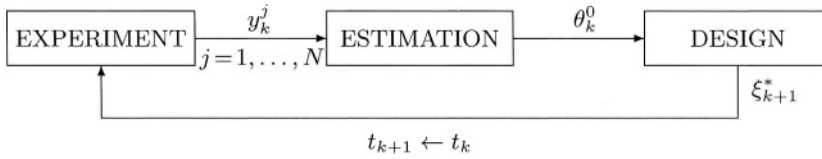


Fig. 1. Sequential design for scanning strategy

optimization algorithms without difficulties. Because the analytical determination of optimal designs is possible only in simple cases, this property has the great practical relevance. A computational procedure for that purpose is given in the next section.

4 Robust Scanning Algorithm

The activation policy of scanning sensors implies that the replications of measurements are not allowed during each phase of an experiment and at most one observation may be made at each available sensor position x. In this situation, the approach can be applied, which consists in selecting in every time step the best N-element subset S^* of a given P-element set X of all potential points where measurements may be taken, provided that $N < P$. It is convenient to assume that each time step corresponds to the next stage of the experiment. This idea is realized with the following iterative routine:

Sequential multi-point correction algorithm

Step 1. Guess an initial N-element set $S_1^0 \subset X = \{x^1, \ldots, x^P\}$ and the weight vector $w_0^{(1)}$ such that $\sum_{i=1}^P w_{0,i}^{(1)} = 1$. Set $k = 1,\ s = 0$. Choose some positive tolerance $\varepsilon \ll 1$.

Step 2. If $k > K$ then STOP, otherwise conduct an experiment for non-zero weight points and solve the problem

$$\hat{\theta}_k = \arg\min_{\theta\in\Theta} \sum_{i=1}^{P} w_{s,i}^{(k)} \|y_k^i - u(x_k^i, t_k; \theta)\|^2 \,\mathrm{d}t,$$

Step 3. Set $R_k^s = X \setminus S_k^s$. Construct the design ξ_k^s by setting

$$w_{s,i}^{(k)} = \begin{cases} 1/N & \text{if } x^i \in S_k^s, \\ 0 & \text{if } x^i \in R_k^s. \end{cases}$$

Step 4. Determine $M(\xi_k^s; \hat{\theta}_k)$ and for $i = 1, \ldots, P$ calculate

$$\psi(x^i, \xi_j^k) = \operatorname{trace}\Big[\overset{\circ}{\Psi}(\xi_k^s) g(x^i, t_k) g^{\mathrm{T}}(x^i, t_k)\Big] - \frac{1}{K} \operatorname{trace}\Big[\overset{\circ}{\Psi}(\xi_k^s) M(\xi_k^s; \hat{\theta}_k)\Big],$$

where

$$\overset{\circ}{\Psi}(\xi_k^s) = \left.\frac{\partial \Psi(M)}{\partial M}\right|_{M=M(\xi_k^s;\hat{\theta}_k).}$$

Step 5. If $\max_{x^i \in R_k^s} \psi(x^i, \xi_k^s) < \min_{x^i \in S_k^s} \psi(x^i, \xi_k^s) + \varepsilon$, then set $S_{k+1}^0 = S_k^s$, increment k by one, set $s = 0$ and go to Step 2.

Step 6. Sort the values of function ψ obtained in the Step 4 in decreasing order and relabel the points x^i by v^r in order to have

$$\psi(v^1, \xi_k^s) \geq \psi(v^2, \xi_k^s) \geq \ldots \geq \psi(v^P, \xi_k^s).$$

Step 7. Construct $S_k^{j+1} = \{v^1, \ldots, v^N\}$. Increment s by one and go to *Step 3.*

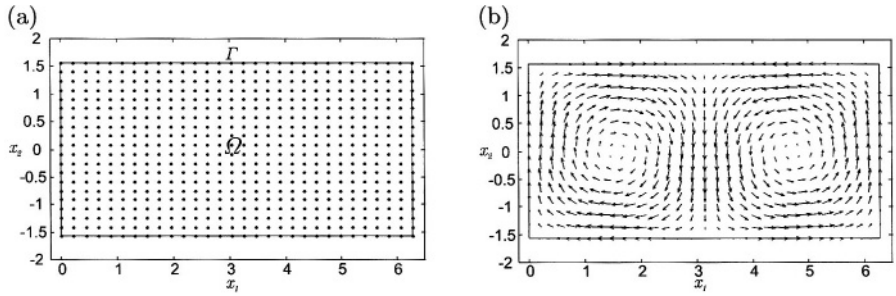


Fig. 2. Domain with admissible sensor locations (a) and wind velocity field (b).

For each time moment t_k (representing consecutive stage of the experiment) this exchange-type algorithm based on the notion of optimal designs with bounded density constructs a sequence of sets S^s converging to optimal S^* in the sense of the sensitivity function ψ (see Step 4 of the algorithm), which is equivalent to the optimality in the sense of criterion Ψ [8]. The procedure described above is an extension of the practical realization of scanning algorithm proposed in [13]. According to a limited space it is impossible to present a quite abstractive reasoning behind this algorithm (for details and convergence conditions we refer the reader to [13]). However, the efficiency of this iterative routine depends strictly on the effectiveness of the estimation problem present in Step 2, which is the main impediment of proposed approach.

5 Simulation Example

The transport-chemistry of air pollutants over a given area was considered as an illustrative example to verify the performance of the proposed algorithm. At the initial time instant, the contamination substance is emitted to the atmosphere near the centre of the given spatial region. The solid rotational velocity field of the transport medium as well as the possible sensor locations over this area are shown in Fig. 2. The pollutant concentration u over a given time interval $T = [0,1]$ is described by the following advection-diffusion process equation:

$$\frac{\partial u(x,t)}{\partial t} + \nabla \cdot \big(\boldsymbol{v}(x)u(x,t)\big) = \nabla \cdot \big(d(x)\nabla u(x,t)\big) \quad \text{in } x \in \Omega \tag{10}$$

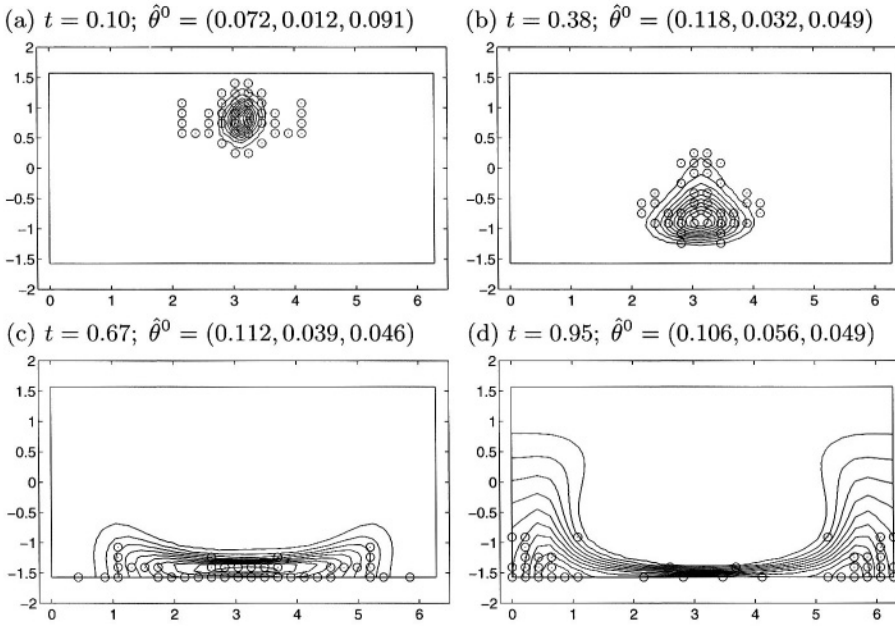


Fig. 3. Sensor configurations for the D-optimality criterion versus contour plots of concentration.

subject to the boundary and initial conditions:

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial n} = 0 & \text{on } \Gamma \times T, \\ u(x,0) = 10e^{-50[(x_1-\pi)^2+(x_2-1)^2]} & \text{in } \Omega. \end{cases} \tag{11}$$

where $\partial u/\partial n$ stands for the partial derivative of u with respect to the outward normal of Γ. Our aim is to provide optimal experimental conditions in the sense of the D-optimum activation policy for scanning sensors to estimate the unknown parameters of the distributed diffusion coefficient in the form

$$d(x) = \theta_1 + \theta_2(x_1 - \pi)^2 + \theta_3 x_2, \tag{12}$$

Sensors may be placed on the grid of size 30 × 20 (see Fig. 1(a)) and the partition of T is defined *a priori* by the switching points $t_k = k/20$, $k = 0, \ldots, 20$. The initial design was formed from randomly generated $N = 40$ points taken with equal weights. The algorithm was implemented with use of the Lahey/Fujitsu Fortran 95 compiler and a low-cost PC (Pentium IV, 1.7 GHz). The estimation phase was performed using simulated output data with 3% maximal noise level (the true value of parameters was assumed to equal $\theta = (0.1, 0.05, 0.05)$). The ultimate consecutive sensor configurations are shown in Fig. 3, where open circles indicate the activated sensor locations. As can be easily seen from Fig. 3 the symmetry of the problem along x_1 axis is retained and the sensors are activated in

each time instance near the region where the greatest changes of concentrations takes place.

6 Concluding Remarks

In this paper, a sequential design technique connected to some replication-free algorithm has been presented as an effective attempt to optimally locate scanning sensors in parameter estimation of distributed systems with respect to parametric uncertainty. The proposed approach, despite its abstract assumptions, results in an algorithm of exchange type, which is relatively easy to implement and gives possibilities to determine much more efficient scanning measurement policies compared with the stationary sensor strategies which have been considered in the literature so far. Nevertheless, the number of sensors which has been used should be relatively high (this is the main assumption while applying the algorithm based on directly constrained design measures). Otherwise, we can resort to standard discrete optimization routines which ensure that the constraints on the design measure and region are satisfied.

Acknowledgement. This research was supported by the State Committee for Scientific Research under Grant No. 7 T11 A 023 20.

References

1. Demetriou, M.A.: Activation policy of smart controllers for flexible structures with multiple actuator/sensor pairs. In A. El Jai and M. Fliess, editors, Proc. 14-th Int. Symp. MTNS, Perpignan, France, 19–23 June 2000, (2000), published on CD-ROM
2. Fedorov, V.V.: Optimal design with bounded density: Optimization algorithms of the exchange type. Journal of Statistical Planning and Inference, **22**, (1989), 1–13
3. Fedorov, V.V., Hackl, P.: Model-Oriented Design of Experiments. Lecture Notes in Statistics, Springer-Verlag, New York, (1997)
4. Kubrously, C.S., Malebranche, H.: Sensors and controllers location in distributed systems - A survey. Automatica, **21(2)**, (1985), 117–128
5. Müller, W.G.: Collecting Spatial Data. Optimum Designs of Experiments for Random Fields. Contributions to Statistics. Physica-Verlag, Heidelberg, (1998)
6. Rafajłowicz, E.: Optimum choice of moving sensor trajectories for distributed parameter system identification. Int. Journal of Control, **43(5)**, (1986), 1441–1451
7. Rao, M.: Measure Theory and Integration. John Wiley & Sons, New York, (1987)
8. Uciński, D.: Measurement Optimization for Parameter Estimation of Distributed Systems, Technical University Press, Zielona Góra, (1999)
9. Uciński, D.: Optimal selection of measurement locations for parameter estimation in distributed processes. International Journal of Applied Mathematics and Computer Science, **10(2)**, (2000), 357–379
10. Uciński, D.: Optimal sensor location for parameter estimation of distributed processes. International Journal of Control, **73(13)**, (2000), 1235–1248
11. Uciński, D.: Optimization of sensors' allocation strategies for parameter estimation in distibuted systems. System Analysis Modell. Simul., **37**, (2000), 243–260

12. Uciński, D.: Sensor motion planning with design criteria in output space. In A. Atkinson, P. Hackl, and W. Müller, (eds), mODa 6, Proc. 6-th Int. Workshop on Model-Oriented Data Analysis, Puchberg/Schneeberg, Austria, 2001, Physica-Verlag, Heidelberg, (2001), 235–242
13. Uciński, D., Patan M.: Optimal location of discrete scanning sensors for parameter estimation of distributed systems. In Proc. 15-th Trennial World Congress of the IFAC, Barcelona, Spain, Pergamon Press, Oxford, (2002), published on CD-ROM
14. van de Wal, M., de Jager, B.: A review of methods for input/output selection. Automatica, **37**, (2001), 487–510
15. Walter, E., Pronzato, Z.: Identification of Parametric Models from Experimental Data, Springer-Verlag, London, (1997)

Security in the OCM-G Grid Application Monitoring System*

Bartosz Baliś[1,2], Marian Bubak[1,2], Wojciech Rząsa[1], Tomasz Szepieniec[2], and Roland Wismüller[3,4]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
{balis,bubak}@uci.agh.edu.pl,wrzasa@student.uci.agh.edu.pl,
[2] Academic Computer Center – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM – Technische Universität München, D-80290 München, Germany
t.szepieniec@cyf-kr.edu.pl,
[4] Department of Software Science, University of Vienna, A-1090 Wien, Austria
wismuell@in.tum.de
phone: (+48 12) 617 39 64, *fax:* (+48 12) 633 80 54 *phone:* (+49 89) 289-28243

Abstract. This paper presents an analysis of a solution proposed to guarantee the security of a grid application monitoring system – the OCM-G. Two aspects of security are distinguished: inter-component communication and forge-component attack. The solution is based on methods of the Grid Security Infrastructure. We show that even authentication, encryption and integrity via digital signatures are not enough to achieve a desired security level. In our system, a special protocol of extending of the logical structures in the OCM-G, so called Virtual Monitoring Systems, was needed in addition to the standard security mechanisms.

Keywords: Grid, application monitoring, security, GSI tools

1 Introduction

Grid is a distributed computing infrastructure designed for advanced science and engineering. The concept of the Grid leans on resource sharing that is not an ordinary file exchange, but *direct access to computers, software data, and other resources* [6]. Careful control of the sharing requires a well defined infrastructure.

Applications designed for the Grid environment are highly distributed and their development process is significantly complicated. Thus, there is a high demand for Grid-enabled tools for application development. These tools are various in their functionality (e.g., debuggers, performance analyzers, etc.), but they all need a *monitoring infrastructure* to observe and/or control an application. The OCM-G – a Grid-enabled application monitoring system, described in detail in [2], is such an infrastructure.

* This work was partly funded by the European Commission, project IST-2001-32243, CrossGrid

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 779–787, 2004.
© Springer-Verlag Berlin Heidelberg 2004

This paper focuses on the security issues related to the OCM-G. At the beginning, it analyses security requirements specific for Grid applications, and then, after presenting the architecture of the OCM-G, it explains the security problems, and proposes adequate solutions. We show that even with all standard security mechanisms such as authentication, encryption, and digital signatures, we do not address all security problems.

2 Security in the Grid

Grid applications communicate via public network - the Internet. Therefore no additional network infrastructure is necessary. However using existing infrastructure to communicate between sites is significant advantage of Grid, it results in additional issues, that should be considered. Security of transmitted data is one of them.

2.1 Internet Security

Nowadays vulnerabilities of the protocols used in Internet, such as IP is widely known and well described [3], [4]. Since a precise description of network vulnerabilities is out of scope of this paper, we only present a brief description of threats.

Sniffing or *eavesdropping* – it is possible in some low level communication protocols where each transmitted packet can be received by each network interface in the network segment. It is significant threat to confidentiality of the transmission.

Spoofing - there are numerous varieties of that attack. It is possible for each protocol commonly used in Internet. *DNS spoofing* allows to impersonate host forging association between host name and it's IP address; *IP spoofing* makes attacker capable of deceiving authentication methods based on source address of the packet; *ARP Spoofing* allows third host to become an agent between two other hosts, and fully control connections.

Session take over (or session hijacking) – it allows an attacker to steal an already established TCP/IP session. Since an authentication is usually performed only on initialization of a connection, as a result of that attack, it is possible to access a host as an authorized user.

A solution which removes threats mentioned above is introduced by asymmetric cryptography (or public-key cryptography) features [9]. Algorithms based on asymmetric cryptography are capable of establishing secure connections over insecure public Internet links. Requirements of a secure transmission are met.

Various concepts based on asymmetric cryptography enable security in several aspects. (1) **Authentication** is performed by the use of certificates. Protocols based on public-key cryptography features are capable of unquestionable verifying identity of the peer passing over unreliable authentication mechanisms offered by the network protocols. (2) **Authenticity and Integrity** can be ensured by authentication of each packet transmitted over the network. This is

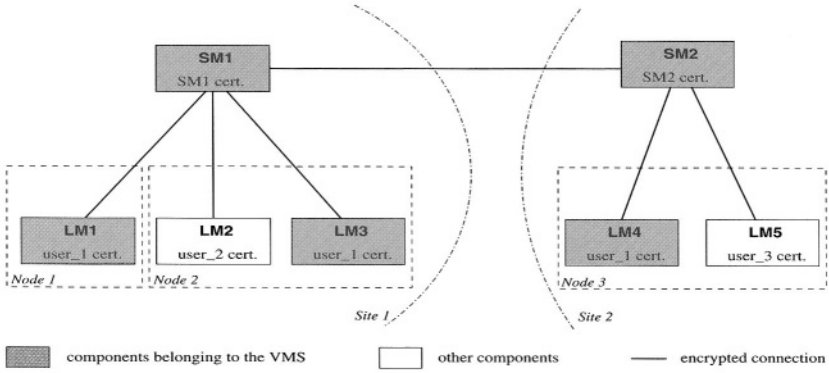


Fig. 1. OCM-G architecture

achieved by the use of digital signatures, so that the receiver of the message can verify identity of the sender. (3) **Confidentiality.** Asymmetric algorithms are significantly slower then symmetric ones and therefore public-key cryptography features are used to exchange symmetric *session key* so the confidential communication is possible.

2.2 Grid Security

All issues described above are essential for security of Grid applications. Communication between sites should be secure and reliable. However, communication is not the only security issue important in the Grid. To address the more complex security requirements of the Grid (e.g. *single sing on, credential delegation*) the Grid Forum Security Working Group was formed to work on the Grid Security Infrastructure (GSI) project [7]. The goal of this project is to work out a consistent security infrastructure for the Grid environment.

At present, there is no single standard addressing all Grid requirements, therefore, the GSI working group works on a new solution, that would be extent of existing standards, meeting above requirements. Existing solutions used by GSI rely on asymmetric cryptography features.

3 OCM-G Monitoring System

3.1 Architecture

The OCM-G is a collection of two types of components: per-site Service Managers (SMs), and per-host-and-user Local Monitors (LMs). Additionally, a part of the OCM-G is linked to the application's executable (Application Module) (see Fig. 1). The communication topology between these components is as follows: (1) LMs are connected to their master SM (on the same site), (2) Each pair of SMs can be connected, if necessary. Additionally, a tool can connect to any SM and send monitoring requests. Consequently, monitoring requests are distributed

in the OCM-G in the following way: (1) the SM which accepted a request, splits it into sub-requests and forwards them either directly to its LMs, or other SMs (if a sub-request operates on an object on a remote site), (2) sub-requests are executed in LMs, (3) replies are collected back via the same route as the distribution was done, until the tool receives a single reply to the request. Each LM belongs to one Grid user (thus, there may be more than one LM on each host – one for each Grid user), while SMs are shared by many users. However, SMs do not require super-user privileges. An important notion in the OCM-G is a *Virtual Monitoring System* (VMS). Each application has its own VMS which is simply the set of OCM-G components (SMs and LMs) which are involved in monitoring the specific application. Note that a VMS is a dynamic entitiy; for example, if a new application process registers in the OCM-G, the VMS may be extended by a new host (LM) or even site (SM). VMSs are important for information distribution – if the state of an application changes (e.g., a new process is created), only components of the VMS should be notified, not the whole OCM-G. We will see that VMSs are also important for security.

3.2 Security Requirements

The OCM-G is a common monitoring infrastructure which allows multiple users to monitor many applications, which not only includes data collection, but also manipulation. Security requirements which must be met in such a system are as follows. (1) **Credibility** requires the results obtained from the system should be correct, specifically a deliberate alteration should be excluded. (2) **Processes Security** ensures that application processes can be controlled by their owner only . In the Grid, it is especially important since in this way unauthorized person may obtain other user's temporary credentials and will be able to pose as the user. (3) **Confidentiality** of the monitoring results transmitted across the OCM-G components should be ensured at least if a user explicitly asks for it. (4) **Site security** cannot be lowered by an additional system service or an additional connection channel for the SM.

In order to respond the above requirements, connections between OCM-G components should address following aspects: (1) *authentication* – on connection establishment, each party should be able to unquestionably identify the other peer. It cannot be achieved with network protocol methods, since they can be easily deceived. (2) *authenticity and integrity* – each OCM-G component should be able to verify who is the author of a received piece of information. This should also ensure that information was not altered during the transmission. (3) *confidentiality* – no one should be able to eavesdrop transmission between OCM-G components.

4 Security in the OCM-G – Problems and Solution

4.1 Communication Security – GSI

Inter-component communication in the OCM-G meeting the requirements outlined in the previous section is achieved with the use of GSI solutions. In order to

perform reliable authentication, certificates are used. Authenticity and integrity of transmitted data is achieved by the use of digital signatures. Asymmetric cryptography features are used by the GSI to establish symmetric *session key.* Thereafter data transmitted between peers of the connection can be encrypted to ensure confidentiality. As a consequence each OCM-G component as well as a tool connecting to the OCM-G should be provided with certificate.

A **tool** and **Local Monitors** always belong to the user, and work on behalf of him. Since each Grid user has his own certificate which confirms his identity in the Grid, it seems to be natural for these components to use credentials of their owner. Therefore, no additional certificates are required for tools and LMs.

Service Managers are shared between users, therefore they cannot use credentials of any of them. Users who run multi-site applications are forced to trust remote sites' SMs, thus they should be able to recognize SMs on the basis of its identity. Moreover, to ensure the required security level, we should be able to distinguish different SMs, consequently each SM should use a different identity confirmed by a different certificate. Additionally, it would be useful to be able to unquestionably verify the location of a SM by the use of its identity.

Therefore, we have decided to provide each SM with a certificate with a specific *Distinguished Name* (DN) in the *Subject* field. Since a DN has a hierarchical structure, it will be possible to obtain information about the location of SM from its identity. It should be also possible to obtain a certificate for SM with a subject corresponding to the site, and with a fixed *Cannonical Name.* For instance `CN=OCM-G Service Manager`.

On establishing a connection with a SM, we are able to verify if the peer address matches address in the DN, and if the certificate was issued for a Service Manager.

4.2 Threat of Forging a Service Manager

We assume that all security solutions related to communication and authentication/authorization described in the previous section are applied. We will show that this is not enough – there is a threat that is not removed despite of the use of GSI to ensure security of communication and authentication.

Source of the threat – forged Service Manager. The problem we consider results from the fact that everyone is able to run a Service Manager. One may obtain SM's source code and modify it, or even write his own program to seemingly behalf as a Service Manager. Such a forged SM could be used to control all monitored applications belonging to different users on different sites. This is possible since once a component is part of the OCM-G, it is allowed to send any monitoring request.

Site administrators – trusted persons? Since we use the GSI to authenticate peer of each connection, we can identify a Service Manager on the basis of its certificate. Thus, the one who wants to perform the forged-SM attack would need a valid SM certificate. Normally, only a site administrator can obtain an SM certificate. However we should assume that a site administrator will not make harm to his own users, but that's exactly the point in this case. An

administrator who could simulate a valid SM would be able to control activity of processes belonging to **other users** on **other sites.** This is obviously not acceptable, since we cannot demand site administrators to trust administrators of other sites.

Solution – secure VMS extension protocol. Conceptually, the solution is simple: since only a member of a Virtual Monitoring System is allowed to send requests related to the application associated with this VMS, we should secure the protocol of extending a VMS. Obviously, we should grant the membership in a VMS only to those SMs which are involved in monitoring of the application associated with the VMS.

This is possible, since the current procedure of joining a VMS is bottom-up: (1) the request to join comes from a LM to the SM; (2) the SM contacts other SMs, already in the VMS; (3) when all the SMs of the VMS share the information about a new SM, the VMS is extended. Thus, we can use a digital signature to **verify if the SM requesting membership in the VMS previously received the appropriate request from a LM.**

Thus, the extended registration message should be prepared by a Local Monitor in the following way. The registration message received from the application process should be concatenated with SM's Distinguished Name, and signed with the user certificate. Note, that the DN of the Service Manager is known by the LM, since the GSI secure channel is established between them before any request is sent. Such a **extended register message** is an irrefutable proof that the **user wishes** this SM to be attached to the Virtual Monitoring System.

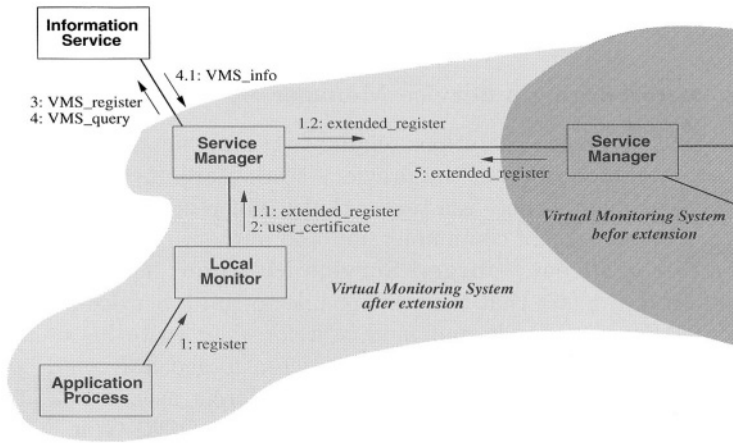


Fig. 2. VMS extending protocol.

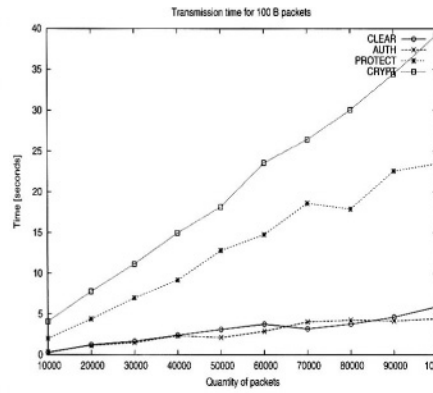
The secure protocol of the Virtual Monitoring System extension is presented in Fig. 2. The protocol is as follows: **(1)** Application process sends `register` message to Local Monitor (message 1). **(2)** Local Monitor connects Service Manager

(SM1) and when the connection is established, it knows the Distinguished Name of SM1. (**3**) The LM prepares `extended_register` message and sends it to SM1 (message 1.1) with the user's temporary certificate (message 2). The certificate is necessary to verify user's signature. **(4)** SM1 registers in an information service (message 3) and obtains information about other Service Managers involved in the application (VMS) (messages 4 and 4.1). **(5)** SM1 requests connections with other SMs from the VMS. After the connections are established, SMs exchange their `extended_register` messages (messages 1.2 and 5) and update their information about the VMS.

The main principle of security solution introduced above is as follows. (1) a Service Manager, to join a VMS, should present a *written permission* of the application owner to other SMs belonging to VMS. (2) a SM joining VMS should extend his VMS data structures with information about other SMs only if the other SMs send it a message with user's *written permission,* issued for these SMs. The *written permission* is issued by the use of asymmetric cryptography functionality – it is a digitally signed register request.

Table 1. Results of the security overhead test.

	Transmission time [s]			
Quant.	CLEAR	AUTH	PROT.	CRYPT
10000	0.27	0.34	2.00	4.09
20000	1.21	1.14	4.38	7.78
30000	1.66	1.49	6.97	11.14
40000	2.40	2.33	9.17	14.95
50000	3.10	2.12	12.81	18.16
60000	3.74	2.87	14.75	23.54
70000	3.17	4.03	18.60	26.41
80000	3.75	4.24	17.90	30.04
90000	4.62	4.15	22.56	34.43
100000	5.91	4.44	23.47	39.21
AVG[ms]	0.0530	0.0448	0.2357	0.3826



5 Security Overhead Test

Since OCM-G is designed to be on-line monitoring system, the overhead introduced by its architecture should be minimized. Therefore, in order to verify proposed solution test of the overhead caused by the proposed security solution was prformed. The *forge-component* security aspect does not cause significant monitoring overhead, since it concerns connection establishment and does not affect data transmission.

The experiment consists in transmitting data through the network between two processes running on different hosts. We measured CPU time required for

transmission different quantities of l00b packets with different security levels: **CLEAR** - no security aspects; **AUTH** - authentication and authorization of the peer; **PROT.** - data consistency ensured; **CRYPT** - data confidentiality ensured. Next security level includes all aspects of the previous. The sender process was executed on host supplied with Intel Celeron 300 MHz CPU and the responder on host supplied with Intel Pentium III 600 MHz CPU. Hosts were connected with 100MB swithed LAN. The results are presented in Tab 1.

We can notice linear relationship between CPU time and quantity of packets for all security levels, however for higher security levels CPU time increases faster. Since the difference between overhead caused by the **PROT.** and the **CRYPT** security levels is significant we should consider introducing lower security level to be used in more secure network environments by the OCM-G. In order to estimate the overhead resulting from proposed solution we present average transmission time for discussed security levels.

6 Summary

In this paper, we have shown a concept of the security solution for the OCM-G Grid application monitoring system. Communication between OCM-G components is secured with methods introduced by *Grid Security Infrastructure.* Thus, secure channel between them can be established, to avoid network protocols vulnerabilities. A second aspect of the OCM-G security is connected with components that are shared between users and applications. We have shown, that by the use of forged components, it was possible for site administrators to control processes of users on remote sites. To solve this problem, we introduced a secure protocol of adding new OCM-G components to existing *Virtual Monitoring System.* The protocol uses asymmetric cryptography features to ensure that the VMS is extended only on a written permission of the user.

Overhead test revealed CPU time for encrypted transmission is more then seven times greater then for insecure connection. However we should realize it is of tenth part of miliseconds, thus even the most CPU time consumptive connection should not cause significant monitoring overhead for the OCM-G.

Acknowledgement. We are very grateful to Dr Włodzimierz Funika for many discussions and helpful remarks.

References

1. Adams, C., Farrell, S. *Internet X.509 Public Key Infrastructure Certificate Management Protocols,* RFC 2510
2. Baliś, B., Bubak, M., Szepieniec, T., Wismüller, R., and Radecki, M.: OCM-G – Grid Application Monitoring System: Towards the First Prototype. Proc. Cracow Grid Workshop 2002, Krakow, December 2002.
3. Bellovin, S., *Security Problems in the TCP/IP Protocol Suite,* Published in Computer Communication Review vol. 19, no. 2 (April 1989) pages 32-48; `http://www.research.att.com/~smb/papers/ipext.ps`

4. Bellovin S. *Defending Against Sequence Number Attacks,* RFC 1948
5. The CrossGrid Project, http://www.eu-crossgrid.org
6. Foster, I., Kesselman, C., Tuecke, S. *The Anatomy of the Grid,* To appear: Intl J. Supercomputer Applications, 2001
7. GSI working group homepage: http://www.ggf.org/security/gsi/index.htm
8. Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A.: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series, (1997)
http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz
9. Menezes, A., van Oorschot, P., Vanstone, S. *Handbook of Applied Cryptography* CRC Press, 1996 http://www.cacr.math.uwaterloo.ca/hac/

Mediators in the Architecture of Grid Information Systems*

Peter Brezany[1], A. Min Tjoa[2], Helmut Wanek[1], and Alexander Wöhrer[1]

[1] Institute for Software Science
University of Vienna, Lichtensteinstrasse 22, A-1090 Vienna, AUT
{brezany|woehrer}@par.univie.ac.at, wanek@chello.at
http://www.par.univie.ac.at/~brezany/
[2] Institute for Software Technology and Multimedia Systems
Vienna University of Technology, Favoritenstrasse 9-11/188/2, A-1040 Vienna, AUT
tjoa@ifs.tuwien.ac.at
http://www.ifs.tuwien.ac.at/~tjoa/

Abstract. Across a wide variety of fields, huge datasets are being collected and accumulated at a dramatical pace. The datasets addressed by individual applications are very often heterogeneous and geographically distributed. In this paper, we describe our extensions and improvements to the reference implementation of the OGSA-DAI Grid Data Service prototype in order to provide a Virtual Data Source – a clean abstraction of heterogeneous/distributed data for users and applications. By picturing general applicable access scenarios we are showing the great need for such a Grid data mediation service as well as the compliance with important requirements of virtual data sources.

1 Introduction

Grid computing at its core enables devices – regardless of their operating characteristics – to be virtually shared, managed and accessed across an enterprise, industry or workgroup. This virtualization of resources places all of the necessary access, data and processing power at the fingertips of those who need to rapidly solve complex business problems, conduct compute-intensive research and data analysis, and engage in real-time.

The World Wide Web began as a technology for scientific collaboration and was later adopted for e-business. Scientists foresee - and indeed, we are experiencing - a similar trajectory for Grid technologies [5]. Many research funding organizations (e.g. e-Science Programme in U.K.) and commercial companies (e.g. IBM) are driving the benefits of Grid computing beyond its academic and research roots into business enterprises. These enterprises demand an infrastructure and tools for data management and analysis. This implies interfaces for federating databases [24] and techniques for metadata generation and management alongside other data issues.

* The work described in this paper is being carried out as part of the research projects "Modern Data Analysis on Computational Grids" and "Aurora" supported by the Austrian Research Foundation.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 788–795, 2004.
© Springer-Verlag Berlin Heidelberg 2004

A wide variety of major e-Science applications [20,11,4] are supported by the Globus Toolkit [9], a community-based, open architecture, open source set of services and software libraries. The development of Open Grid Services Architecture (OGSA) technical specification is ongoing within the Global Grid Forum inside the tasks called the *Open Grid Services Infrastructure (OGSI).* The Globus project is developing the *Globus Toolkit 3.0 (GT3),* which is based on OGSI mechanisms; the first implementation, *GT3,* is already available.

In recent years, the focus has shifted from compute-intensive tasks to more data-intensive applications, where significant processing is done on very large amounts of data. According to the analysis reported in [23], there is a dearth of Grid applications that use databases to store scientific data - almost all existing applications use files. However, if the Grid is to support a wider range of applications, both scientific and commercial, then database integration into the Grid will become important. Therefore, within the context of OGSA activities, the Global Grid Forum Database Access and Integration Services (DAIS) Group developed a specification for a collection of OGSI-compliant Grid database services. The first implementation [21] of the service interfaces, *OGSA-DAIS Release 3,* is already available.

The next logical step is the support for federating data resources, as depicted in Fig. 1, which is vital to the success of the Grid. The alternative of forcing each application to interface directly to a set of databases and resolve federation problems internally would lead to application complexity, and duplication of effort. The factors that make Grid database federation different include for example high dynamic flexibility, extreme performance, and semantic aspects [1].

The central component of Fig. 1 is the *Mediator,* which is realized as a special Grid Service (GDMS) exposing a relational database and an XML database to various applications as one cohesive data repository. By means of the *SDE* (Service Data Element)[1] and *perform* ports [6] the applications can query the mediation service features (metadata), status information and the database data, respectively.

This paper describes the design and implementation of one of the first mediation systems for databases integrated into the Grid. Our contribution significantly leverages the functionality of the OGSA-DAI reference Grid Data Service implementation.

The remaining part of the paper is organized as follows. In Section 2 we are delineating why there is a great need for data mediation on the Grid and what important requirements have to be fulfilled by the mediators. The kernel part of the paper is Section 3, which introduces the OGSA-DAI reference implementation and our extensions and improvements to this architecture. Section 4 discusses related work whereas Section 5 briefly outline the future work. The paper is closed with our conclusions in Section 6.

2 Data Access Scenarios

Our example access scenario is derived from the domain of health care. Let's suppose you want to mine data sources from different sites as depicted in Fig. 2. The data of the two involved hospitals is distributed over the three departments A, B and C. Although

[1] Information about Grid service instances, which is structured as a set of named and typed XML elements, encapsulated in a standard container format

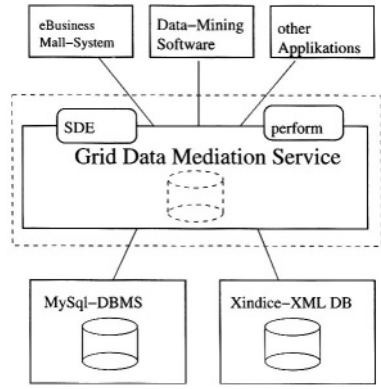


Fig. 1. Grid Data Mediation Service (GDMS) providing a *virtual data source* (VDS) for different applications, handling and hiding the heterogeneity of the two involved databases

the two hospitals store the same information about their patients, the data structures are different. Let's assume that the name information of hospital one is represented by the patient's full name, and the name information of an patient at hospital two is divided into first name (fn) and last name (In). Hospital one has *no central* patient database and so the data is divided and stored in two different databases - the administrative info in the administration department database (in Fig. 2 called site A) and the medical information in the care unit database (in Fig. 2 called site B). The other informations provided like date of first treatment, day of birth and adress have the same structure but are accessible over different names. To *materialize* the virtual data source (i.e. to reconstruct it from its fragments), the following operations are required: R = (A JOIN B) UNION C.

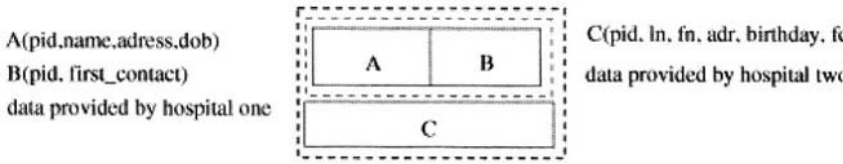


Fig. 2. Vertical and horizontal fragmentation of the virtual data source over three departments – relation A and B at the first hospital and relation C at the second hospital

Each data resouce understands the GDMS queries and returns the results in XML format – which are afterwards merged/processed with the help of the given mapping schema in Fig. 4.

It is clear, that *virtualization* almost always involves a loss of data access performance. Since many applications may use the Grid primarily for high performance, the mediator is discardable. Virtualized access can be provided, but is not the only behavior.

An application that wants high performance is able to directly access the underlying sources by requesting a GDS for every data source, e.g., in order to apply optimizations specific to a particular data format.

With our architecture described in Section 3, we meet the concerns of the following important requirements of virtual data sources [1]: (a) When more than one data resource is specified, the Grid must provide the ability to link them together, even if they have different data structures, to produce a single logical target that gives consistent results. (b) When linking data resources, the Grid must provide the ability to use data in one resource as the matching criteria or conditions for retrieving data from another resource. (c) The Grid must be able to construct distributed queries when the target data resources are located at different sites, and must be able to support heterogeneous and federated queries when some data resources are accessed through different query languages.

3 Architecture of the Grid Data Mediation System (GDMS)

The GDMS architecture is illustrated in Fig. 3. The left hand side shows the simplified functional structure of the OGSA-DAI release 2 prototype. As one can easily see, it provides three Grid Services: a GDSR (Grid Data Service Registry), GDSF (Grid Data Service Factory) and a GDS (Grid Data Service).

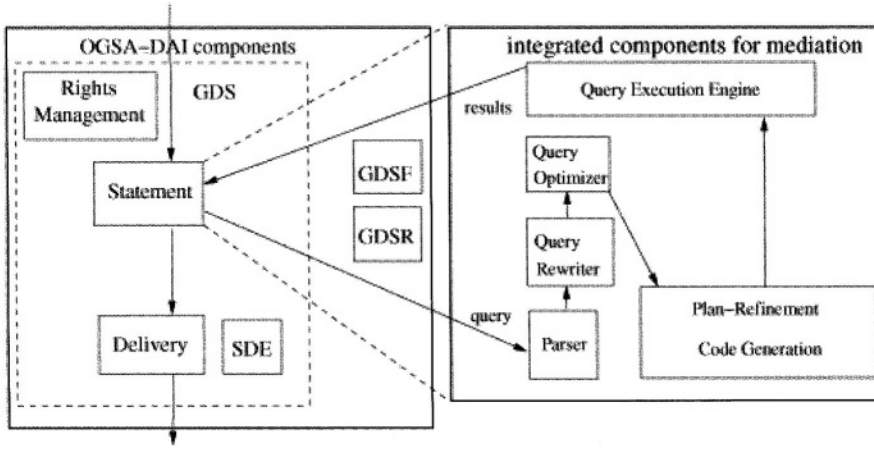


Fig. 3. In the left part of the picture a simplified structure of the current OGSA-DAI architecture is described, in the right the integrated GDMS modules replacing the standard statment handling

A persistent *GDSR* provides a searchable directory facility for OGSA-DAI services. A service can be registered via a Service Handle[2], together with information describing the service and its capabilities.

[2] A globally unique name

The persistent *GDSF* provides a service creation facility, creating GDSs which facilitate access to a particular data resource. Via a configuration file can be specified which GDSRs to use and to what DBMS and database/collection the GDSs created by the GDSF will interact. This file contains some of the *metadata* needed to interact with the data resource like the physical and logical schema of the data resource – also the *mapping schema* (for example, the schema in Fig. 4 needed for our scenario in Section 2) for describing the mediation task and characteristics of the VDS can be stored there for very complex or often used mediation tasks. For highly dynamic federations, this metadata can also be passed to the GDSF at runtime.

The transient *GDS* is the primary OGSA-DAI service. GDSs provide access to data resources using a document-oriented model via an XML document called GDS-Perform document – which allow detailed instructions for the Delivery activity in Fig. 3 to specify how the results (e.g. via output stream or GridFtp) or status (e.g. completed or detailed error messages) of the operation should be returned to the client or a third party, to be included. When a GDS is created, it is configured by a GDSF using the already mentioned GDSF configuration file which specifies the data resource the GDS will interact with and the data retrieval/update operations the GDS can perform. The *Rights Management*

```
<Table name="patient">
  <union>
    <join>
      <source key="pid" id="A">
        <map dest="p_id" src="pid"/>
        <map dest="p_name" src="name"/>
        <map dest="p_adr" src="adress"/>
        <map dest="p_dob" src="dob"/> </source>
      <source key="pid" id="B>
        <map dest="p_id" src="pid"/>
        <map dest="p_fc" src="first_contact"/> </source>
    </join>
      <source key="pid" id="C">
        <map dest="p_id" src="pid"/>
        <map dest="p_name" src="concat(fn,ln)"/>
        <map dest="p_adr" src="adr"/>
        <map dest="p_dob" src="birthday"/>
        <map dest="p_fc" src="fc"/> </source>
  </union>
</Table>
```

Fig. 4. Mapping schema of the table for the example scenario given in Fig. 2

is supporting the concept of *Virtual Organizations* (VOs) where you can define access rights to the data sources individually for each VO.

With the help of *SDEs* of a Grid Service, the metadata for the service instance is provided. This includes static (e.g. supported activities) and dynamic (e.g. status of request) information about the service.

For the integration of the results from the different data sources, our GDMS uses a *mapping schema,* which defines mapping information between the virtual data source and the participating data sources. The XML instance outlined in Fig. 4 is an example for a mapping schema in order to mediate the data resources given in our example scenario in Section 2. Each element *Table* represents the mapping information for a virtual table contained in the *VDS*. This element contains other subelements such as *union* and *join.* Union operations are defined by enumerating the fragments to merge in the *union* element, which are in our example the results of a join operation and the results of a projection. The *join* element contains the information required for the join operation. The participating data sources are kept in the *source* elements with their key names.

On the right side of Fig. 3, the applied *textbook architecture,* we are using for our mediator, is illustrated. This architecture was used, for example, in the IBM's Starburst project [15] and can be used for any kind of database system including centralized, distributed, or parallel systems [18]. Below we briefly describe the main components of this architecture.

Parser. In the first phase, the query is parsed and translated into an internal representation (e.g. a query graph) that can be easily processed by the later phases.

Query rewrite. It transforms a query in order to carry out optimizations that are good regardless of the physical state of the system. Typical transformations are the elimination of redundant predicates or simplification of expressions. In our system, where data is distributed, this part also selects the partitions of a table that must be considered to answer a query. With the help of the *mapping schema* given in Section 3 and the extended *logical schemas* stored (for each data source) in the GDSF, the basic part of the mediation is done.

Query Optimizer. This component carries out optimizations that depend on the physical state of the system. E.g. the optimizer is able to choose another *replica* (if specified in the GDSF) when one replicated data resource isn't responding in order to materialize the VDS anyhow.

Plan Refinement/Code Generation. This module transforms the plan produced by the optimizer into an executable plan.

Query Execution Engine. This element provides generic implementations for every operator. Our query execution engine is based on an iterator model [10]. In such a model, operators are implemented as iterators and all iterators have the same interface. As a result, any two iterators can be plugged together (as specified by the consumer producer relationship of a plan), and thus, any plan can be executed. Another advantage of the iterator model is that it supports the pipelining of results from one operator to another in order to achieve good performance.

4 Related Work

Basically the work presented here addresses integration of Grid database sources and database mediation. Grid database access is being developed and researched by the DAIS-WG [23] in close cooperation with the OGSA-DAI project [22].

Database mediation has been studied for quite a while [26,17], where recently also seems to be a trend to use XML either to transport queries and/or query results or to describe database schemas and even to store the data itself [2,19].

Optimization of query execution for parallel and distributed databases [18] is also an extremely relevant topic. Since Grid Services are distributed over the Internet which mostly offers very indeterministic connections (speed of data transfer, reliability, etc.), adaptive optimization strategies seem promising [16].

The GDMS presented here is also intented to build a basis for future Grid data mining projects – a survey of parallel and distributed data mining can be found in [27]. And finally semantic models and ontologies could be used to automatically generate mappings between schemas of different and heterogeneous databases [13].

5 Future Work

A prototype implementation of the GDMS for horizontal partitioning has been developed to prove the feasibility of the described concept. To increase the usability and performance of our mediation system our future work includes the following research agenda: allow more complex and colorful user queries; refining the *mapping schema* to support mediation tasks as e.g. *outerjoin* and *union all;* include more sophisticated methods for *replica* selection; and extend the developed concepts to a *parallel mediation service.*

6 Conclusions

In the described research effort, we focus on the application and extension of the Grid technology to the mediation of datasources. Our extendable and flexible system architecture provides a clean abstraction of heterogeneous data for users/applications, supports a wide variety of data formats and is useable with a subset of SQL92. By providing different views of data, pre-defined or determined at run-time, according to various applications needs, it fulfills requirements for high dynamic and flexible data access.

References

1. Atkinson, M.P., Dialani, V., Guy, L., Narang, I., Paton, N.W., Pearson, D., Storey, T., Watson, P.: Grid Database Access and Integration: Requirements and Functionalities. (2003) Global Grid Forum 7.
2. Baru, C., et al.: XML-based information mediation with MIX. SIGMOD '99 (1999)
3. eScience Centre, N. (http://umbriel.dcs.gla.ac.uk/NeSC/general/)
4. EU: Datagrid project, (http://eu-datagrid.web.cern.ch/eu-datagrid/)
5. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. Intl. J. Supercomputer Applications, 15(3) (2001)

6. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
7. GGF-DAI-WG. (http://www.cs.man.ac.uk./grid-db/)
8. Global Grid Forum: (http://www.globalgridforum.org)
9. Globus: Toolkit, (http://www.globus.org/toolkit/)
10. Graefe, G.: Query evaluation techniques for large databases. ACM Computing Surveys (CSUR) **25** (1993) 73–169
11. Grid-Physics-Network, (http://www.griphyn.org)
12. Hong, N., Krause, A., Malaika, S. McCane, G., Laws, S. Magowan, J., Paton, N.W., Riccardi, G.: Grid Database Service Specification. (2003) Global GF 7
13. Hull, R.: Managing semantic heterogeneity in databases: a theoretical prospective (1997)
14. IBM: Research. (http://www.research.ibm.com)
15. IBM: Starburst project. http://www.almaden.ibm.com/cs/starwinds/starburst.html(1984-1992)
16. Ives, Z.G., Florescu, D., Friedman, M., Levy, A., Weld, D.S.: An adaptive query execution system for data integration (1999)
17. Josifovski, V., Risch, T.: Comparison of amos ii with other data integration projects (working paper) (1999)
18. Kossmann, D.: The state of the art in distributed query processing. ACM Computing Surveys (CSUR) **32** (2000) 422–469
19. Lee, K., Min, J., Park, K.: A Design and Implementation of XML-based mediation Framework(XMF) for Integration of Internet Information Resources. In: Proceedings of the 35th Hawaii International Conference on System Sciences - 2002. (2002)
20. NASA: Information power grid, (http://www.ipg.nasa.gov/)
21. OGSA-DAI: Project release page, http://www.ogsadai.org (2003)
22. OGSA-DAI: Distributed query processor, http://www.ogsadai.org/dqp/ (2003)
23. Paton, N.W., Dialani, V., Storey, T., Atkinson, M.P., Pearson, D., Watson, P.: Database Access and Integration Services on the Grid. (2002)
24. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys **22** (1990) 183–236
25. Stockinger, H.: Database replication in world-wide distributed data grids. PhD Thesis, University of Vienna (2001)
26. Wiederhold, G.: Mediators in the architecture of future information systems. The IEEE Computer Magazine (1992)
27. Zaki, M.J.: Parallel and distributed association mining: A survey. IEEE Concurrency **7** (1999) 14–25

Towards the Wisdom Grid: Goals and Architecture*

Ivan Janciak[1], Peter Brezany[1], and A. Min Tjoa[2]

[1] Institute for Software Science
University of Vienna, Liechtensteinstrasse 22, A-1090 Vienna, Austria
{janciak|brezany}@par.univie.ac.at
[2] Institute for Software Technology and Multimedia Systems
Vienna University of Technology, Vienna, Austria
tjoa@ifs.tuwien.ac.at

Abstract. Multi agent systems, Grid technology, Semantic Web, and Web Intelligence paradigm are three modern approaches in information technologies, which we link in the research effort described in this paper to create a new-generation infrastructure called the *Wisdom Grid* with the mission to maintain, share, discover, and expand knowledge in geographically distributed environments. The paper introduces motivating ideas for this project, proposes the system architecture of one instance of the *Wisdom Grid,* and describes its functionality by means of a case study of one medical application.

1 Introduction

1.1 Motivation

The Web has significant impacts on both academic and ordinary daily life. It revolutionizes the way in which information is gathered, stored, processed, presented, shared, and used. Moreover, the Web provides the infrastructure for the Grid an emerging platform to support on-demand "virtual organizations" for coordinated resource sharing and problem solving on a global scale [15]. The Grid is sometimes heralded as the next generation of the Internet or the Web. There are strong connections between Grid, the Internet, and Web developments, as will be discussed later in this section.

Early Grid efforts (early to mid 1990s) started as projects to link supercomputing sites; at this time this approach was known as *metacomputing*. The objective was to provide Web computational resources to a range of high performance applications. Today the Grid infrastructure is capable of binding together more than just a few specialized supercomputing centers. It is more ubiquitous and can support diverse applications requiring large-scale computation and data.

Grid technologies are currently evolving toward an Open Grid Services Architecture (OGSA) [9], in which the Grid provides an extensible set of services that virtual organizations can aggregate in various ways. OGSA development is a natural extension of the activities defining the standards for Web Services.

* The work described in this paper is being carried out as part of the research projects "Aurora" and "Advanced Data Analysis on Computational Grids" supported by the Austrian Research Foundation.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 796–803, 2004.
© Springer-Verlag Berlin Heidelberg 2004

As already mentioned, Grid computing began with an emphasis on compute intensive tasks which, while benefiting from massive parallelism for their computation needs were not data intensive; the data that they manipulate was not proportional to the computation performed. Later, this focus shifted to more data-intensive applications [6], where significant processing was done on very large amounts of data and recently several research projects also addressed knowledge discovery in large databases attached to the Grid [4, 1,5].

Meanwhile, *"a new generation of Web technology, called the Semantic Web, has been designed to improve communications between people using different terminologies, to extend to interoperability of databases, to provide tools for interacting with multimedia collections, and to provide new mechanisms for the support of "agent-based" computing in which people and machines work more interactively."* [3]. These ideas led Grid scientists to the notion of *Semantic Grid,* where they plan to apply Semantic Web technologies in Grid computing developments [11].

Web Intelligence (WI) is a new direction for scientific research and development that explores the fundamental roles as well as practical impacts of Artificial Intelligence (AI) and advanced Information Technology (IT) on the next generation of Web-empowered products, systems, services, and activities [17]. Our vision is that a similar research direction, correlated with WI research, is also needed in the Grid research domain. Therefore, this paper introduces an analogous paradigm, *Grid Intelligence,* as a basis for developing a new-generation information technology infrastructure, the **Wisdom Grid** (WG), which will allow the creation of Grid applications that will help people achieve better ways of living, performing scientific work, treating patients, working, learning, etc.

1.2 Motivating Scenario

We believe that the best way of grounding the subsequent discussion on the Wisdom Grid architecture, its design goals, functionality, and the techniques used is in terms of a scenario. To this end, we will use the following scenario, which is derived from discussions with health care professionals involved in treatment of patients with traumatic brain injuries (TBIs) [4].

A physician has a patient with a specific diagnosis and wants to know what should be done next in the treatment of this patient, what the possible risks are, what kind of drugs should be used, what outcome for this and similar patients can be assumed, etc. The physician (user) has several options to determine answers to these questions. Traditionally, he could discuss these issues with his colleagues, with a specialist via online chat, he could look into a patient database to see what developments are available from other patients with the same or similar symptoms and diagnosis, he could search the Internet for appropriate medical articles and reports on this problem, etc. However, in this way it is not possible to obtain urgent information, which may often be a very critical issue in many medical applications, like TBI, or in crisis management environments in general.

To solve the above problem, an advanced information technology support which fulfills a set of requirements is needed. These requirements include: (1) ability to access

and analyze a huge amount of information that is typically heterogeneous and geographically distributed; (2) intelligent behavior - ability to maintain, discover, extend, present, and communicate knowledge; (3) high performance (real-time or nearly real-time) query processing; and (4) high security guarantee. This combination of requirements results in complex and stringent demands that, until recently, could not be satisfied by any existing computational and data management infrastructure.

In tackling these problems, we have initiated a research effort to design and experimentally implement a novel infrastructure called *Wisdom Grid* (WG), which aims to fulfill the requirements outlined above. The WG concepts are mainly based on the following technologies: basic Grid [10], Data Grid [6], Knowledge Grid [2], Semantic Web and Grid [3,16], and agents [14].

Now a new information and knowledge acquisition style based on the cooperation of the physician (user) with the WG components can be briefly described by the following scenario.

We assume that the user has his own *Personal Agent* (P-Agent), which could be a specialized medical agent having detailed information about the user, his professional field, department specialization etc., or it could be a very simple agent that is only able to ask questions and provide the answers. The user sends such an agent, together with information about the problem, to our *Domain Agent* (D-Agent). These agents can only communicate if they understand each other, so first they negotiate about the communication language and the domain in which they are going to communicate. D-Agent informs P-Agent about the domain of interest it is able to answer, and P-Agent passes this information to the user. The user then creates a particular question and fills it in with real input data. When D-Agent receives this question from P-Agent, it tries to find the answer in its *knowledge base.* There can be an exact answer or just a reference to the resource (location) where the desired information is stored and how to retrieve this information from the resource. If the information is somewhere on the Grid (e.g. in a special medical database) then a specific Grid service, e.g. data mining service, is queried for this information.

A special *Knowledge Explorer Agent* (E-Agent) is submitted to ask another agent, for example specialized medical agent, who has required information or is able to search for it. The answer returned to P-Agent could also be a location of resources on the web or in a medical library where the user can find information about the problem. In addition, it could be an email address or a phone number of a specialist who has appropriate knowledge about the user's request. Finally, P-Agent receives complex information about the user's problem and it is up the P-Agent to represent this information to the user.

1.3 Organization of the Paper

The structure of the rest of the paper is organized as follows: Section 2 presents the architecture and functionality of the Wisdom Grid. The knowledge base organization is introduced in Section 2.3 and we briefly conclude in Section 3.

2 The System Design

In this section we describe the system architecture, outline the functionality of the components involved, and describe interactions between individual components.

Architecture. The architecture of our Wisdom Grid system is sketched out in Fig 1. The agents provide distributed intelligence services, which involve communication and decision making activities. The Grid is the basic infrastructure, which provides secure access to distributed data and knowledge resources.

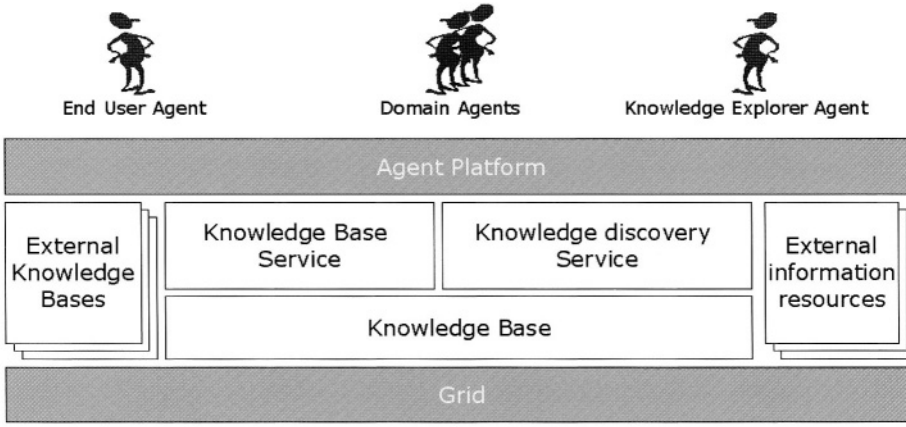


Fig. 1. System Architecture

WG services can be geographically distributed and can use all advantages of grid computing. Taken actions in knowledge discovery are invoked and controlled by internal agents which communicate with services and transport data between them. This agent manages and executes services in ways described in internal workflow scenarios.

2.1 Agent Platform

We decided to use agent technology [14] for our communication purposes. Agents build the communication aspect of our system and represent it to the outside environment. This part of the *Wisdom Grid* plays the main role in the interaction of agents and WG services. Agents' messages are transformed to exact actions and results of these actions are converted back to messages. This platform closely cooperates with the knowledge base service, which is always queried for information about all the services and resources that can be used in the search for knowledge. FIPA Standards [8] are used for the agent life cycle and two types of agents are managed by this platform:

a) **Domain Agent.** *(D-Agent)* These agents represent the system to the outside world and they are mediators between knowledge demands and knowledge provided by WG. From the outside world point of view, they are intelligent

software agents specialized for a particular domain of knowledge stored in the knowledge base. This part of the knowledge base is used as agent's ontology with which it presets itself to other agents submitted to query WG for knowledge. Those personal agents (P-Agents) are able to address questions to the D-Agents and present answers to the user. The success of this activity strongly depends on the mutual understanding of the P-Agent and D-Agent and also on the query construction technique. P-Agent is not part of the system because it is created by the user and is not registered on our agent platform. It can ask the D-Agent for the list of domains, about which it has knowledge, or P-Agent can inform the D-Agent about the ontology, which describes its domain and merges it together with D-Agent's ontology to select one common domain.

b) **Knowledge Explorer Agent.** *(E-Agent)* This agent is used by the WG in the process of searching for other knowledge providers and extends the knowledge base with information about services or resources which can be later queried by the Knowledge discovery service. The agent is able to query other agents and also to search the Semantic Web [3], which can be considered as an external knowledge base.

2.2 Knowledge Discovery Service

This service acts as a client of external information resources or uses prepared clients or agents in communication with them. It provides external services with input data and returns their output. The knowledge base service supports the knowledge discovery service with information about resources. Such information usually consists of location, type of resource, communication protocols, and everything needed for successful cooperation between them. Semantic markup for web services (DAML-S) [7] is used for this purpose. Knowledge discovery service uses instances of the DAML-S ontology to identify their profile, model and grounding. Resource or output data from it can be used as an input parameter for another external service. Therefore workflows from the knowledge base are used to fulfill this requirement. For example, references to databases integrated into the Grid or other databases provided by cooperating institutions (hospitals, research institutes, etc.) can be used as an input for external services.

External information resources. As external information resources we consider resources, which provide WG with some kind of useful knowledge. It could be a Web/Grid service, web page, agent, database, online library, etc. We mainly focus on Web/Grid services able to perform knowledge discovery (advanced data analysis like data mining, Online Analytical Processing (OLAP), etc. [12]). Locations and descriptions of external services are stored in the knowledge base.

2.3 Knowledge Base Service

This service operates over the knowledge base. Its function is to parse, search, query and extend knowledge base. It supports the Knowledge Discovery service with data about information resources. It uses a reasoner to determine which resource meets the query restrictions. The DAML Query Language (formal language and protocol for querying

repositories) is used for this purpose. The E-Agent extends this service with the ability to search in external, geographically distributed knowledge bases and also in the Semantic Web, often presented as global knowledge base.

Knowledge Base *is a database that stores particular data about real objects and relations between these objects and their properties in particular domains.* The knowledge base mainly stores data about information resources; what they do, how they work, how to query them and what knowledge they produce. The knowledge base is divided into individual domains of knowledge provided by WG. Every domain is associated with a D-Agent. This agent can operate over more then one domain. Ontologies are sets of associated slots of classes, their properties, relations, and restrictions that describe individual domains. It means that the ontology provides exact consequence to the objects stored in the knowledge base. The ontology is written in the DAML+OIL [13] language and for better search and query performance it is stored in the database. Ontologies in the knowledge base are used to describe semantics of the information resources. For example a real database of patients in a hospital is an instance of a database ontology, and on the other hand, it is also an instance of the ontology defining patient class and his properties. This fact that one object can be an instance of more ontologies is useful in the process of mapping between real names and their semantic. It can be used, for example, in the process of integrating different databases (different in structure, names, location etc., but having the same semantics).

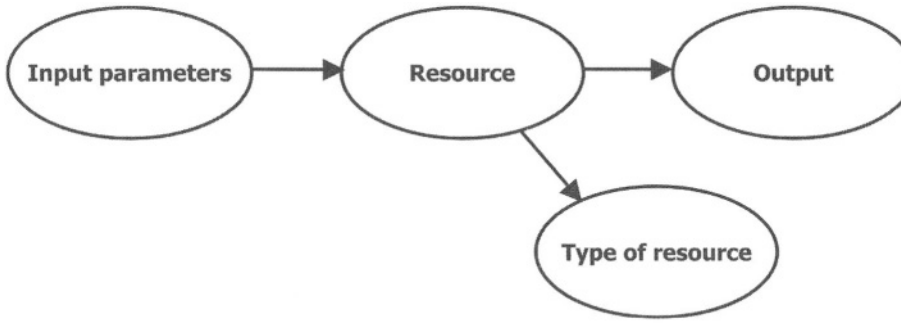


Fig. 2. Top ontology of the information resource

The knowledge base can be extended manually by a domain expert, a person who understands the context of a particular domain, or automatically by the knowledge base service which takes into account knowledge founded by the Knowledge discovery service or by the E-Agent. Such information is usually a location of a knowledge resource and its description. The knowledge base also stores semantic descriptions of data used as an input and output parameters for services and information about quality of service. From this point of view the knowledge base can simply be considered as a semantic registry of the services and other information resources.

The top level information resource ontology is depicted in Fig. 2. This high level abstraction simply says that the resource need input parameters and gives some output. It can be used for simple dynamic web pages, databases as well as for Grid services. For example, the service at *http://gridminer/decision-tree.gwsdl* with input parameters *jdbc://hospital/db, user=test, table=test,* and *class=outcome* gives a decision tree as an output located at *http://gridminer/decision-tree/tree.xml,* or another example: the resource *jdbc://hospital/db* returns a list of patients if input is query: *"SELECT patients FROM test"*. Detailed descriptions of resources and properties are specified in the DAML+OIL language and for every type of resource, specialized templates exist to express resource requirements and properties in some goal achievement process.

The knowledge base also contains internal information about WG services, their detailed description, and their functionality and ability in processing information flows. For this purpose workflows diagrams, also stored in the knowledge base and defined by specialized ontology, are used.

External Knowledge Bases. For external knowledge bases, we consider ontologies available on the web and freely accessible. We assume that the advent of the Semantic Web will bring the ontologies of many different domains as its components. Therefore, the E-Agent is proposed, which is able to search widely distributed knowledge bases.

3 Conclusions

This paper has outlined our vision of the Wisdom Grid as a future infrastructure for advanced applications. We have also presented an instance architecture of the Wisdom Grid, described its main components and the structure of the services as well as the concepts of the Wisdom Grid knowledge base in the context of a medical application example. The prototype implementation of the knowledge discovery services is currently in progress. The future work is to create WG portal to simplify cooperation between the real user and his agent through web browser.

References

1. Cannataro, M., Talia, D., Trunfio, P.: Design of distributed data mining applications on the KNOWLEDGE GRID. Proceedings NSF Workshop on Next Generation Data Mining Baltimore (November 2002) 191–195
2. Cannataro, M., Talia, D.: KNOWLEDGE GRID: An Architecture for Distributed Knowledge Discovery. Communications of the ACM (January 2003) 89–93
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
4. Brezany, P., et al.: Knowledge grid support for treatment of traumatic brain injury victims. Proceedings of the Conference ICCSA 2003 Montreal (May 2003)
5. Cannataro, M., Talia, D., Trunfio, P.: Distributed data mining on the grid. *Future Generation Computer Systems.* (2002) 1101–1112
6. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. (1999)
7. The DAML Services Coalition: DAML-S: Semantic markup for web services. http://www.daml.org/services (2002)

8. Foundation for Intelligent Physical Agents: Fipa - agent management specification. http://www.fipa.org (2000)
9. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration. (2002)
10. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. Intl. J. Supercomputer Applications 15(3) (2001)
11. Goble, C., De Roure, D.: The semantic grid: A future e-science infrastructure. www.semanticgrid.org
12. Han, J.: Data Mining. Concepts and Techniques. Morgan Kaufmann (2000)
13. Horrocks, I.: DAML+OIL: a reason-able Web ontology language. in Proceedings of EDBT (March 2002)
14. Jennings, N., R., Wooldridge, M., J.: Agent technology : Foundations, Applications, and Markets. Springer-Verlag (1998)
15. De Roure, D., Baker, M., A., Jennings, N., R., Shadbolt, N., R.: The evolution of the grid. In F. Berman, A. J. G. Hey, and G. Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality,* John Wiley & Sons (2003) 65–100
16. De Roure, D., Jennings, N., R., Shadbolt, N., R.: The semantic grid: A future e-science infrastructure. In F. Berman, A. J. G. Hey, and G. Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality,* John Wiley & Sons (2003) 437–470
17. Zhong, N., Liu, J., Yao, Y., (eds.): *Web Intelligence.* Springer-Verlag (2003)

Automatic Flow Building for Component Grid Applications

Marian Bubak[1,2], Kamil Górka[1], Tomasz Gubała[1], Maciej Malawski[1], and Katarzyna Zając[1]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
{bubak.malawski,kzajac}@uci.agh.edu.pl,
{kgorka,gubala}@student.uci.agh.edu.pl
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
phone: (+48 12) 617 39 64, fax: (+48 12) 633 80 54

Abstract. The Grid application can be defined as a set of cooperating components connected together to achieve desired functionality. The task of finding and connecting such modules distributed throughout vast and frequently changing Grid environment can be difficult. We present the Application Flow Composer system, designed for generation of Grid application workflow description which uses information about the compounds available on the Grid. The system is built within Common Component Architecture (CCA) and may be used to compose applications designed with the same architectural model. As a registry containing Grid component definition documents, the OGSA Registry Grid Service was applied. The tests enclosed show system operation and its time performance.

Keywords: Grid computations, workflow composition, CCA, OGSA

1 Problem Outline

The Grid application may be described as a distributed computation which involves many heterogeneous components spread across the Internet and connected one with the other [4]. The workflow of such Grid application is a flow of remote method invocations. Such invocation occurs when one component tries to use some functionality delivered by another one. Hence, to fully define particular application one has to list all its compounds and specify every connection between them. This description may be submitted to some virtual machine capable of dispatching the whole on the Grid. There are already few environments in scientific community enabling this process. Among them are the *Open Grid Service Architecture* (OGSA) [4,12], built on the top of Web Services model, and the *Common Component Architecture* (CCA) [3], designed from scratch for scientific modular applications. In the CCA standard, there is a proposition of *Application Factories* [5] capable of dispatching whole CCA-based applications in specified CCA framework. In the OGSA environment there is a concept of GSFL language for workflow and dataflow description [11] accompanied by an engine capable of building desired application on the Grid.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 804–811, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The problem arises when the user or system building the Grid application description does not exactly know how to do this. This trouble can be caused by lack of precise knowledge of which components use or, provided one knows that, where to find instances of these components on the Grid. The former case forces the user to try many different combinations of workflow before the application can work, the other case makes him search manually in component Registries. We are proposing the system, called *Application Flow Composer* System (AFC System), which can reasonably aid either a human or an application in the task of constructing Grid application workflow. We have chosen the CCA workflow model because it is really suitable for the application composition purposes, and it is relatively mature, while the GSFL language for Grid Services is still in design phase.

2 Idea of Application Flow Composer for CCA Model

The CCA standard defines just three entities building any system: CCA Components, CCA Uses Ports and CCA Provides Ports [3]. The components, resembling the idea of compounds building application structure, are equipped with ports, either Uses Ports of Provides Ports or both simultaneously. The Provides Port is a set of methods implemented by a component which means that the component guarantees some functionality through this port. On the other hand, the Uses Port is a requirement of the component needing particular port in order to work properly. The main concept of application composition in CCA architecture is simple: the Uses Port of requesting component should be connected to appropriate Provides Port of providing component so the former can easily call methods of the later and use theirs output. In consequence, the method of workflow description is easy: a complete list of components building the application accompanied by a list of every connection between these components.

The AFC System is built in CCA technology, so its main modules are implemented as CCA components. Also, the model of application description used in this system is CCA-compliant. The main function of this system is workflow composition. An input to this method is a document generally describing the application, i.e. what components are starting and finishing the computation and, optionally, what connections are proposed. This description, uncomplete by definition, is called Initial Workflow Description (IWD) and is formed within XML standard. As an output, the AFC system builds another XML document, syntactically compatible with the input one, called Final Workflow Description (FWD). This one consists of a list of the application workflows which the AFC system was able to built. There can be described more then one resultant flow (if the initial requirements was general enough) as well as none, if the requested flow construction is impossible.

Having such final document the user is able to dispatch the application within some Grid-aware CCA framework (like the XCAT Framework [9]), using some facility capable of activating the application on the Grid using its description (like mentioned earlier Application Factory [5]).

3 AFC System Design

3.1 Static Structure

The system we propose must implement the functionality of constructing application workflows. This requirement imposes the need of some source of information about the current state of the environment, i.e. what components are currently available, what ports do they provide and where to find them on the Grid. There should be persistent link between the composer part of the AFC system and the informational part, providing the former with necessary data as frequently as it asks for it. In consequence, there are two main modules (see Fig. 1), the one responsible for workflow document composition (we call it *Flow Composer Component* or FCC) and the other, which should obtain every piece of information which can be used by the system (*Flow Composer Registry,* FCR). This module is called 'Registry' because its main function is to receive, contain and publish on demand the information about the application components available on the Grid. The FCR Registry cooperates with the Registry Grid Service from Globus Toolkit package. It exposes two CCA Provides Ports, *Component Lookup* Provides Port and *Component Registration* Provides Port which resemble two additional obligatory functions of the FCR Registry, component querying and component registration, respectively. The entities which are stored within this registry internally, are XML documents fully describing CCA components, called *Component Information Documents* (CIDs). It is a responsibility of external components providers to register such descriptions in the FCR.

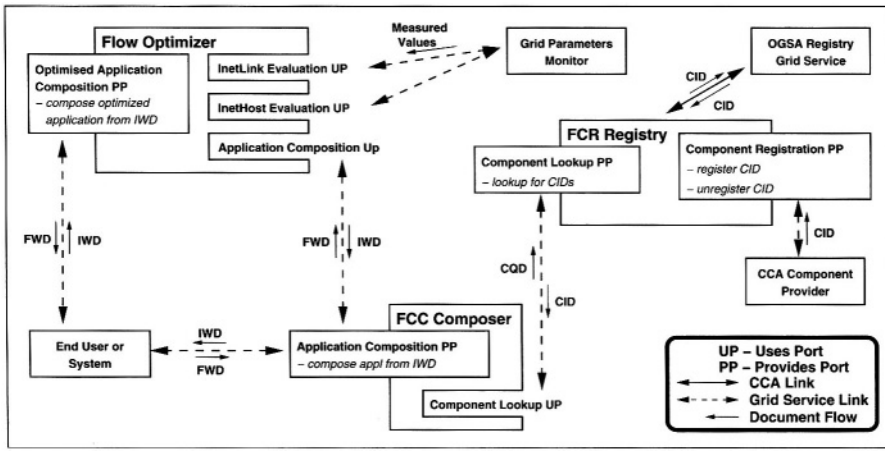


Fig. 1. Main components building AFC System

The FCC Composer has two ports, one of Provides Port type (*Application Composition* Provides Port) and the other of Uses Port type what indicates that it uses the functionality of the FCR Registry to obtain needed information. To

make the AFC system work the user has to dispatch one instance of these two components and connect them with one link (as shown in Fig. 1).

The third component, *Flow Optimizer* (FO), is not obligatory (the user can easily do without it by contacting FCC directly). The need for optimization component has arisen when it has become clear that the amount of flows produced by the FCC Composer can be enormous. We have decided to include such an optimizer which tries to choose better solutions and discard worse ones, hence the volume of output data is much smaller. The FO component decisions depend on the information it can achieve from some kind of Grid monitoring facility, measuring values important for Grid application performance; it could be hosts CPU performance or Internet links throughput (see Fig. 1). With this information at its disposal, the FO can pick the flows which hopefully reveal better performance then others.

3.2 Internal Flow of Documents

Apart of static aspect of the AFC system, Fig. 1 shows dynamic flow of documents. The only interaction between the AFC System and external entities is through its ports. The Application Composition port accepts IWD documents as an input to its main method, and returns final FWD documents (see Sect. 2). The ports of the FCR Registry accepts either CID documents while registering new components, or queries if an external system tries to obtain some information from the registry. These queries are standardized in XML schema called *Component Query Document* (CQD) and allow for requesting components of particular name or providing specified port. All of the documents being sent to the FCR are transported through CCA links, so overall picture stays consistent with this technology. The two later documents, CID and CQD, are also extensively used by composition algorithm, which achieves the information needed with the same method as any other system, by querying the registry.

The internal communication between the FCR Registry and the real database, incorporated into Registry Grid Service, uses the same CID document format. It is the document stored internally in Grid Service. It is the only link not being the CCA connection, but as long as it is truly internal mechanism, whole system is still CCA-compliant. The last communication shown in Fig. 1 is the link between the FO and some sort of monitoring facility. The idea is similar to that implemented with FCC-FCR pair. The monitor is crafted as a fully qualified CCA component, but *de facto* it is only some interface to real Grid monitor, which can be implemented in other technology (e.g. using Grid Index Information Service (GIIS) facility from Globus Toolkit [8]).

3.3 Composition Algorithm

This section briefly describes the application composition algorithm implemented within the FCC Composer. To explain it, let's follow usual scenario of flow composition (which is depicted in Fig. 2). The user submits an IWD document

which contains requirements of desired Grid application. Among other information there are specified crucial components for the flow (including so called *initial* and *final* components, starting and finishing the workflow). First, the composer searches the FCR for them to discover whether they are available. If so, it distinguishes every CCA Uses Port which is needed by the components and tries to find some providers of these ports. It can look for the provider either locally in the flow built so far or, if the provider was not found, it queries for it the FCR Registry. There is a possibility that both of this method will be used when the AFC System is instructed to search through full solution space, so *every* available provider will be found and investigated.

When providers have been discovered, they are installed into the flow. For every provider apart the first, the system must duplicate flows and incorporate different providers in subsequent solutions. This means that with full solution space search, the find of N providers will result with multiplication of the number of resultant flows by the factor of N. For every such newly constructed flow, the composer makes one step further, trying to find provider for the Uses Ports needed by the components found recently. This process lasts until every component has its providing peer and the components marked by the user in IWD document as *initial* ones (this means, no needing any provider to work), will be incorporated in the flow. The other means by which the composition can stop is the lack of required information in the registry, i.e. provider of some particular CCA Port was not found. In this case, the system suffers partial failure, but tries to return to the user as complete description as it is possible.

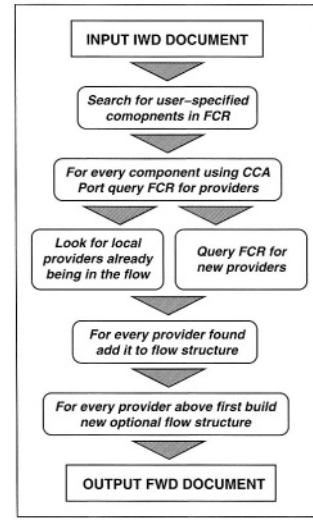


Fig. 2. Composition algorithm

It is clear that this composition method can produce very large amount of output data with many flows defined, so there are some restrictions which the user can impose on the composition process so it won't build such large numbers of flows. Further, the user can invoke the FO Optimizer in order to pick the most preferred solutions (as in Fig. 1).

3.4 Implementation

There are few implementations of the CCA technology (i.e. CCA frameworks): CCAFFEINE [1], SciRun2 [10], XCAT [9] and we have chosen to use the XCAT Framework 1.0. It has been the most mature project in the moment we had to choose, and, what is important, it is the only truly dedicated to totally distributed, Grid-based applications. Also, to make our future system CCA-compliant too, we have assumed that both modules of the system, the FCC

Composer and the FCR Registry, will be CCA components. The other issue has been to choose the registry capable of containing the CID documents within and being able to be an implementation of FCR Registry concept. Here, our choice has been the Open Grid Services Infrastructure (OGSI) technology with its widely known implementation, Globus Toolkit 3.0 [2]. Although it would be better to have the registry issue solution within CCA technology, there is no such implementation already available (actually, there is only sketch description of such entity, called *Component Browser* (see [5]), capable of registering and publishing CCA components information). To overcome this, we have decided to use Virtual Organization GS Registry available in Globus Toolkit 3.0, and we applied it as a Component Browser. In result, we have implemented the FCR Registry as a CCA component using Grid Service with Registry Port Type.

4 System Operation Test

4.1 Test Description

In order to show the way our system works a simple test was performed. We put six components into the registry and then we executed the AFC system composition method. It built four flows and we can watch now the specific cases of the flow creating process.

The components we inserted into the registry were chosen in the special way. When we ran our system it found four expected flows, as shown in Fig. 3. The rectangles represent the components. The inner squares are the Uses Ports (UP) and the outer ones are Provides Ports (PP). The number means the type of the port. The lines represent connections between the ports in the flows (found by the AFC System).

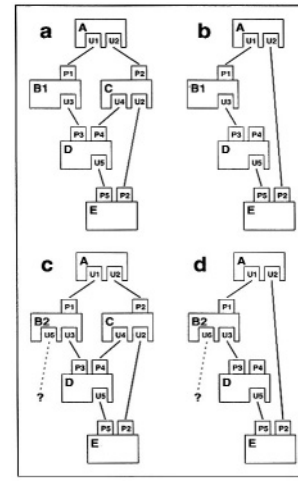


Fig. 3. Test overview - generated flows

The first flow (marked as **a** in Fig. 3) was the best one. All the ports were connected and it is complete. The second flow (diagram **b**) was also complete because there is no need to connect all the PPs (only the UPs must be connected because they determine the input data source for the component). Syntactically these two cases are equally good, so the user must choose the better of this two proposals for his purpose. It depends of the semantics of the components. Sometimes the user can adequately configure the FO optimizer so it can pick the better one automically.

The third and fourth flows (depicted as **c** and **d** respectively) are not completed since one of the UPs in the **B2** component was not connected to any of PPs. However, the user can make use of it if he knows the component which

provides that port on his own (it could simply not be registered within FCR registry). As we can see, there are two factors making the computation more complex: first, the component replication (as **B1** and **B2** pair) and the second, the port providers replication (here, providers of **P2** port). Both effects cause the growth of the number of generated flows.

4.2 Discussion of the Test Execution Performance

We executed the test several times on AMD AthlonXP 1466MHz. The overall average flow composing time was 3766 *ms*. The communication and registry searching took 2933 *ms* on average, thus the flow computing time was 833 *ms*. In our opinion, these times are reasonable. The real applications would consist of larger number of components as well as there would be more components in the registry which would make these times longer, but not very much. It is possible to reduce the number of generated flows and the registry calls what may accelerate the computations, however, the results may be worse.

Table 1 shows the times measured for our test. The querying time consists of communication between the FCR and the Registry Grid Service, and it also contains the time needed to parse some XML documents and to find the component in database. The communication time include passing the data between FCC and FCR and some XML parsing. Poor performance of registry searching is caused by communicating with Grid Service as well as document parsing and generating.

Table 1. Average computation, communication and querying times

Name	Time [*ms*]	%
Overall	3766	100
Communication	2933	78
Querying	2020	54
Computation	833	22

5 Summary and Possibilities of Future Development

The AFC system is capable of solving the workflow construction problem. Its performance is heavily dependent on the amount of data contained within the FCR Registry - the more data the FCC has at its disposal the more suitable flows it can build. Apart of this, there are also other use cases of the system: it can act as a CCA components registry for external systems. The way of querying for the CID documents is standardized enough to be usable by other applications as well as the method of registering new components. If the concept of Component Browser is not rejected, the FCR module can be of use as well defined registry.

The system is built on the top of two highly popular Grid architectures what guarantees that it can be of use in the future, especially when these two technologies get combined one with the other (as was proposed in [6]). The main advantage of the AFC system is its design. We have tried to make it clear and modular, so possible replacements and upgrades are easy to apply. Actually, there are some ideas of improvements. First, the FO component designed and

implemented so far is rather simple and its optimization technique is straightforward, so more sophisticated methods may be applied. Also some real solutions as the source of the Grid environmental data for FO should be applied (a candidate is the Grid Index Information Service (GIIS) [8] infrastructure). When the new XCAT Framework 2.0 [7] is published, the AFC system should migrate to integrate with Application Factories [5].

Acknowledgements. This work was partly funded by the European Commission, project IST-2001-32243, CrossGrid. We are very grateful to Mr Michał Kapałka for his contribution.

References

1. Allan B., Armstrong C., Wolfe A., Ray J., Bernholdt D., Kohl J.: The CCA Core Specification in a Distributed Memory SPMD Framework, Sandia National Lab.
2. Globus Toolkit 3.0 Alpha Version 3, March 2003, http://www.globus.org
3. Common Component Architecture Forum: The Common Component Architecture Technical Specification, version 0.5, http://www.cca-forum.org/documents
4. Foster I, Kesselman C., Nick J., Tuecke S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
5. Gannon D., Ananthakrishnan K., Krishnan S., Govindaraju M., Ramakrishnan L., Slominski A.: Grid Web Services and Application Factories. Indiana University, http://www.extreme.indiana.edu/xgws/af
6. Govindaraju M., Krishnan S., Chiu K., Slominski A., Gannon D., Bramley R.: Merging the CCA Component Model with the OGSI Framework, Indiana University http://www.extreme.indiana.edu/xcat
7. Govindaraju M., Krishnan S., Chiu K., Slominski A., Gannon D., Bramley R.: XCAT 2.0: Design and Implementation of Component Based Web Services, Technical report, Indiana University, June 2002
8. Grid Information Index Service, http://www-fp.globus.org/toolkit/information-infrastructure.html
9. XCAT Framework 1.0.1, http://www.extreme.indiana.edu/xcat
10. Johnson C., Parker S., Weinstein D., Heffernan S.: Component-Based Problem Solving for Large-Scale Scientific Computing, in *Journal on Concurrency and Computation: Practice and Experience on Concurrency and Computation: Practice and Experience,* 2002 No. 14 pp. 1337-1349
11. Krishnan S., Wagstrom P., von Laszewski G.: GSFL: A Workflow Framework for Grid Services, in Argonne National Lab., Preprint ANL/MCS-P980-0802, August 2002
12. Tuecke S., Czajkowski K., Foster I., Frey J., Graham S., Kesselman C., Vanderbilt P.: Grid Service Specification, version 4, Draft 4 October 2002

Grid Architecture for Interactive Applications*

Marian Bubak[1,2], Maciej Malawski[1], and Katarzyna Zając [1]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
{bubak.malawski,kzajac}@uci.agh.edu.pl
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
phone: (+48 12) 617 39 64, fax: (+48 12) 633 80 54

Abstract. This paper presents the architecture of the CrossGrid, i.e. the general overview of software components and their dependencies. The components are applications, tools that support application development and new Grid services that are being elaborated within the Project. Dependencies on external components, such as DataGrid and Globus software are presented as well.

Keywords: Grid, architecture, services, interactive applications, OGSA

1 Introduction

The CrossGrid Project [5] is oriented towards compute- and data-intensive applications which are characterized by the interaction with a person in a processing loop. Each application requires a response from the Grid to an action by a human agent in different time scales. These applications are: simulation and visualization for surgical procedures, flooding crisis team decision support system, distributed data analysis in high-energy physics (HEP) and air pollution combined with weather forecasting.

The interactive applications pose specific requirements for the Grid infrastructure. These requirements are addressed by the Grid services developed in CrossGrid. User-friendly access to Grid resources for remote users is handled by portals together with the Roaming Access Server and schedulers adapted for interactive applications. Grid monitoring services provide information on both infrastructure and application execution. Data Access services are used to optimize the time needed to transfer large data files from tertiary storage to computing nodes.

The creation of interactive Grid applications also calls for specialized tools that facilitate the process of code development and optimization. Such tools, perfected within CrossGrid, are Marmot for MPI verification, Grid Benchmarks for testing the behavior of applications in a Grid environment, a performance analysis tool (G-PM) that can perform on-line measurements on running applications and a performance prediction tool.

* This work was partly funded by the European Commission, project IST-2001-32243, CrossGrid

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 812–820, 2004.
© Springer-Verlag Berlin Heidelberg 2004

This paper presents the improved version of the architecture [1] of the CrossGrid Project. Dependencies on external components, namely DataGrid [6] and Globus software are presented as well.

2 Interactive Applications

2.1 Biomedical Application – Runtime Steering

The medical application is a distributed near-real-time simulation with a user interacting in virtual reality (VR) or other interactive display environments. A 3-D model of arteries is the input to a blood flow simulation. The medical application requires a distributed environment consisting of simulation, interaction and visualization components which will allow the user to change simulation parameters in near-real time. The problem-solving environment (PSE) consists of a set of Grid services which allow the setup and interactive steering of complex Grid applications with separate modules for simulation and visualization based on an existing standard, called the High Level Architecture (HLA) [11]. The use case scenario is as follows:

1. The user (physician) submits a request for access to Grid resources, via a Portal.
2. The 3D Visualization System starts, using the Grid Visualization Kernel (GVK) Grid Service to present previous simulation results.
3. The physician studies previous results and proposes new simulation criteria (i.e., selects bypass designs and updates selected parameters).
4. The physician submits new simulation criteria, via the Portal (i.e., once the new parameters are selected, the new job is submitted).
5. The Scheduling Agents (SA) service schedules a new simulation job taking into account the available Grid resources.
6. The Application and the GVK initialize simulation and visualization of the new job (i.e., once the new job is scheduled, simulation starts and updates are fed into the GVK for visualization and monitoring by the physician).
7. The Grid Performance Monitoring (G-PM) tool makes selected performance data accessible to a system administrator, via the Portal.
8. The physician accesses the simulation state using the portal.
9. The physician monitors ("monitoring" via GVK includes visualization) the progress and of the job and controls its execution via the GVK. He/she may change parameters, review results, and store comments as well as selected parameters in a local repository.
10. The physician stops the simulation.

2.2 Flood Forecasting – Cascade of Simulations

Flood forecasting starts with meteorological simulations conducted at different resolutions, from mesoscale to storm-scale. Selected hydrological models are then

used to determine water discharges from the affected area, and with this information hydraulic models simulate flow through various river structures. An interactive Grid system fulfilling the needs of this application should allow experts to prepare and control the cascades of meteorological, hydrological and hydraulic simulations basing on the assumption that each preceding step of the cascade produces input for the next simulation. The use case scenario is as follows:

1. The expert prepares a meteorological simulation, chooses boundary conditions (time/date), the model configuration file, and any parameters.
2. The expert prepares a hydrological simulation and adjusts the post-processing of meteo-output. Partial meteo-results (hourly files, 48 pieces) can be immediately post-processed and visualized (even before the entire meteo-simualtion finishes), so that hydrological simulations can be started.
3. The meteorological simulation executes. With scripts waiting for meteo-results, on-line visualization and/or hydrological simulations can start.
4. The expert checks if the simulation (meteo- and hydro-) results are valid, based on his/her knowledge and experience.
5. The expert chooses if the last simulation in the cascade (hydraulic-2D) is required. If so, the hydraulic simulation is started.

2.3 Interactive Particle Physics Data Analysis

Distributed data analysis in HEP addresses access to large distributed databases in the Grid environment and development of distributed data mining techniques suited to the HEP field. Data mining services based on supervised and unsupervised learning with processing on worker nodes and on the database side are being elaborated. HEP and air pollution modeling applications require support from a Grid interactive system that allows for on-line progress monitoring of their results in order to help operators decide about further job execution (i.e. interrupting the execution or letting it finish). The use case scenario is as follows:

1. The user submits a request for access to Grid resources, via the Portal.
2. The relevant dataset is selected from a list built from the information in the Metadata catalogue.
3. The Interactive Session Resource Broker (ISRB) asks the Replica manager to provide the optimal location of the servers with the requested Dataset.
4. The physicist submits the relevant variables to Grid middleware, via the Portal (i.e., once new parameters are selected, the job is submitted).
5. The Scheduling Agents (SA) service schedules the new simulation job taking into account the available Grid resources.
6. The Interactive Session Manager (ISM) receives the XML input from the UI and sends a simple distributed job that fills a histogram in each node, collecting information from all nodes to build a global histogram.
7. The user trains a neural network over several selected variables with given parameters. A plot showing the relative errors in each step is updated and presented to the user via the UI.

8. The user monitors progress and controls job execution; he/she may decide to interrupt the execution (CTRL-C) or let it finish and save all XML flow between the UI and the ISM on the Web server where the Portal is running.
9. The user stops the simulation.

3 Overview of the CrossGrid Architecture

The general CrossGrid architecture, designed to fulfill the requirements presented in the previous section is shown in Fig. 1. The components are organized in six layers according to their role in the Grid. They are described in detail in the following sections. Dependencies between architecture components are presented in Table 1.

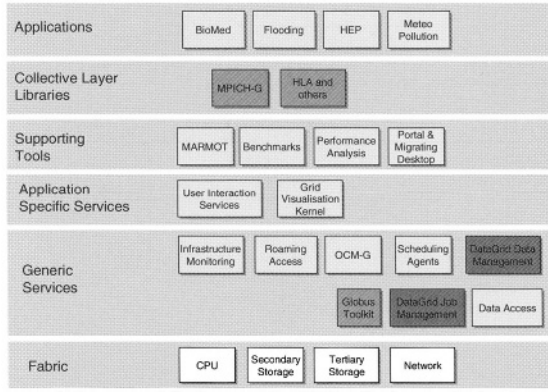


Fig. 1. The CrossGrid architecture

4 Collective Layer Libraries

MPICH-G2 is a Grid-enabled implementation of the MPI v1.1 standard. It is a widely used library within the CrossGrid project. All tools are oriented towards support of MPI applications. Most of the applications will use MPICH-G as a basic collective layer library in the Grid environment. The Grid scheduling system (WP 3.2) is aimed to support submission of parallel interactive jobs on Grid resources.

The High Level Architecture (HLA) was developed as an IEEE standard to facilitate interoperability among simulations and promote reuse of simulation components. HLA Runtime Infrastructure (RTI) federations are distributed systems that consist of multiple processes (federates) communicating across computer nodes. In order to send messages, the applications that are plugged into the RTI have to publish well-defined objects in a tuple space. The applications that want to receive messages have to subscribe to those objects; RTI then notifies subscribed applications that the object has been updated.

Table 1. Dependencies between CrossGrid components; x - direct, i- indirect dependency

	MARMOT	Benchmarks	Perf. Anal.	Perf. Pred.	UIS	GVK	Portal +RA	Sched.	OCMG	Santa-G	Jiro	Postproc.	Data access	Globus GT 2	EDG	Condor-G	Globus GT 3	MPICH-G
Biomed	x	x	x	x	x	x	X	x					I				x	
Flood	x	x	x	x			x	x			x		i					x
HEP	x	x	x	x			x	x					i					x
Pollution	x			x			x			x			i					x
MARMOT								x										x
Benchmark							x		x	x	x				x		x	x
Perf. Anal.									x									x
Perf. Pred.		x								x	x			x				x
UIS																	x	
GVK (1.1)														x				
Portal + RA								x						x	x			
Sched.							x					x			x	x		x
OCMG														x				x
Santa-G																		
Jiro														x	x			
Postproc.														x				
Data access							x							x	x			

5 Tools

MPI debugging and verification tool. MARMOT is a library that is linked to the MPI application in addition to the existing MPI library and that allows a detailed analysis of this application during runtime. It generates a human-readable log file which points out any violations of the MPI standard and unusual behavior or possible problems. MPI calls are traced on each node throughout the whole application. When detecting a deadlock the last few calls (as configured by the user) can be traced back on each node. Application kernels are used for development and testing of the tool, and eventually the tool will be used to facilitate the application development process.

Benchmarks. GridBench is a software system that provides a suite of benchmarks (synthetic or application-based) for characterization of Grids. It also provides automatic generation of job descriptions suitable for execution on the CrossGrid testbed, archives the derived metrics for reference and comparison over time and publishes them in MDS for easy access by users or services. GridBench requires the ability to submit MPI jobs through the Resource Broker and it must be accessible through the CrossGrid Portal. GridBench also depends on the monitoring software.

Grid performance measurement tool. The Grid Performance Measurement (G-PM)[4] tool will provide functionality for basic performance measure-

ments of both Grid applications and the Grid environment. The results of these basic measurements can be directly visualized by the visualization component. In addition, they can serve as input to the high level analysis component and the performance prediction component. The performance analysis tool is not directly connected to the application process, but uses the OCM-G [3] monitoring system. OCM-G and the tool communicate using the OMIS interface.

Performance Prediction Tool This tool provides analytical models to predict the performance of MPI communication routines and specific application kernels in a Grid environment. The tool is used by application developers and users who are interested in interactively analyzing the performance of selected kernels under different conditions. The results can be used to modify the parameters of the parallel execution of an application, such as the number of nodes, the size of the problem, the distribution of data, etc. The infrastructure monitoring component provides parameters that characterize the Grid.

6 Grid Services

User Interaction Service. The User Interaction Service (UIS) comprises a set of Grid Services which allow the setup and interactive steering of complex Grid applications consisting of modules for simulation and visualization. The services focus on setting up and controlling HLA Runtime Infrastructure (RTI) processes that coordinate distributed application components. The User Interaction Service takes advantage of service discovery mechanisms that are present in OGSA and will ultimately be integrated into the OGSA model.

Grid visualization kernel. The Grid Visualization Kernel (GVK) will interconnect distributed simulation and visualization clients, provide uniform interfaces to various visualization devices and therefore allow interactive, near-real-time visualization for Grid applications. GVK connects application elements (visualization and simulation) providing interaction between them in near-real-time. For communication it uses the Globus-IO library.

Portals and Roaming Access. The Migrating Desktop and Portal are separate front-ends which provide a set of components interfaced with the underlying Grid services allowing users to use Grid resources. They will also offer users the possibility to access the same working environment from any workstation by means of the Roaming Access Server (RAS) with additional services: the LDAP manager service responsible for storing user profiles and session manager services responsible for managing application-user sessions. RAS is responsible for communicating with other CrossGrid modules used by the portal. The main external services of RAS include: Job Submission service, Copy files services, Application plugin - the application developer's task is to write an application-specific plugin that can be placed in a container in the portal. RAS uses the DataGrid Replica Manager and Replica Catalogue for managing and accessing files.

Grid Resource Management. The task of scheduling user jobs and submitting them to Grid resources in CrossGrid is done by Scheduling Agents.

By current design, their role is to accept requests in the form of Job Description Language (JDL) files and, based on their optimization strategies which use information from monitoring systems, to decide where to submit the job. For actual job submission and management, DataGrid and Condor-G software is used. Scheduling agents extend the EDG Resource Broker (RB) by providing additional functionality supporting parallel jobs (related to support for interactive applications and the person in a loop concept native to CrossGrid architecture).

Monitoring. Monitoring services in CrossGrid can be divided into two groups: application monitoring and infrastructure monitoring. The online monitoring [3] of running applications provides data for the performance analysis tool. There are two different infrastructure monitoring services: the Jiro-based invasive monitoring system, which gathers data about infrastructure (computing nodes, routers) and non-invasive monitoring (SANTA-G) for monitoring network traffic. Monitoring services will be used by various tools. Infrastructure monitoring will be used by the Benchmarks and Performance Prediction Tool while Application Monitoring will be used by the Performance Monitoring Tool. Information will be published in external Grid Information Services, such as MDS or R-GMA.

Data Access. Data Access extends the basic functionalities of DataGrid software. It will extend the existing systems by optimization of access to tape-resident data. A component expert system provides the framework for deploying various access time estimators and is going to be integrated with the DataGrid Replica Management and optimization algorithms. It is placed on a relatively low level, and therefore only accessible by Crossgrid applications through higher level services, such as GridFTP or the DataGrid Replica Manager.

7 Migration to OGSA vs. Migration to GT3

We can define two different aspects of migration. The first one - migration to GT3 - means using GT3 instead of GT2 as a set of basic services; the second - migration to OGSA [7] - means making the CrossGrid services OGSI-compliant.

Migration to GT3. Migration to GT3 - namely replacing GT2 with GT3 as the basic infrastructure in our testbed - is more a technical issue than an architectural one. As GT3 offers basically the same functionalities as GT2 and no special new features that may be crucial for interactive applications, the demand for switching to GT3 is not strong. However, as some of the new services developed in Crossgrid are planned to be ultimately compatible with OGSA, they will require the availability of GT3, containing an implementation of OGSI. This means we need to have a stable testbed operating with GT2 as well as an experimental testbed running GT3. Such experimental migration requires a lot of effort related to learning and testing the new technology, which is still under development and will certainly be less stable than older versions of Globus.

Migration to OGSA. Migration to OGSA is not strictly a technical question, but rather an architectural and conceptual one. OGSA sees the Grid as a set of dynamically changing services, accessible by standard protocols. Migrat-

ing to OGSA means designing the offered functionalities in the form of services and making these services compatible with the Grid Service Specification [10]. In the case of CrossGrid, the new functionality offered by new middleware has been designed in the form of services from the very beginning. This means that from the architectural point of view, CrossGrid design is compatible with the OGSA model. There are still technical problems to be solved i.e. for C/C++ based services, as there is no C language support in OGSA.

8 Concluding Remarks

Taking into account the current evaluation of OGSA and GT3, the decision for the second project year is that GT2 should be kept as the basic testbed infrastructure, to maintain compatibility with DataGrid, while the experimental installation of GT3 is being tested by the Biomedical application. The new services and tools are developed in such a way that future migration to GT3 and OGSA will be possible. Further decisions are planned following another evaluation at the end of the 2nd project year.

Acknowledgements. We wish to thank M. Turała, M. Garbacz, P.M.A. Sloot D. van Albada, L. Hluchy, W. Funika, R. Wismüller, J. Kitowski, and J. Marco for discussions and suggestions, and also P. Nowakowski for his comments. This research is partly funded by the European Commission IST-2001-32243 Project CrossGrid.

References

1. Bubak, M., M. Malawski, K. Zajac: Towards the CrossGrid Architecture. In: D. Kranzlmueller et al.: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 9th European PVM/MPI Users' Group Meeting, Linz, Austria, 2002, LNCS 2474, pp. 16-24.
2. CrossGrid - Development of Grid Environment for Interactive Applications. Annex 1 - description of Work. http://www.eu-crossgrid.org
3. Balis, B., Bubak, M., Funika, W., Szepieniec, T., and Wismuüller, R.: An Infrastructure for Grid Application Monitoring In: D. Kranzlmueller et al.: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 9th European PVM/MPI Users' Group Meeting, Linz, Austria, 2002, LNCS 2474.
4. Bubak, M., Funika, W., and Wismüller, R.: The CrossGrid Performance Analysis Tool for Interactive Grid Applications. In: D. Kranzlmueller et al.: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 9th European PVM/MPI Users' Group Meeting, Linz, Austria, 2002, LNCS 2474.
5. CrossGrid Project: Deliverables of first year of the project: http://www.eu-crossgrid.org
6. DataGrid Project: http://www.eu-datagrid.org
7. Foster, I., Kesselman, C., Nick, J.M., and Tuecke, S.: The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration, January 2002, http://www.globus.org

8. Global Grid Forum: http://www.ggf.org
9. CrossGrid Software Repository http://gridportal.fzk.de
10. Open Grid Services Infrastructure v1.0 (Draft 29, April 5, 2003) http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf
11. HLA specification, http://www.sisostds.org/stdsdev/hla/

Pegasus and the Pulsar Search: From Metadata to Execution on the Grid

Ewa Deelman[3], James Blythe[3], Yolanda Gil[3], Carl Kesselman[3], Scott Koranda[4], Albert Lazzarini[2], Gaurang Mehta[3], Maria Alessandra Papa[1], and Karan Vahi[3]

[1] Albert Einstein Institute, Golm, Germany
papa@aei.mpg.de
[2] Caltech, Pasadena, CA 91125
lazz@ligo.caltech.edu
[3] USC Information Sciences Institute, Marina Del Rey, CA 90292
{deelman.blythe.gil,carl.gmehta,vahi@isi.edu}
[4] University of Wisconsin Milwaukee, Milwaukee, WI 53211
skoranda@gravity.phys.uwm.edu

Abstract. This paper describes the Pegasus workflow mapping and planning system that can map complex workflows onto the Grid. In particular, Pegasus can be configured to generate an executable workflow based on application-specific attributes. In that configuration, Pegasus uses and AI-based planner to perform the mapping from high-level metadata descriptions to a workflow that can be executed on the Grid. This configuration of Pegasus was used in the context of the Laser Interferometer Gravitational Wave Observatory (LIGO) pulsar search. We conducted a successful demonstration of the system at SC 2002 during which time we ran approximately 200 pulsar searches.

1 Introduction

Grid computing has made great progress in the last few years. The basic mechanisms for accessing remote resources have been developed as part of the Globus Toolkit and are now widely deployed and used. Among such mechanisms are:

- Information services that allow for the discovery and monitoring of resources. The information provided can be used to find the available resources and select the resources that are the most appropriate for a given task.
- Security services that allow users and resources to mutually authenticate and allows the resources to authorize users based on local and global policies.
- Resource management that allow for the remote scheduling of jobs on particular resources.
- Data management services that enable users and applications to manage large, distributed and replicated data sets. Some of the available services deal with locating particular data sets, others with efficiently moving large amounts of data across wide area networks.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 821–830, 2004.
© Springer-Verlag Berlin Heidelberg 2004

With the use of the above mechanisms, one can manually find the available resources and schedule the desired computations and data movements. However, this process is time consuming and can potentially be complex. As a result it is becoming increasingly necessary to develop higher level services that can automate the process and provide an adequate level of performance and reliability.

The NSF-funded Grid Physics Network (GriPhyN [1]) project aims to develop just such services. In this paper we focus in particular on a workflow management system that can map complex workflows onto the Grid. In general, GriPhyN aims to support large-scale data management in physics experiments such as high-energy physics, astronomy and gravitational wave physics. GriPhyN puts data both raw and derived under the umbrella of Virtual Data. A user or application can ask for data using application-specific metadata without needing to know whether the data is available on some storage system or if it needs to be computed. To satisfy the request, GriPhyN will schedule the necessary data movements and computations to produce the requested results.

The paper is organized as follows: first, we describe Pegasus, the system we developed to map both domain-independent and domain-specific requests onto the Grid. Section 3 describes in more detail the heart of domain-specific Pegasus that includes an A1-based planner. Following is a description of applying Pegasus to the gravitational wave pulsar search. Section 5 gives a summary of our results and experiences demonstrating our system during the SC 2002 conference, and discusses the related work. We conclude with final remarks in Section 6.

2 Pegasus, a Workflow Mapping System

In general we can think of applications as being composed of application components. The process of application development (shown in Figure 1) can be described as follows. First, the application components are selected, their input and output file names are identified by their logical names (names that uniquely identify the content of the file, but not its location), and the order of the execution of the components is specified. As a result, we obtain an abstract workflow (AW), where the behavior of the application is specified at an abstract level (without identifying the resources needed for execution.) Next this workflow needs to be mapped onto the available Grid resources, performing resource discovery and selection. Finally the resulting concrete workflow (CW) is sent to the executor, such as Condor-G/DAGMan [21] for execution. Pegasus [10,9,18, 16], which stands for Planning for Execution in Grids, was developed at ISI as part of the GriPhyN project. Pegasus is a configurable system that can map and execute complex workflows on the Grid. Pegasus exists in two configurations: Abstract workflow-driven and metadata-driven.

2.1 Abstract Workflow-Driven Pegasus

In the first configuration, Pegasus is an application-independent software that takes an abstract workflow as input and produces a concrete workflow that

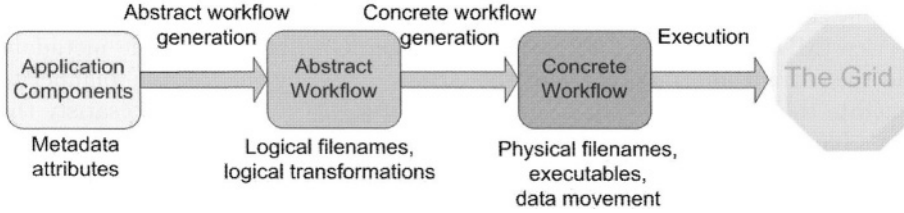


Fig. 1. Generating and Mapping Workflows onto the Grid.

can be executed on the Grid. This version of Pegasus is available as part of the GriPhyN Virtual Data Toolkit [3]. In that configuration, Pegasus receives an abstract workflow (AW) description from Chimera[20], produces a concrete workflow (CW), and submits it to DAGMan for execution. The workflows are represented as Directed Acyclic Graphs (DAGs). AW describes the transformations and data in terms of their logical names. CW, which specifies the location of the data and the execution platforms, is optimized by Pegasus from the point of view of Virtual Data. If data products described within AW are found to be already materialized (via queries to the Globus Replica Location Service (RLS)) [13], Pegasus reuses them and thus reduces the complexity of CW. Pegasus also consults the Transformation Catalog (TC) [19] to determine the locations where the computations can be executed. If there is more than one possible location, a location is chosen at random. Pegasus also adds data transfer and data registration nodes. Transfer nodes are used to stage data in or out. Registration nodes are used to publish the resulting data products in the RLS. They are added if the user requested that all the data be published and sent to a particular storage location. Once the resources are identified for each task, Pegasus generates the submit files for DAGMan. In that configuration, Pegasus has been shown to be successful in mapping workflows for very complex applications such as the Sloan Digital Sky Survey [6] and the Compact Muon Source [18].

2.2 Metadata-Driven Pegasus

Pegasus can also be configured to perform the generation of the abstract workflow based on application-level metadata attributes (See Figure 2.) Given attributes such as time interval, frequency of interest, location in the sky, etc., Pegasus is able to produce any virtual data products present in the LIGO Pulsar search, described in the Section 4.

Pegasus uses the Metadata Catalog Service (MCS) [26] to perform the mapping between application-specific attributes and logical file names of existing data products. A1-based planning technologies, described in the next section, are used to construct both the abstract and concrete workflows. MCS is also used to determine the metadata and logical file names for all other sub products that can be used to generate the data product. Pegasus then queries the RLS to find

the physical locations of the logical files. The Globus Monitoring and Discovery Service (MDS) [14] is used to find the available Grid resources. The metadata and the current information about the Grid are used by the Pegasus planner to generate the concrete workflow (in the form of a DAG) necessary to satisfy the user's request. The planner reuses existing data products where applicable. The generated plan specifies the sites where the job should be executed and refers to the data products in terms of metadata. This metadata-defined plan needs to mapped to particular file instances. Pegasus determines the logical names for the input data in the plan by querying the MCS and then the physical names by querying the RLS. In addition it queries the Transformation Catalog to get the complete paths for the transformations at the execution locations described in the plan. Finally, the submit files for Condor-G/DAGMan are generated.

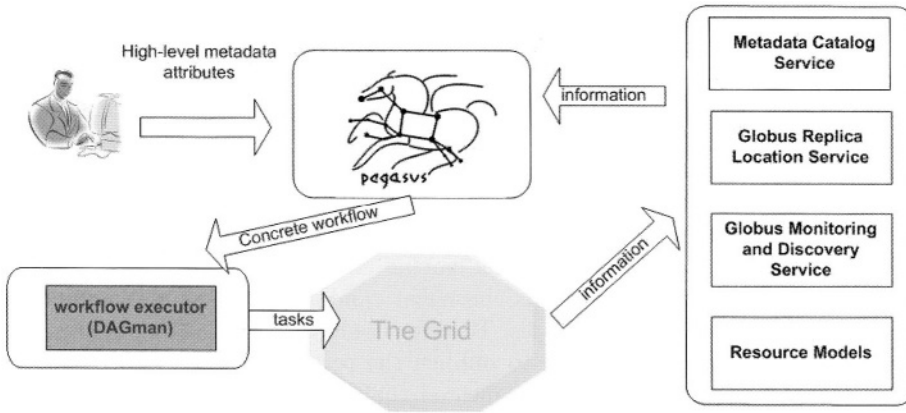


Fig. 2. Configuration of Pegasus when used to construct both the abstract and concrete workflows.

Pegasus contains a Virtual Data Language generator that can populate the Chimera catalog with newly constructed derivations. This information can be used for provenance tracking of the derived data products. Finally, Pegasus also sends the concrete workflow to DAGMan for execution. As the result of execution of the workflow, the newly derived data products are registered both in the MCS and RLS and thus are made available to the following requests. We have configured Pegasus to support the LIGO and GEO pulsar searches. Details about the search can be found in Section 4 and [17].

3 Planning Approach

The Abstract and Concrete Workflow are generated using the Prodigy planner [27]. The planner models the application components along with data transfer and data registration as operators. Each operator's parameters include the

host where the component is to be run and as a result an output plan corresponds to a concrete workflow. In addition, some of the effects and preconditions of the operators capture the data produced by components and their input data dependencies. The state information used by the planner includes a description of the available resources and the relevant files that have already been created. The input goal description can include (1) a metadata specification of the information the user requires and the desired location for the final output file, (2) specific components to be run or (3) intermediate data products. Several issues make this application domain challenging, we touch upon them as we describe the domain model in more detail.

3.1 State Information

The planner's world state includes information about resources. Some state information changes slowly, such as the operating system or total disk space available on a resource, and some of the information can change in seconds or minutes, such as the available memory or job queue length. In the long run the planner may need to reason about how the information can change over time, but in our initial implementation we only model the type of a host, network bandwidths and file information.

It is useful for the planning state to include metadata about the files for several reasons. First, the planner can assume the task of creating both the abstract and concrete workflows. Second, it is also more appropriate to reason at the level of the metadata rather than at the level of the files that represent that data content. Rather than search for a file with appropriate characteristics, the components are linked by the characteristics themselves. This also avoids quantifying over the set of existing files that may change during planning as objects are created and destroyed.

3.2 Goal Statements

In most planning applications, goals refer to properties that should be true after the plan has been executed. For the planner, such goals include having data described by the desired metadata information on some host. However, it is also sometimes useful to specify goals that refer to intermediate components or data products, or for registering certain files. Thus the goal statement can specify a partial plan.

3.3 Operator Descriptions

The operators themselves represent the concrete application of a component at a particular location to generate a particular file or a file movement across the network. Their preconditions represent both the data dependencies of the component, in terms of the input information required, and the feasible resources for running the component, including the type of resource. These operators capture

information similar to that represented in Chimera's Virtual Data Language [20], such as the name of the component and its parameters. However, the operators also contain the additional information about the preconditions necessary for the use of the component, and provide the effect of the application of the component on the state of the system, such as the consumption of the resources. Further information about resource requirements, such as minimal physical memory or hard disk space, is a planned extension.

Plans generated in response to user requests may often involve hundreds or thousands of files and it is important to manage the process of searching for plans efficiently. If a component needs to be run many times on different input files, it is not useful for the planner to explicitly consider different orderings of those files. Instead the planner reasons about groups of files that will be treated identically. An auxiliary routine allocates the files to different groups, looking for a locally optimal allocation. Since the number of input files or groups may vary by component and even by invocation, the preconditions are modeled using quantification over possible files.

3.4 Solution Space and Plan Generation Strategy

In our initial approach, we seek high-quality plans with a combination of local search heuristics, aimed at preferring good choices for individual component assignments, and an exhaustive search for a plan that minimizes the global estimated runtime. Both aspects are necessary: without the global measure, several locally optimal choices can combine to make a poor overall plan because of conflicts between them. Without the local heuristics, the planner may have to generate many alternatives before finding a high quality plan.

4 LIGO and GEO Pulsar Search

LIGO (Laser Interferometer Gravitational-Wave Observatory,) [2,4,7] is a distributed network of three km-scale interferometers occupying two sites in the U.S. The construction project was funded by NSF and jointly built by Caltech and MIT. GEO 600 is a 600 meter interferometer installed in Hannover, Germany built by a British-German collaboration. The observatories' mission is to detect and measure gravitational waves predicted by general relativity, Einstein's theory of gravity, in which gravity is described as due to the curvature of the fabric of time and space. One well-studied source of gravitational waves is the motion of dense, massive astrophysical objects such as neutron stars or black holes. Other signals may come from supernova explosions, quakes in neutron stars, and pulsars.

Gravitational waves interact extremely weakly with matter, and the measurable effects produced in terrestrial instruments by their passage will be miniscule. In order to establish a confident detection or measurement, a large amount of auxiliary data will be acquired and analyzed along with the strain signal that measures the passage of gravitational waves. The amount of data that will be

acquired and cataloged each year is in the order of tens to hundreds of terabytes. Analysis on the data is performed in both time and Fourier domains.

Searching for pulsars that may emit gravitational waves requires, involves among others, a Fourier analysis of a particular set of frequencies over some time frame. To conduct a pulsar search, for example, the user must find a number of files of raw data output corresponding to this time frame, extract the required channel, concatenate the files and make a series of Fourier transforms (FT) on the result. The desired frequencies must then be extracted from the set of FT output files, and processed by a separate program that performs the pulsar search.

Depending on search parameters and the details of the search being conducted, a typical LIGO or GEO pulsar search may require thousands of Fourier transforms, some of that may have already been performed and stored at some location in the Grid. The results must be marshaled to one host for frequency extraction, and the final search must be executed on a different host because of the program requirements. In all, many gigabytes of data files may be generated, so a fast-running solution must take the bandwidth between hosts into account.

We have tailored the metadata-driven Pegasus to support LIGO and GEO pulsar searches. This involved developing application-specific operators for the planner and providing a Globus interface to the LIGO data analysis facilities that are customized to the LIGO project needs. This included developing a new Globus jobmanager [15] to enable scheduling of jobs on the LIGO analysis system and providing a GridFTP [5] interface to stage data in and out of the system.

5 Results and Related Work

The Metadata approach described in this paper was first demonstrated at the SC 2002 conference held in November at Baltimore. The Pegasus system was configured to generate both the abstract and the concrete work flows and run the LIGO and GEO Pulsar searches. For this demonstration the following resources were used: 1) *Caltech (Pasadena, CA):* LIGO Data Analysis System (LDAS) and Data Storage. 2) *ISI (Marina del Rey, CA):* Condor Compute Pools, Data Storage, Replica Location Services, and Metadata Catalog Services. 3) *University of Wisconsin (Milwaukee):* Condor Compute Pools and Data Storage.

The requests for pulsar searches were generated using an auto generator that produced requests both for known pulsars (approximately 1300 known pulsars) as well as random point searches in the sky. A user could also request a specific pulsar search by specifying the metadata of the required data product through a web-based system. Both the submission interfaces as well as all the compute and data management resources were Globus GSI (Grid Security Infrastructure) enabled. Department of Energy issued X509 certificates were used to authenticate to all the resources.

During the demonstration period and during a subsequent run of the system approximately 200 pulsar searches were conducted (both known as well as random) generating approximately 1000 data products involving in the order of 1500 data transfers. The data used for this demonstration was obtained from

the first scientific run of the LIGO instrument. The total compute time taken to do these searches was approximately 100 CPU hrs. All the generated results were transferred to the user and registered in the RLS. The metadata for the products was registered in the MCS as well as into LIGO's own metadata catalog. Pegasus also generated the corresponding provenance information using the Virtual Data Language and used it to populate in the Chimera Virtual Data Catalog.

The execution of the jobs was monitored by two means. For each executable workflow, a start and end job were added. They logged the start time and the end time for the workflow into a MySQL database. This information was then published via an http interface. We also implemented a shell script that parsed the condor log files at the submit host to determine the state of the execution and published this information to the web interface.

5.1 Related Work

Central to scheduling large complex workflows is the issue of data placement, especially when the data sets involved are very large. In Pegasus we give preference to the compute resources where the input data set is already present. Others [23, 24] look at the data in the Grid as a tiered system and use dynamic replication strategies to improve data access. In [25] significant performance improvement are achieved when scheduling is performed according to data availability while also using a dynamic replication strategy.

While running a workflow on the Grid makes it possible to perform large computations that would not be possible on a single system, it leads to a certain loss of control over the execution of the jobs as they might be executed in different administrative domains. To counter this, there are other systems [11,22] that try to provide QoS guarantees required by the user while submitting the workflow to the Grid. NimrodG uses the information from the MDS to determine the resource that meets the budget constraints specified by the user, while [22] monitors a job progress over time to ensure that guarantees are being met. If a guarantee is not being met schedules are recalculated.

Other work has focused on developing application specific schedulers that maximize the performance of the individual application. In AppLeS [8], scheduling is done on the basis of a performance metric that varies from application to application. This leads to a customized scheduler for each application and not a general solution. Some schedulers have focused on parameter sweep applications, where a single application is run multiple times with different parameters [12]. Since there are no interdependencies between jobs, the scheduling process is far simpler from the one addressed here.

Each of the systems mentioned above are rigid because they use a fix set of optimization criteria. In this work we are developing a framework for a flexible system that can map from the abstract workflow description to its concrete form and can dynamically change the optimization criteria.

6 Conclusions and Future Work

The work presented in this paper describes the Pegasus planning framework and its application to the LIGO and GEO gravitational wave physics experiments. The interface to the system was at the level of the application and AI planning techniques were used to map user requests to complex workflows targeted for execution on the Grid. As part of our future work, we plan to investigate the planning space further, explore issues of planning for only parts of the workflow at a time, using dynamic system information to make more reactive plans, etc.

In our demonstration we used scientifically meaningful data and used both generic Grid resources as well as LIGO specific resources enabled to work within the Grid. The results of our analysis were fed back to the LIGO metadata catalogs for access by the LIGO scientists.

Although we were able to model the pulsar search within the planner, the issue of expanding this approach to other applications needs to be evaluated.

Acknowledgments. We would like to thank the following LIGO scientists for their contribution to the development of the grid-enabled LIGO software: Stuart Anderson, Marsha Barnes, Kent Blackburn, Philip Charlton, Phil Ehrens, Ed Maros, Greg Mendell, Mary Lei, and Isaac Salzman. This research was supported in part by the National Science Foundation under grants ITR-0086044(GriPhyN) and EAR-0122464 (SCEC/ITR). LIGO Laboratory operates under NSF cooperative agreement PHY-0107417.

References

1. *GriPhyN.* http://www.griphyn.org.
2. *LIGO.* http://www.ligo.caltech.edu.
3. *The Virtual Data Toolkit,* http://www.lsc-group.phys.uwm.edu/vdt/home.html.
4. A. Abramovici et al. *LIGO: The Laser Interferometer Gravitational-Wave Observatory (in Large Scale Measurements).* Science, 1992. 256(5055): p. 325-333., 1992.
5. W. Allcock et al. *Data management and transfer in high performance computational grid environments,.* Parallel Computing Journal, 28, 5, 2002a, 749-771., 2002.
6. J. Annis et al. *Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey.* in Supercomputing. 2002. Baltimore, MD., 2002.
7. B. C. Barish and R. Weiss. *LIGO and the Detection of Gravitational Waves. Physics Today, 1999. 52(10): p. 44.* 1999.
8. F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. *Application-Level Scheduling on Distributed Heterogeneous Networks.* In Proceedings of Supercomputing'96 (Pittsburgh, 1996)., 1996.
9. J. Blythe, E. Deelman, Y. Gil, and C. Kesselman. Transparent grid computing: a knowledge-based approach. In *Innovative Applications of Artificial Intelligence Conference,* 2003.

10. J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, and K. Vahi. The role of planning in grid computing. In *International Conference on Automated Planning and Scheduling,* 2003.
11. R. Buyya, D. Abramson, and J. Giddy. *An Economy Driven Resource Management Architecture for Global Computational Power Grids.* in The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000). 2000. Las Vegas, USA,., 2000.
12. H. Casanova et al. *Heuristics for Scheduling Parameter Sweep Applications in Grid environments,* in 9th Heterogeneous Computing Workshop (HCW'2000). 2000. Cancun, Mexico., 2000.
13. A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, and Giggle. *A framework for constructing scalable replica location services,.* Proceedings of Supercomputing 2002 (SC2002), November 2002., 2002.
14. K. Czajkowski, *S.* Fitzgerald, I. Foster, and C. Kesselman. *Grid information services for distributed resource sharing.* In Proceedings of the 10th IEEE 5'gmposium on High-Performance Distributed Computing, August 2001., 2001.
15. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. *A Resource Management Architecture for Metasystems.* Lecture Notes on Computer Science, 1998., 1998.
16. E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. *Grid Resource Management,* chapter Workflow Management in GriPhyN. Kluwer, 2003.
17. E. Deelman et al. *GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists.* in 11th Intl Symposium on High Performance Distributed Computing. 2002., 2002.
18. E. Deelman et al. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing, 2003.,* 1(1), 2003.
19. E. Deelman, C. Kesselman, and G. Mehta. *Transformation Catalog Design for GriPhyN.* GriPhyN technical report 2001-17, 2001.
20. I. Foster et al. *Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation.* in Scientific and Statistical Database Management. 2002., 2002.
21. J. Frey et al. *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* in 10th International Symposium on High Performance Distributed Computing. 2001: IEEE Press., 2001.
22. P. Keyani, N. Sample, and G. Wiederhold. *Scheduling Under Uncertainty: Planning for the Ubiquitous Grid.* Stanford Database Group.
23. K. Ranganathan and I. Foster. *Design and Evaluation of Dynamic Replication Strategies for a High Performance Data Grid.* in International Conference on Computing in High Energy and Nuclear Physics. 2001., 2001.
24. K. Ranganathan and I. Foster. *Identifying Dynamic Replication Strategies for a High-Performance Data Grid.* In Proceedings of the Second International Workshop on Grid Computing (2001)., 2001.
25. K. Ranganathan and I. Foster. *Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications.* in International Symposium for High Performance Distributed Computing (HPDC-11). 2002. Edinburgh., 2002.
26. G. Singh et al. *A Metadata Catalog Service for Data Intensive Applications.* SC03, 2003.
27. M. Veloso, J. Carbonell, et al. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical AI,* 7:81–120, 1995.

Flood Forecasting in a Grid Computing Environment*

Ladislav Hluchý, Ján Astaloš, Miroslav Dobrucký, Ondrej Habala, Branislav Šimo, and Viet D. Tran

Institute of Informatics, SAS, Dubravska cesta 9, Bratislava 8407, Slovakia
hluchy.ui@savba.sk

Abstract. This paper presents a prototype of a flood forecasting system based on Grid computing. Flood forecasting is a complex problem that requires cooperation and coupling of many simulation models. To enable this coupling in an effective way, a part of the CrossGrid project is aimed towards developing a Grid Computing Environment (GCE) for flood forecasting, as described here. The Grid computing environment consists of a cascade of simulation models, a storage system for computed and measured data and other used datasets, a web-based portal with collaboration tools and a powerful computation facility. The whole system is tied together by Grid technology and is used to support a virtual organization of experts, developers and users.

1 Introduction

Over the past few years, floods have caused widespread damages throughout the entire Europe. They have affected most of the European population and they resulted in heavy material losses. The need for better flood protection has become imminent.

In this paper we present the Grid Computing Environment for Flood Forecasting, a system intended as a support tool for hydrometeorological experts. Grid computing environments (GCEs) have increasingly gained attention in the past few years. Advances in technological infrastructure as well as a better awareness of the needs of application scientists and engineers have been the primary motivating factors. In particular, the shift in emphasis from low-level application scheduling and execution [2] to high-level problem solving indicates that Grid comput is becoming increasingly important as a way of doing science. A GCE is a Problem Solving Environment (PSE) [1] with specifically formed computation kernel, using the power of Grid Computing. Good examples of some GCEs can be found in [8].

The system described herein is composed of a cascade of three simulation models - meteorological, hydrological and hydraulic ones. The whole cascade is able to predict water flow in a flooded area, but users may also reduce their

* This work is supported by EU 5FP CROSSGRID IST-2001-32243 RTD project and the Slovak Scientific Grant Agency within Research Project No. 2/3132/23.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 831–839, 2004.
© Springer-Verlag Berlin Heidelberg 2004

questions to simple weather prediction or to river level development in a certain area (a hydrograph).

The interface of this GCE is a WWW-based portal, enabling users to run simulations and evaluate results from anywhere in the world, using a simple computer with web browser and Internet connection. The front-end is a collection of web pages with options to run desired simulations. Behind this, a sophisticated collection of data, model codes, scripts and Grid middleware is hidden. The GCE uses public-key based authentication mechanisms, enabling secure and private data transfer, processing and storage. Furthermore, the system encompasses some collaboration tools, enabling users to exchange files and to communicate with each other. The basic control and data flow through the system is shown in Fig. 1. It illustrates well the typical PSE features of the whole system - complex computation kernel is hidden behind preprocessing/postprocessing systems and easy-to-use, intuitive web forms.

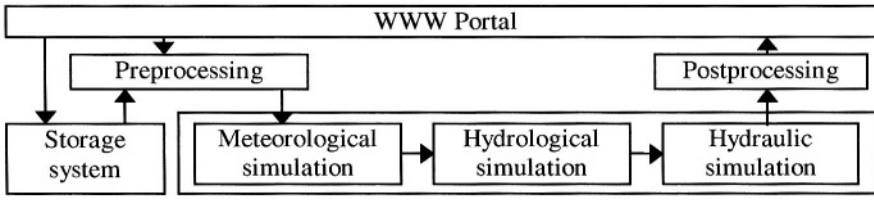


Fig. 1. General schema of the GCE for Flood Forecasting

The basic data flow is going from the storage system, through the cascade of simulations, postprocessing and visualization, and then the results are displayed to the user. The whole system is controlled by the central portal interface through configuration files and/or commands to existing processes. Of course, the GCE is much more complicated and versatile than shown in Fig. 1. The simulations can be run standalone, without the need for the complete cascade to compute, partial results are stored into the storage system and users can control various parameters of any of its components.

This software is developed as a part of the CrossGrid project [9]. The final product will enable much more than the first software release described in the next sections. It will include complex data management, probably a collection of concurrent models for each stage of the simulation cascade, much more sophisticated and comfortable user interface with options for prediction and warning automation, while retaining scalability and ease of use.

2 Simulation Cascade

The GCE is based on the cascade of three types of simulations: meteorological, hydrological and hydraulic.

Each of the three simulations can be considered as a block that can be further decomposed. Each simulation block consists of several sub-blocks. Each simulation takes the input data set from the storage, preprocesses the data and starts the computational part. After its finish the post-processing of the output dataset is performed.

Pre- and post-processing filter blocks (dashed boxes in the figure) extract the data of interest from the dataset and format it to a structure suitable for the following processing or storage. The filter blocks may or may not appear in actual simulation block depending on specific data structure requirements of individual components. Output filtering before storing the resultant datasets may be important in case of large datasets (like meteorological output) where only a part there of is interesting.

2.1 Meteorological Modeling

Forecasting of flood events requires quantitative precipitation forecasts as well as forecasting of temperature (to determine snow accumulation/melting). The simulation of atmospheric processes with the objective to predict future developments is the objective of Numerical Weather Prediction (NWP). The meteorological models are generally computationally intensive and they are usually running on the supercomputer class systems. The output of mesoscale models is used by meteorologists as a boundary condition for regional models. This nesting requires efficient management and transfers of large (tens to hundreds of megabytes) datasets. The prediction of flash floods will require employment of high resolution storm-scale models.

In the first software release of our system the ALADIN/SLOVAKIA [14] model will be used, currently operated by Slovak Hydrometeorological Institute (SHMI). ALADIN is a LAM (Limited Area Model) developed jointly by Meteo France and cooperating countries. It can be viewed as an extension to ARPEGE, a global NWP model operated in Meteo France. The main purpose is to provide more detailed short range forecasts inside limited domain of interest. Currently the ALADIN model is operated in 13 Euro-Mediterranean countries.

The ALADIN model has high demands on communication network and thus is not very suitable for distributed running in the Grid. The main benefit of using Grid environment could be in performing of parameter studies.

Visualization of output data on the portal is currently done by generation of JPEG images using postprocessing application CHAGAL and combination of NCAR and ImageMagick converters. Input data for hydrological models (quantitative precipitation forecasts for irregular area) is extracted by an application developed by SHMI.

In the first prototype the processing of models on Grid resources is done by submission to CrossGrid testbed by means of DataGrid job submission service. Input and output data are stored on Storage Element. In the future, we plan to integrate replica management system.

It is common practice in weather forecasting to compare outputs of multiple meteorological models to get more accurate forecasts. Therefore also other me-

teorological models (e.g. MM5 and storm scale models) will be adapted for Grid environment. Horizontal resolution of ALADIN/SLOVAKIA (currently 7 km) needs to be increased, which imposes higher computing power demands. Also, processing of ALADIN/LACE for Central Europe (used as boundary conditions for ALADIN/SLOVAKIA) will be tested in Grid environment.

2.2 Hydrological Modeling

Several hydrological simulation models wiil be used, depending on conditions and needs which model will be applied for which situation and territory; they can also be used in combination. We plan to integrate the following models in our GCE:

- HEC-1 model - is single-event-based, designed to simulate the surface runoff response of a river basin to precipitation by representing the basin as an interconnected system of hydrologic and hydraulic components [12]. The result of the modeling process is the computation of streamflow hydrographs at desired locations in the river basin.
- HSPF - Program for continuous simulation of watershed hydrology and water quality [13]. The HSPF model uses information such as the time history of rainfall, temperature and solar radiation; land surface characteristics such as land use patterns; and land management practices to simulate the processes that occur in a watershed. The result of this simulation is a time history of the quantity and quality of runoff from an urban or agricultural watershed.

We started to use the HSPF model and other models will be added later. These models need not be parallelized as they are not computationally intensive. However, they are data-intensive, and numerous parameters need to be specified by model calibration via parametric run in a Grid environment. Model tuning needs more effort to obtain land characteristic data and developers can profit from utilization of our first GCE version running in our Grid testbed.

Measured historical data are collected in tables (plain text, Excel table) and transformed into WDM file [13] using converter scripts and USGS tools (annie, iowdm, hspf) [15]. Also HSPF itself can manage most of the input data and transform it to/from WDM file.

New forecasted data (precipitation and temperature for 2 days) can be read by HSPF, inserted to WDM file and immediately used for simulation. When measured data are ready in following days, HSPF can overwrite the forecasted ones. HSPF run is steered by UCI file in which starting/ending data and other parameters (tuning parameters) can be modified via portal. Output data are stored back into the WDM file and are also written into small output file that is passed to the next step in the simulation cascade.

2.3 Hydraulic Modeling

Two different hydraulic models are used in the GCE. One of them is FESWMS (Finite Element Surface-Water Modeling System) Flo2DH [7] which is a 2D hydrodynamic, time-implicit, depth- averaged, free-surface, finite element model.

Flo2DH computes water surface elevations and flow velocities for steady states and unsteady sub-critical flow at nodal points in a finite element mesh representing a body of water (such as a river, harbor, or estuary).

The second model is DaveF that is currently developed in cooperation with the author of FESWMS. It is 2D hydrodynamic, time-explicit, cell-centered, Godunov-type finite volume model. DaveF can simulate super-critical flow (flood wave, flash flood) with wetting/drying in large extent.

Both models are supported by SMS graphical user interface that can be used for pre- and post- processing. For better visualization, the simulation results can be also imported into GIS (Geographic Information System) software such as ArcView, GRASS.

As simulation of floods is very computation-expensive, both models are optimized and parallelized by II-SAS. The details of mathematical approaches and parallelization process of hydraulic models have been published in several independent papers [10], [11], so they are not included in this paper.

3 Portal Interface

Web based portal provides user interface to the GCE. Two main parts are a simulation part and a collaboration part. The simulation part represents the more important part of the portal as it provides the interface to the simulations that are the core of the forecasting system. Authenticated users can start simulations, watch their progress, get the files with results and see their visualizations.

The collaboration part is to be implemented yet. It will provide some of the functionality of the software termed as groupware. We intend to incorporate tools already developed by third parties. Both parts are described in more detail in the following sections.

3.1 Collaboration

Appropriate collaboration among users of the flood portal, i.e. experts in respective fields, is important for making the right decisions in the time of the crisis and in the process of the tuning of the system during common days. The problem of effective collaboration is being solved by software tools called groupware. The reason for embedding some of the groupware technologies into the flood portal is to enable the communication (in various ways) among many users of the portal. While they could use common telecommunication infrastructure such as mobile phones, specific groupware technologies provide added value services.

Groupware facilitates the work of groups. This technology is used to communicate, cooperate, coordinate, solve problems, compete or negotiate. It relies on modern computer networks and technologies such as e-mail, mailing lists, newsgroups, chat, videoconferences, shared drawing boards, etc.

There are two dimensions that categorize groupware [17]: time and place. Time: synchronous (communication occurs at the same time) or asynchronous (communication occurs at different times). Place: the same place (people meet

in the same room) or different places (participants are in geographically distributed locations). Asynchronous groupware: e-mail, newsgroups, discussion board, workflow system, hypertext and group calendars. Synchronous groupware: chat, shared whiteboards and video communications.

Web log is a list of time-ordered messages (usually regarding certain subject). Such web logs containing various statements of officials, weather predictions and actual river status proved very useful for public during the summer floods in the Czech Republic in 2002 and were later used by insurance companies.

3.2 Simulations in the Portal Prototype

The portal prototype (Fig. 2) is based on the GridPort toolkit, [16], which provides an easy-to-use interface to such Grid functionalities as authentication, job management, file transfers, and resource information services and which uses the services of underlying Globus toolkit - a Grid middleware.

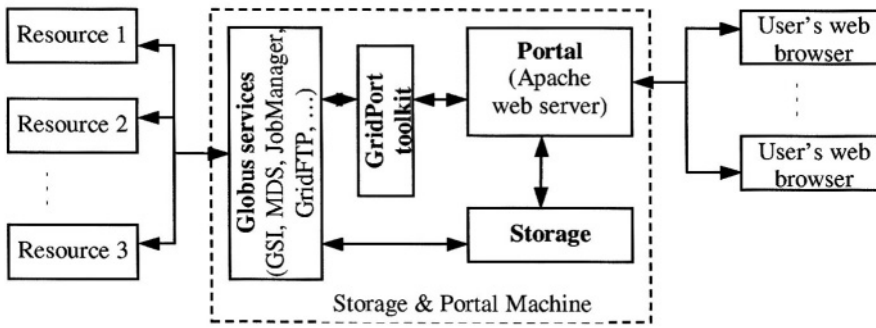


Fig. 2. Architecture of the portal

The portal prototype enables its users to access the resources of the Flood Virtual Organization. User interface is divided into several sections. Some are general; the others provide interface and functionality tailored to specific application. The general ones are: authentication, job management, file management, and Grid file transfers.

Authentication provides means for the users to log in to the Grid using standard X.509 certificates, to log out, and to check current status. The used version of the GridPort toolkit requires user private keys and certificates to be accessible by the web server, thus increasing the risk of user's key becoming compromised in the event of the web server being broken into. Therefore we are working to incorporate the MyProxy service [18] to out portal in order to increase the security.

Using the Job Management feature, one can submit general jobs to the Grid resources, list all his/her jobs and view their status, view standard output/error, or cancel a job. By general job submission we mean the possibility to submit

any executable or a script prepared by the user. It is important in order to give the users the ability to perform computations that are not directly incorporated into the portal.

File management allows users to upload and download their files from the portal to a local computer and vice versa, view and edit files that reside on the portal without the need to download them to local computer and upload them back. Grid file transfers include GASS, GridFTP and third party (using GridFTP) transfers.

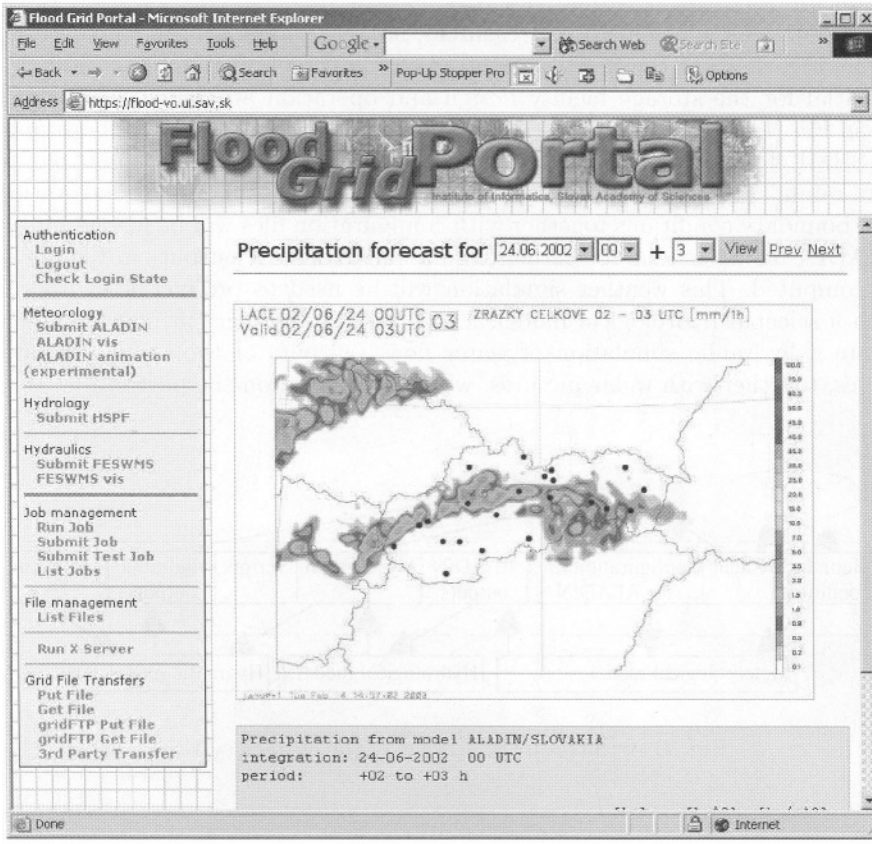


Fig. 3. Portal interface with a precipitation forecast from the ALADIN model

The application specific interface (Fig. 3) includes three types of application - meteorological imulation (ALADIN), hydrological simulation (HSPF) and hydraulic simulation (FESWMS). For each application a user can specify the most important input values via a web form, submit the job and view results. Result can be viewed in the form of a textual or a graphical output. Graphical output

is produced either by application itself (ALADIN), or created using stand-alone visualization tools (HSPF, FESWMS). It allows a user to assess the situation quickly while the textual output provides more details and higher accuracy.

4 Storage Facility

The storage space for simulation outputs and direct measurements used by the application is provided by II SAS. Hourly outputs of meteorological simulation, hydrographs provided by the hydrological part of the cascade and selected hydraulic outputs will be stored. The storage will also hold configuration files for the simulations and some other resources, needed to operate the application.

In Fig. 4, from the entities involved in the simulating process only those essential for the storage facility design and operation are ashown. As we can see in Fig. 4, initial input data (ALADIN boundary conditions) will be supplied by SHMI. The same organization will also provide postprocessed radar pictures with rainfall density and other meteorological data measured in ground stations. The boundary conditions together with configuration files will be loaded into the ALADIN model code and weather simulation (ALADIN outputs in Fig. 4) will be computed. This weather simulation will be used to produce a hydrograph with a selected hydrological model. The produced hydrograph may be used to create a hydraulic simulation of water flow. Outputs of the three simulation stages, together with radar pictures, will be displayed on the portal.

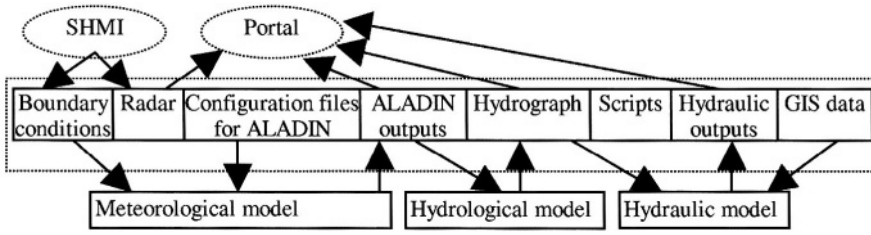


Fig. 4. Architecture and usage of the storage system

The stored files are accessible through standard Grid tools used in the CrossGrid testbed [3]. We will also have to establish a common description scheme for these files and to store the metadata in a Grid-aware database system. We are going to use the Datagrid Spitfire [6] engine, together with MySQL RDBMS [4].

The metadata structure (to be developed) will have to include detailed information about origin of the file, time of its creation, the person who actually created it, etc. In case the file is the output of a simulation, the metadata will also contain names of the input files, model executable and configuration files.

5 Conclusion

We have designed and implemented a Grid Computing Environment for flood forecasting as a part of the CrossGrid project. The main component of the system is a cascade of meteorological, hydrological and hydraulic simulations. The final application is going to be used by a virtual organization for flood prevention.

References

1. Gallopoulos, S., Houstis, E., Rice, J.: Computer as Thinker/Doer: Problem-Solving Environments for Computational Science. IEEE Computational Science and Engineering Magazine, 1994, Vol. 2, pp. 11-23.
2. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., 1999.
3. Marco, R.: Detailed Planning for Testbed Setup. The CrossGrid Project, 2002. http://grid.ifca.unican.es/crossgrid/wp4/deliverables/CG-4-D4.1-001-PLAN.pdf
4. Widenius, M., Axmark, D.: MySQL Reference Manual. O'Reilly & Associates, Inc., 2002.
5. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl. J. Supercomputer Applications, 11(2):115-128, 1997.
6. Bell, W. H., Diana Bosio, D., Hoschek, W., Kunszt, P., McCancel, G., Silander, M.: Project Spitfire - Towards Grid Web Service Databases. Global Grid Forum 5, Edinburgh, 2002.
7. FESWMS – Finite Element Surface Water Modeling System. http://www.bossintl.com/html/feswms.html
8. Ramakrishnan, N., Watson, L. T., Kafura, D. G., Ribbens, C. J., Shaffer, C. A.: Programming Environments for Multidisciplinary Grid Communities. http://citeseer.nj.nec.com/ramakrishnan01programming.html
9. EU 5FP project CROSSGRID. http://www.crossgrid.org/
10. L. Hluchy, D. Froehlich, V. D. Tran, J. Astalos, M. Dobrucky, G. T. Nguyen: Parallel Numerical Solution for Flood Modeling Systems. Parallel Processing and Applied Mathematics PPAM'2001, pp. 485-492.
11. L. Hluchy, V. D. Tran, J. Astalos, M. Dobrucky, G. T. Nguyen, D. Froehlich: Parallel Flood Modeling Systems. International Conference on Computational Science ICCS'2002, pp. 543- 551.
12. HEC-1 http://www.bossint1.com/html/hec-1.html
13. Hydrological Simulation Program-Fortran. http://water.usgs.gov/software/hspf.html
14. The International ALADIN Project. http://www.cnrm.meteo.fr/aladin/
15. USGS Surface-Water Software http://water.usgs.gov/software/surface_water.html
16. NPACI GridPort Toolkit. https://gridport.npaci.edu/
17. Pfeifer: The Domain and Goals of CSCW. 1995. http://ksi.cpsc.ucalgary.ca/courses/547-95/pfeifer/cscw_domain.html
18. Novotny, J., Tuecke, S., Welch, V.: An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

Adaptive Grid Scheduling of a High-Throughput Bioinformatics Application*

Eduardo Huedo[1], Rubén S. Montero[2], and Ignacio M. Llorente[2,1]

[1] Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas
Centro de Astrobiología (CSIC-INTA)
28850 Torrejón de Ardoz, Spain
{huedoce, martinli}@inta.es
[2] Departamento de Arquitectura de Computadores y Automática
Universidad Complutense
28040 Madrid, Spain
{rubensm, llorente}@dacya.ucm.es

Abstract. Grids provide a way to access the resources needed to execute the compute and data intensive applications required in the Bioinformatics field. However, in spite of the great research effort made in the last years, application development and execution in the Grid continue requiring a high level of expertise due to its heterogeneous and dynamic nature. In this paper, we show the procedure to adapt an existing Bioinformatics application to the Grid using the Grid *W*ay tool. The Grid *W*ay allows the efficient resolution of large computational experiments by reacting automatically to Grid- and application-generated dynamic events.

1 Introduction

The Globus toolkit [1] has become a *de facto* standard in Grid computing. Globus services allow secure and transparent access to resources across multiple administrative domains, and serve as building blocks to implement the stages of Grid scheduling [2]: resource discovery and selection, and job preparation, submission, monitoring, migration and termination. However, the user is responsible for manually performing all the scheduling steps in order to achieve any functionality. Moreover, the Globus toolkit does not provide support for adaptive execution, required in dynamic Grid environments.

Bioinformatics, which relies on the management and analysis of huge amounts of biological data, could enormously benefit from the suitability of the Grid to execute high-throughput applications. Moreover, collections of biological data are growing very fast due to the proliferation of automated high-throughput experimentation processes, and biotechnology organizations. The analysis of the data generated by these high-throughput laboratory techniques will only be possible through high-throughput Grid computing.

* This research was supported by Ministerio de Ciencia y Tecnología through the research grant TIC 2002-00334 and Instituto Nacional de Técnica Aeroespacial.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 840–847, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Probably, one of the most challenging problems that the Grid computing community has to deal with, to efficiently execute applications as the one described above, is the fact that Grids present unpredictable changing conditions, namely:

- *High fault rate:* In a Grid, resource or network failures are the rule rather than the exception.
- *Dynamic resource availability:* Grid resources belong to different administrative domains; so that, once a job is submitted, it can be freely cancelled by the resource owner. Furthermore, the resources shared within a virtual organization can be added or removed continuously.
- *Dynamic resource load:* Grid users access resources that are being exploited by other grid users, as well as by internal users. This fact may cause that initially idle hosts become saturated, and vice versa.
- *Dynamic resource cost:* In an economy driven grid [3], resource prices can vary depending on the time of the day (working/non-working time) or the resource load (peak/off-peak).

Consequently, in order to obtain a reasonable degree of both application performance and fault tolerance, a job must adapt its execution according to grid resource attributes, availability, performance, and cost. Adaptive Grid scheduling has been widely studied in the literature [4,5,6,7,8]; previous works have clearly demonstrated the critical factor of the dynamic information gathered from the grid to generate reliable schedules.

In this paper, we will analyze the Grid execution of an existing Bioinformatics application (section 3) using the Grid *W* ay tool, whose architecture and main functionalities are briefly described in section 2. In section 4 the procedure needed to port the application to the Grid is discussed. Then, in section 5 we will show the benefits of adaptive scheduling in the execution of the Bioinformatics application, to provide both fault tolerance (section 5.1) and performance improvement (section 5.2). The experiments were conducted on the CAB-UCM testbed. The paper ends with some conclusions and hints about future research.

2 The Grid *W* ay Framework

The core of the Grid *W* ay framework [9] is a personal *submission agent* that performs all scheduling stages and watches over the correct and efficient execution of jobs. Adaptation to changing conditions is achieved by dynamic rescheduling of jobs when one of the following circumstances is detected:

1. *Grid-initiated rescheduling:*
   - A new "better" resource is discovered [10].
   - A resource fails or is no longer available.
   - A submitted job is cancelled or suspended.
2. *Application-initiated rescheduling:*
   - Performance degradation or performance contract violation is detected.
   - The resource demands of the application change.

The framework has been designed to be modular. The following modules can be set on a per job basis:

- *resource selector,* which searchs candidate resources meeting the application demands.
- *performance evaluator,* which evaluates the application performance.
- *prolog,* which prepares the remote system and performs input file staging.
- *wrapper,* which executes the actual job and returns its exit code.
- *epilog,* which performs output file staging and cleans up the remote system.

The *submission agent* also provides the application with the fault tolerance capabilities needed in such a faulty environment:

- The GRAM *job manager* notifies submission failures as GRAM callbacks. This kind of failures include connection, authentication, authorization, RSL parsing, executable or input staging, credential expiration and other failures.
- The GRAM *job manager* is probed periodically at each *polling* interval. If the *job manager* does not respond, then the GRAM *gatekeeper* is probed. If the *gatekeeper* responds, a new *job manager* is started to resume watching over the job. If the *gatekeeper* fails to respond, a resource or network failure occurred. This is the approach followed in Condor/G [11].
- The job exit code captured can be used to determine whether the job was successfully executed or not. If the job exit code is not set, the job was prematurely terminated, so it failed or was intentionally cancelled.

When an unrecoverable failure is detected, the *submission agent* retries the submission of *prolog, wrapper* or *epilog* a number of times specified by the user and, when no more retries are left, it performs an action chosen by the user among two possibilities: stop the job for manually resuming it later, or automatically reschedule it.

We have developed both an API and a command line interface to interact with the *submission agent.* They allow scientists and engineers to express their computational problems in a Grid environment. The capture of the job exit code allow users to define complex jobs, where each depends on the output and exit code from the previous job. They may even involve branching, looping and spawning of subtasks, allowing the exploitation of the parallelism on the work flow of certain type of applications.

Our framework is not bounded to a specific class of applications, does not require new services, and does not necessarily require source code changes. We would like to remark that the Grid *W* ay framework does not require new system software to be installed in the Grid resources. The framework is currently functional on any Grid testbed based on Globus. We believe that is an important advantage because of socio-political issues: cooperation between different research centers, administrators, and users can be very difficult.

3 The Target Application

As target application, we have used a Bioinformatics application aimed at predicting the structure and thermodynamic properties of a target protein from its amino acid sequences. The algorithm [12], tested in the 5th round of Critical Assessment of techniques for protein Structure Prediction (CASP5), aligns with gaps the target sequence with all the 6150 non-redundant structures in the Protein Data Bank (PDB), and evaluates the match between sequence and structure based on a simplified free energy function plus a gap penalty term. The lowest scoring alignment found is regarded as the prediction if it satisfies some quality requirements. For each sequence-structure pair, the search of the optimal alignment is not exhaustive. A large number of alignments are constructed in parallel through a semi-deterministic algorithm, which tries to minimize the scoring function.

We have applied the algorithm to the prediction of thermodynamic properties of families of orthologous proteins, i.e. proteins performing the same function in different organisms. If a representative structure of this set is known, the algorithm predicts it as the correct structure. The test presented in this work is an example of this application, where we have applied the structure prediction algorithm to 88 sequences of the Triose Phosfate Isomerase enzyme present in different organisms. The whole experiment was submitted as an array job, where each sequence was analyzed in a separate task of the array, specifying all the needed information in a `job template` file. The results of the comparative study of this and other proteins are presented elsewhere.

4 Changes in the Application to Be Grid-Aware

Due to the high fault rate and the dynamic rescheduling, the application must generate `restart files` in order to restart the execution from a given point. If these files are not provided, the job is restarted from the beginning. User-level checkpointing managed by the programmer must be implemented because system-level checkpointing is not currently possible among heterogeneous resources. The application has been modified to periodically generate an architecture independent `restart file` that stores the best candidate proteins found to that moment and the next protein in the PDB to analyze.

In order to detect performance slowdown, the application is advised to keep a `performance profile` with its performance activity in terms of application intrinsic metrics. We have modified the application to provide a `performance profile` that stores the time spent on each iteration of the algorithm, where an iteration consists of the analysis of a given number of sequences.

In order to adapt the execution of a job to its dynamic demands, the application must specify its host requirements through a `requirement expression`. The application could define an initial set of requirements and dynamically change them when more, or even less, resources are required. Also, in order to prioritize the resources that fulfil the requirements according to its runtime

Table 1. The CAB-UCM research testbed.

Name	Model	Nodes	Speed	OS	Memory	VO	GRAM
ursa	Sun Blade 100	1	500MHz	Solaris 8	256MB		fork
draco	Sun Ultra 1	1	167MHz	Solaris 8	128MB	DACYA	fork
pegasus	Intel Pentium 4	1	2.4GHz	Linux 2.4	1GB		fork
solea	Sun Enterprise 250	2	296MHz	Solaris 8	256MB	QUIM	fork
babieca	Compaq Alpha DS10	5	466MHz	Linux 2.2	256MB	CAB	PBS

needs, the application must specify its hosts preferences through a `ranking expression`. A compute-intensive application would assign a higher rank to those hosts with faster CPUs and lower load, while a data-intensive application could benefit those hosts closer to the input data.

In the experiments described in the next section the application does not impose any requirement to the resources. The `ranking expression` uses a performance model to estimate the job turnaround time as the sum of execution and transfer time, derived from the performance and proximity of the candidate resources [13]. The application doesn't dynamically change its resource demands.

The `requirement expression` and `ranking expression` files are used by the *resource selector* to build a list of potential execution hosts. Initially, available compute resources are discovered by accessing the GIIS server and those resources that do not meet the user-provided requirements are filtered out. At this step, an authorization test is performed to guarantee user access. Then, the resource is monitored by accessing its local GRIS server. The information gathered is used to assign a rank to each candidate resource based on the user-provided preferences. Finally, the resultant prioritized list of candidate resources is used to dispatch the jobs.

In order to reduce the information retrieval overhead, the GIIS and GRIS information is locally cached at the client host and updated independently in order to separately determine how often the testbed is searched for new resources and the frequency of resource monitoring. In the following experiments we set the GIIS cache timeout to 5 minutes and the GRIS cache timeout to 1 minute.

5 Experiences

We have performed the experiments in the CAB-UCM research testbed, depicted in table 1. The testbed is highly heterogeneous and dynamic, and consists of three *virtual organizations* (VO). QUIM and DACYA VOs are connected through a metropolitan area network belonging to the UCM, and both are connected to the CAB VO through a wide area network belonging to the RedIRIS spanish academic network.

The experiment files consists of: the executable (0.5MB) provided for all the resource architectures in the testbed, the PDB files shared and compressed

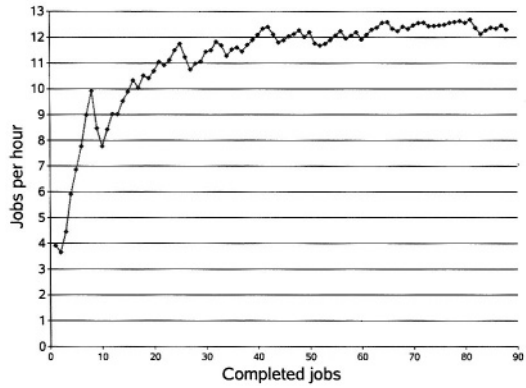


Fig. 1. Throughput when all machines in the testbed were up.

(12.2MB) to reduce the transfer time, some parameter files (1KB), and the file with the sequence to be analyzed (1KB). The final file name of the executable is obtained by resolving the variable `GW_ARCH` at runtime for the selected host, and the final name of the file with the sequence to be analyzed, with the variable `GW_TASK_ID`. Input files can be local or remote (specified as a GASS or GridFTP URL), and both can be compressed (to be uncompressed on the selected host) and declared as shared (to be stored in the GASS cache and shared by all the jobs submitted to the same resource).

Figure 1 shows the behaviour when all machines in the testbed were up. Total experiment time was 7.15 hours, and the mean throughput was 12.30 jobs/hour, which supposes a mean job turnaround time of 4.88 minutes.

5.1 Adaptive Scheduling to Provide Fault Tolerance

Figure 2 shows the attained throughput when babieca was temporarily shutdown for maintenance. Total experiment time was 7.31 hours (only 9.6 minutes more than the previous experiment), and the mean throughput was 12.04 jobs/hour, which supposes a mean job turnaround time of 4.98 minutes. The mean throughput dropped from 12.71 to 10.61 jobs/hour when babieca was down, but when it went up, it begun to grow to 12.04 jobs/hour.

5.2 Adaptive Scheduling to Provide Performance Improvement

Figure 3 shows the attained throughput when pegasus was discovered in the middle of the experiment, because it was turned on in that moment. Total experiment time was 8.65 hours, and the mean throughput was 10.17 jobs/hour, which supposes a mean job turnaround time of 5.9 minutes. Before discovering pegasus, the mean throughput was only 8.31 jobs/hour, and after that, it increased to 10.17 jobs/hour.

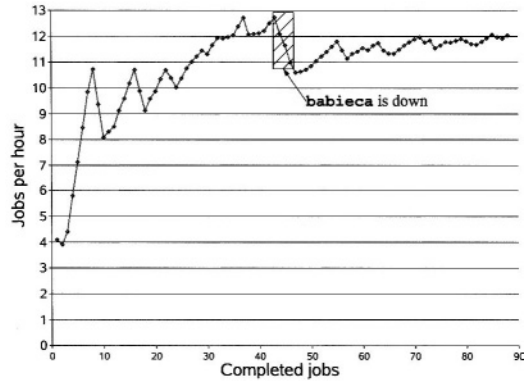


Fig. 2. Throughput when babieca was temporarily down.

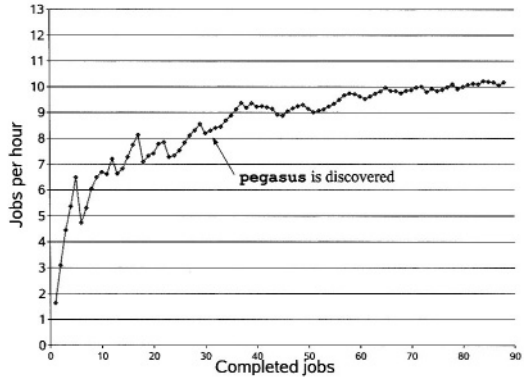


Fig. 3. Throughput when pegasus was discovered in the middle of the experiment.

6 Conclusions

We have tested the Grid *W* ay tool in our research testbed with a high-throughput application. We have seen the benefits of adaptive scheduling to provide both fault tolerance and performance improvement. In a future work we will study the adaptation of the application execution when there are more nodes than tasks to be executed in the Grid. In this case, tasks allocated to slow nodes, would be migrated to the available fast nodes when the *performance evaluator* detects a performance slowdown or the *resource selector* finds them as better resources, thus providing *preemptive scheduling* that will promote the application performance.

This promising application shows the potentiality of the Grid to the study of large numbers of protein structures, and suggests the possible application of this methods to the whole set of proteins in a complete microbial genome.

Acknowledgments. We would like to thank Ugo Bastolla, staff scientist at Centro de Astrobiología and developer of the Bioinformatics application utilized in the experiments, for his support on understanding and modifying the application.

References

1. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl. J. of Supercomputer Applications **11** (1997) 115–128
2. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report WD8.5, Global Grid Forum (2001) Scheduling Working Group.
3. Buyya, R., D.Abramson, Giddy, J.: A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems (2002) Elsevier Science (to appear).
4. Berman, F., et al.: Adaptive Computing on the Grid Using AppLeS. IEEE Transactions on Parallel and Distributed Systems **14** (2003) 369–382
5. Buyya, R., Abramson, D., Giddy, J.: Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computation Grid. In: Proceedings of the 4th IEEE Intl. Conference on High Performance Computing in Asia-Pacific Region (HPC Asia). (2000) Beijing, China.
6. Wolski, R., Shao, G., Berman, F.: Predicting the Cost of Redistribution in Schedulling. In: Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Applications. (1997)
7. Allen, G., et al.: The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. Intl. Journal of High-Performance Computing Applications **15** (2001)
8. Vadhiyar, S., Dongarra, J.: A Performance Oriented Migration Framework for the Grid. In: Proceedings of the 3rd IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid). (2003)
9. Huedo, E., Montero, R.S., Llorente, I.M.: An Experimental Framework for Executing Applications in Dynamic Grid Environments. Technical Report 2002-43, ICASE – NASA Langley (2002) To appear in Intl. J. of Software – Practice and Experience.
10. Montero, R.S., Huedo, E., Llorente, I.M.: Grid Resource Selection for Opportunistic Job Migration. In: Proceedings of Intl. Conf. on Parallel and Distributed Computing (EuroPar 2003). LNCS, Springer-Verlag (2003)
11. Frey, J., et al.: Condor/G: A Computation Management Agent for Multi-Institutional Grids. In: Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10). (2001)
12. Bastolla, U.: Sequence-Structure Alignments With the Protfinder Algorithm. In: Abstracts of Fifth Community Wide Experiment on the Critical Assessment of Techniques for Protein Structure Prediction. (2002) Available at http://predictioncenter.llnl.gov/casp5.
13. Huedo, E., Montero, R.S., Llorente, I.M.: Experiences on Grid Resource Selection Considering Resource Proximity. In: Proceedings of 1st European Across Grids Conference. (2003)

Advanced Replica Management with Reptor*

Peter Kunszt, Erwin Laure, Heinz Stockinger, and Kurt Stockinger

CERN, European Organization for Nuclear Research, Switzerland

Abstract. Data replication is one of the best known strategies to achieve high levels of availability and fault tolerance, as well as minimal access times for large, distributed user communities using a world-wide Data Grid. In certain scientific application domains the data volume can reach the order of several petabytes; in these domains data replication and access optimization play an important role in the manageability and usability of the Grid.
In this paper we present the design and implementation of our replica management Grid middleware providing a high-level user and application interface to replication services. It is designed to be extensible and evolvable so that user communities can adjust its detailed behavior according to their QoS requirements. Our prototype implementation is based on the web service paradigm, in accord with the emerging Open Grid Services Architecture, and provides promising results.

1 Introduction

Grid computing addresses the issue of distributed computing over the wide-area network that involves large-scale resource sharing among collaborations of individuals or institutions. We distinguish between computational Grids and data Grids: Computational Grids address computationally intensive applications that deal with complex and time intensive computational problems usually on relatively small data sets whereas data Grids address the needs of data intensive applications that deal with the evaluation and mining of large amounts of data in the terabyte and petabyte range.

One of the principal goals of data Grids is to provide easy-to-use, transparent access to globally distributed data and to abstract the complexities from the user and the applications. Essentially, it aims at making data access and location as easy as on a single computer. In order to achieve this goal, the most important issues that need to be addressed are how to optimize access to data over the wide area to avoid large penalties on data access times, and how to provide a solid, highly extensible and performing security and data access policy framework.

Optimization of data access can be achieved via data replication, whereby identical copies of data are generated and stored at various sites. This can significantly reduce data access latencies. However, dealing with replicas of files adds a number of problems not present when only a single file instance exists.

* This work was partially funded by the European Commission program IST-2000-25182 through the EU DataGrid Project.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 848–855, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Replicas must be kept consistent and up-to-date, their locations must be stored in catalogs, their lifetimes need to be managed, etc.

Many underlying services required for replica management have been developed: file transfer services and protocols (GridFTP), replica catalogs, and security mechanisms. Additional services are under development: data access and integration services, replica location services, etc. However, using these services in isolation is an error-prone and tedious task, hence there is a clear need for higher level services that abstract all the intricacies of the basic Grid services from the users.

In this paper we present the design and implementation of a replica management service that is intended to provide the application programmer with an easy-to-use, intuitive interface, hiding the details of the underlying services. We have designed and implemented a prototype, "Reptor", in the context of the EU DataGrid project (EDG). Experimental work shows promising results. A more detailed discussion of Reptor can be found at [9].

2 Design of a Replica Management Framework

In [8] we presented the design of a general replica management framework. In that model, a replica management service contains all the logic to coordinate the underlying services, providing the user with a unified interface to all replica management functionalities. The entry point to all services of our replica management system [8] is the **Replica Management Service** which interacts with the following subcomponents of the system:

- The **Core** module coordinates the main functionality of replica management, which is replica creation, deletion, and cataloging by interacting with third party modules. These external modules include transport services, replica location services, meta-data services for storing replication meta-data such as file meta-data (size, checksum, etc.), management meta-data, and security meta-data (such as access control lists), and processing services that allow pre- and post-processing of files being replicated.
- The goal of the **Optimization** component is to minimize file access times by pointing access requests to appropriate replicas and pro-actively replicating frequently used files based on access statistics gathered.
- The **Consistency** module takes care of keeping the set of replicas of a file consistent as well as the meta information stored in various catalogs.
- The **Subscription** module takes care of subscription-based replication where data appearing at a data source is automatically replicated to subscribed sites.
- The **Session Management** component provides generic check-pointing, restart, and rollback mechanisms to add fault tolerance to the system.
- **Collections** are defined as sets of logical filenames and other collections.
- The **Security** module manages the required user authentication and authorization, in particular, issues pertaining to whether a user is allowed to create, delete, read, and write a file.

In our current implementation, the logic of the coordinating replica management service is implemented on the client-side, i.e. there is no actual service associated with it; this is to defer the issues we face with the details of authorization with respect to delegation of rights to services. The other subservices and APIs are modules and services in their own right, allowing for a multitude of deployment scenarios in a distributed environment.

One advantage of such a design is that if a subservice is unavailable, the Replica Manager Client can still provide all the functionality that does not make use of that particular service. Also, critical service components may have more than one instance to provide a higher level of availability and to avoid service bottlenecks. We foresee to provide consistent replicas for stateful services; services like the Replica Location Service are already designed to be distributed in their own right [6].

3 Prototype Implementation

We implemented a first prototype of the replica management system, named "Reptor", within the framework of the EU DataGrid (EDG) project. The services are all implemented in the Java language. The servers are deployed as web services. We depend only on open source technologies like the Jakarta Tomcat servlet container, the MySQL database and Apache Axis.

The prototype implementation provides the core module interfacing to the transport and catalog services as well as on the optimization component, leaving the collection, session management, and consistency modules for future versions.

Security is provided within the Grid Security Infrastructure (GSI) framework. Our web services are able to authenticate Grid certificates via our trust manager, an authentication module which can be plugged into web service containers. We also have a fine grained authorization mechanism in place that can interpret certificate extensions provided by the Virtual Organization Membership Service VOMS [1]. The advantage of such an approach is that our security infrastructure is backward compatible with the existing GSI implementations that simply ignore these extensions.

In wide area distributed computing, web service technologies are becoming increasingly popular since they provide easy and standardized access to distributed services in a multi-lingual, multi-domain environment. The same is true in Grid computing where the upcoming OGSA standard aims at leveraging web services in the Grid context. Due to the general recognition of the web service standard and to be prepared to move to the OGSA standard, we have chosen to adopt the web services paradigm.

The control and data channels between the services and their clients is managed through interfaces published in WSDL. For the control channel we use SOAP over HTTPS. For data channels that require higher levels of performance, we rely on the existing mechanisms like GridFTP for file transport.

We have found that the available technologies to build standard web service components already address many of the issues of Grid computing and we could profit from the extensive code base made available by the web service community.

Interaction with External Services. As described above, Reptor needs to interact with many Grid services such as the replica location service and the information services. We have implemented Reptor as set of modules that is easy to extend and interface to other Grid components.

We have defined interfaces for each Grid service that Reptor is accessing. In the spirit of general software engineering principles as well as OGSA, we have made all these modules pluggable: if one of the services needs to be replaced by another service providing the same functionality, the appropriate implementation of the interface needs to be provided to Reptor, which can be configured to load the given implementation instead of the default ones. Currently, Reptor has been tested with the following Grid services:

- *Replica Location Service (RLS)* as the replica catalog service: used for locating replicas by their unique identifiers (GUIDs).
- *Replica Meta-data Catalog (RMC)* stores the user-definable logical file name aliases to GUIDs and simple meta data on replicas, e.g. the owner, file size, and time stamps.
- *Relational Grid Monitoring Architecture (R-GMA)* as the information service: used for obtaining information about the topology of the Grid. In addition, we have interfaced Reptor with *Globus' MDS.*
- The *EDG Network Monitoring Services* providing statistics about network characteristics.
- The *EDG Storage Element Services* providing information about the storage latency.

The Reptor Client. We provide a command line interface and an API for the clients. The client is a thin layer that interacts with the internal and external services, acting as the coordinator of the replica management sub-services.

We have a pure Java version of the client service using the Commodity Grid for Java *CoG* that provides GridFTP client functionality. However, in order to be able to support parallel streams for better performance, we can also configure the Reptor client to use the native GridFTP C libraries where available. The current client functionalities are summarized in Table 1.

The Optimization Service. The goal of the optimization service is to select the best replica with respect to network and storage access latencies. In other words, if for a given file several replicas exist, the optimization service determines the replica that should be accessed from a given location. Similarly, the optimization service might also be used to determine the best location for new replicas. We currently do not take into account an eventual possibility to access the data directly over the wide area — this is the subject of ongoing work — but assume that in order to access a file it has to be available in the local area network.

Table 1. Main Replica Manager Commands

Command	Description
registerFile	register file with the Replica Location Service (RLS)
listReplicas	list all replicas registered with RLS
copyAndRegisterFile	copy file to SE and register with RLS
replicateFile	replicate file to SE and register with RLS
listBestFile	return replica with minimal transfer time no replication is involved
unregisterFile	unregister file from RLS
deleteFile	unregister file from RLS and delete from SE
getBestFile	return replica with minimal transfer time this command might involve replication
getAccessCost	get access cost of replicas for specific destination according to the network cost function

The *Replica Optimization Service (ROS)* is implemented as a light-weight web service (called *Optor*). It gathers information from the EDG network monitoring service and the EDG storage element monitoring service about their network and storage access latencies. Based on this information Optor takes a decision which network link should be used in order to minimize the transfer time between two end points as described in [4].

4 Experimental Results

All our experiments were run on five major sites of the EU DataGrid testbed in France, Italy, the Netherlands, Switzerland and the United Kingdom. Each site consists of *Computing Elements* (CE) and *Storage Elements* (SE). The components needed by the of Replica Manager Client were deployed at dedicated hardware at CERN, each machine running a single service only. We had one machine each to run the User Interface (i.e. the Replica Manager Client), the Replica Location Service, the Replica Optimization Service and the Information Service (R-GMA in our case).

The network monitoring components have been deployed at each testbed site by the EDG networking work package.The monitors show that there are very large deviations in network performance over time. The spikes and dips in these network metrics usually span tens of minutes and hours — this motivates our strategy of optimization based on recent network metrics.

Reptor provides two different replication options, *conventional replication* and *optimized replication.* The conventional option takes source and destination SEs as input parameters. The optimized option takes a logical file name (LFN) and the destination SE as input parameters and chooses the best replica as described above: For a given LFN, all physical replicas are identified through the RLS and for each replica the transfer costs to the specified destination SE are calculated using the Network Monitor. The replica having the minimal cost is used as the source file for replication.

By having both options, users may either control the source and destination of the copy themselves or let the system decide on the best resources to use on their behalf.

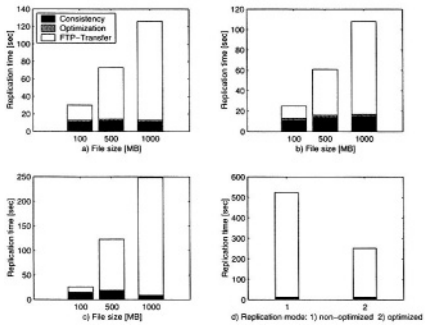


Fig. 1. Performance measurements of replicating files with sizes 100MB, 500MB and 1GB between a) CERN and RAL, b) RAL and NIKHEF, and c) NIKHEF and CNAF. d) Performance measurements of replicating 1GB files to NIKHEF 1) using conventional replication: IN2P3 was specified as the source and 2) using optimized replication: RAL was picked as the better source automatically.

In Figure 1 we show our measurements for replicating files between three sites within the EU DataGrid testbed. The file sizes range from 100 MB to 1 GB. We use GridFTP as the file transfer mechanism with 8 parallel streams. Note that Reptor adds some overhead to the basic file transfer due to various consistency checks: Reptor checks whether the file already exists on the destination before starting the copy, obtains file meta-data, and checks the file existence after having performed the copy. This overhead is constant, independent of the actual file size. For replicating 1 GB files we can observe an overhead of around 10% due to the additional consistency checks.

The next experiment shows the possible performance gain one can obtain by optimized replication (see Figure 1 d)). Due to the different network bandwidths within the testbed and their rapid change in time we gain factors of two and more in transfer efficiency by automatically choosing the best network link for file replication. If the users have to specify the source for the copy by themselves, they can achieve at best the same result. Without an automatic replica selection mechanism they most likely will choose a suboptimal network link. In both cases, the consistency overhead can be neglected since it is identical for both replication scenarios.

To sum up, we demonstrated network optimized replication with Reptor and achieved a substantial gain in efficiency especially in a distributed Grid environment where the network latencies are subject to rapid change. We have found that the changes are slow enough so that recent network monitoring information is usually still applicable at the time of the file transfer.

5 Related Work

Early prototype work has been performed with a data replication tool named *Grid Data Mirroring Package (GDMP)* [10]. It is a basic file replication tool with support for mass storage system staging and automatic replication through a publish-subscribe notification system.

The Globus project provides in its toolkit a replica catalog based on the LDAP protocol and a straightforward replica manager that can manage the copy and registration as a single step. We have been using and extending these modules within the EU DataGrid project by a user-friendly API and mass storage bindings. The *edg-replica-manager* can be regarded as the direct predecessor of Reptor. However, neither the Globus replica manager, nor edg-replica-manager nor GDMP provide replica optimization or any other of the capabilities that we foresee for the immediate future.

An integrated approach for data and meta-data management is provided in the Storage Resource Broker (SRB) [3]. Data cataloging and access to relational and object-oriented databases is provided. Replica optimization is not provided yet but SRB might profit from approaches like ours.

The most related work with respect to replica access optimization can be found in the Earth Since Grid (EDG) [2] project where preliminary replica selection has been demonstrated using the Network Weather Service (NWS). Rather than providing an integrated replication framework with consistency checks etc., GridFTP transfer sources and destinations are selected via NWS.

Within the Grid and High-Energy Physics community, one of the most closely related projects is SAM [11] (Sequential data Access via Metadata) that was initially designed to handle data management issues of the D0 experiment at Fermilab. It also addresses replication job submission issues.

Finally, since file replication often deals with large storage systems, local storage resource management becomes vital and we participate in the standardization effort of the Storage Resource Management (SRM) [5] specification that builds the natural link between Globus, Grid-wide replica management and local file management.

6 Conclusion and Future Work

We have presented the design and a prototype implementation of a high level replica management tool providing the functionality required for efficient data access in a Grid environment. We have discussed our optimized replica selection mechanism that takes into account network monitoring data in order to identify the optimal network link with respect to data transfer. The experimental results show that our approach significantly reduces wide area transfer times.

Currently, we have implemented the core and optimization functionality of the replica management design. In the immediate future we will extend the functionality by adding the first versions of the collection and consistency modules to Reptor. The consistency module is based on some of our previous ideas [7] and will contain a file versioning and update mechanism based on file replacement or binary difference between two files.

Future work also includes the refining and exact definition of the semantics for the session management module. The security framework in the current Reptor prototype provides authentication based on GSI and will be extended to

allow for fine-grained authorization to be achieved through a role based authentication/authorization management system [1].

Acknowledgments. We would like to thank first and foremost our colleagues in WP2 who have had contributions to the design and development of Reptor and for running the experimental testbed: Diana Bosio, James Casey and Leanne Guy. We would like to thank our colleagues in the EU DataGrid project who provided us with invaluable feedback on our existing replica management tools as well as supported us in the setup of the experimental testbed. We are also thankful to Ann Chervenak and Carl Kesselman for many stimulating discussions.

References

1. R. Alifieri, R. Cecchini, V. Ciaschini, L. Agnello, A. Frohner, A. Gianoli, K. Loerentey, and F. Spataro. An Authorization System for Virtual Organizations. In *European Across Grids Conference,* Santiago de Compostela, Spain, February 2003.
2. B. Allcock, I. Foster, V. Nefedov, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Drach, and D. Williams. High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies. In *Supercomputing 2001,* Denver,Texas, November 2001.
3. C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Research Broker. In *CASCON'98,* Toronto, Canada, 30 November - 3 December 1998.
4. W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Design of a Replica Optimisation Framework. Technical Report DataGrid-02-TED-021215, CERN, Geneva, Switzerland, December 2002. EU DataGrid Project.
5. I. Bird, B. Hess, A. Kowalski, D. Petravick, R. Wellner, J. Gu, E. Otoo, A. Romosan, A. Sim, A. Shoshani, W. Hoschek, P. Kunszt, H. Stockinger, K. Stockinger, B. Tierney, and J.-P. Baud. Srm joint functional design. Global Grid Forum Document, GGF4, Toronto, February, 2002.
6. A. Chervenak, E. Deelman, I. Foster, L. Guy, A. Iamnitchi, C. Kesselmanand, W. Hoschek, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A Framework for Constructing Scalable Replica Location Services. In *SC'2002,* Baltimore, USA, November 2002.
7. D. Düllmann, W. Hoschek, J. Jean-Martinez, A. Samar, B. Segal, H. Stockinger, and K. Stockinger. Models for Replica Synchronisation and Consistency in a Data Grid. In *10thIEEE Symposium on High Performance and Distributed Computing (HPDC-10),* San Francisco, California, August 7-9 2001.
8. L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Replica Management in Data Grids. Technical report, GGF5 Working Draft, July 2002.
9. P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Replica Management with Reptor. Technical Report DataGrid-02-TED-030408, DataGrid, April 2003.
10. H. Stockinger, A. Samar, S. Muzaffar, and F. Donno. Grid Data Mirroring Package (GDMP). *Journal of Scientific Programming,* 10(2), 2002.
11. I. Terekhov, R. Pordes, V.White, L. Lueking, L. Carpenter, H. Schellman, J. Trumbo, S. Veseli, and M. Vranicar. Distributed Data Access and Resource Management in the D0 SAM System. In *10thIEEE Symposium on High Performance and Distributed Computing (HPDC-10),* San Francisco, California, August 7-9 2001.

SuperVise: Using Grid Tools to Simplify Visualization

James Osborne and Helen Wright

Department of Computer Science, University of Hull, UK

Abstract. The visualization process is complicated enough for visualization experts but more so for domain specialists. Problems include managing the transfer of large datasets and choosing appropriate techniques. When we consider performing these tasks in a modern visualization laboratory setting, with multiple software packages producing output on a range of devices, the problems become even more intractable. The practicalities of developing such complicated visualizations necessitate the presence of a visualization engineer, or visioneer, to drive the hardware and software based on requirements elicited from the domain specialist. Using SuperVise the visioneer avoids repeating the implementation of given visualization techniques. SuperVise decouples the visualization processing from the display of results, allowing domain specialists to make greater use of available software and hardware, using different packages and on different devices, with a minimum of specialist knowledge.

1 Introduction

Visualization laboratories contain a diverse range of hardware and software and using these resources to produce modern visualizations requires the presence of a visioneer. Management of resources and knowledge of the idiosyncrasies of hardware and software is the major factor preventing domain specialists from making use of these systems unaided.

The field of grid computing has delivered many promising technologies to help in the management of resources and job execution. By developing a higher-level, task-oriented model of visualization amenable to execution across the grid, we have addressed the problems faced by domain specialists in a modern visualization laboratory setting.

In section 2 of this paper we first outline the problems faced by domain specialists in today's visualization laboratories. In section 3 we discuss a conceptual model that looks upon visualization as a series of tasks amenable to execution in a grid environment. We then go on to discuss the core components of the SuperVise system itself: the desktop, viewer, middleware, grid tools, visualization tools and repositories. We conclude in section 5 with a discussion of the current system status and our future plans.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 856–863, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 The Problem

In a modern visualization laboratory such as the HIVE (Hull Immersive Visualization Environment), a wide range of visualization hardware and software is available. The facilities offered by the HIVE include a 7m x 2m back-projected stereo display wall, a 4m x 2m front-projected stereo display wall, a hemisphere display, a number of head mounted displays, a number of stereo-capable workstations, a number of haptic-capable workstations and a High Performance Computing (HPC) cluster. The HIVE also offers a range of visualization software, from graphics and visualization libraries such as OpenGL and VTK, through to Modular Visualization Environments (MVEs) such as IRIS Explorer and OpenDX.

Two main user groups are supported by the HIVE. The first group, visioneers, are familiar with most if not all of the complicated hardware and software available. They are also able to produce visualizations for the second group, domain specialists. This group understands the value that visualization adds to their research but may be unfamiliar with visualization systems. We aim to make a wider range of visualization tools more accessible to domain specialists and reduce the requirement for assistance from visioneers, thereby enriching the domain specialists' experience in the HIVE.

There are three main issues that must be addressed. The first is how to manage a wide range of diverse resources – these can include local HIVE resources in terms of visualization hardware and software systems, as well as access to data in local and remote locations. The second issue is job submission; we may wish to perform visualization processing on a range of hardware, such as a local workstation if the task does not require extensive processing, or on the HPC cluster for a task that does. Finally there is the issue of security; we may wish to deny access to a subset of the available hardware if the domain specialist is not trained in its use. We may also wish to offer encryption of sensitive data whilst in transit.

Computational 'grids', by analogy with the electricity grid, are a mechanism whereby scientists can bring computing power from diverse sources to bear on complex problems. Issues addressed by the grid community include resource management, job scheduling, data transport, user authentication and encryption – precisely the same as those faced by domain specialists working in the HIVE.

3 Conceptual Model

There are already a number of visualization models such as those developed by Upson et. al. [1] and Haber & McNabb [2]. Upson et. al's model explains visualization in terms of a series of processes, whereas Haber & McNabb's model explains visualization in terms of a series of states. These models describe a 'visualization pipeline', where raw data input at one end is gradually transformed into an image output at the other end.

These models of visualization use implementation level terms to describe the data transformations. To support domain specialists a new conceptual model

is required that concentrates on the tasks to be accomplished, rather than the data transformations involved, thus our new model of visualization draws on the user's experience to describe the process of visualization. Three processes are involved in specifying a visualization in this way, namely, the specification of the input format of their data, the technique to use to visualize it, and their requirements for output, which might be stereo rendering, head tracked camera control, haptic interaction and so on.

Our pipeline model (figure 1) therefore consists of three phases or tasks, namely, input format conversion, core technique selection, and output conversion and display. The corresponding implementation level processes and states occurring in existing models are shown in grey.

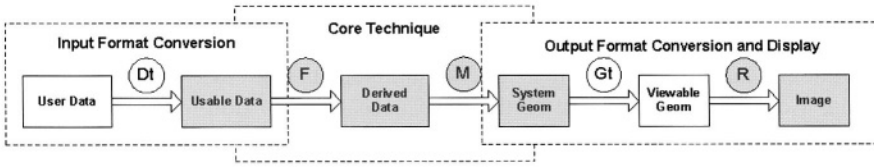


Fig. 1. The SuperVise Conceptual Model

The User Data state has been added to represent raw input data in a given format. The Data Transformation process (Dt) represents the transformation of input data from one format into another with no loss of information. The need to transform data from one format to another which is usable by the visualization system is a common requirement but it is not traditionally considered in conceptual models. The Filtering process (F) might involve interpolation if the data is sparsely arranged, or selective reduction if there is too much data. The Map process (M) represents the transformation of Derived Data into geometry. The System Geometry state represents output geometry in a format native to the visualization system, whilst the Geometry Transformation process (Gt) represents the transformation of this native geometry into another format that is supported by the chosen viewer, not necessarily that shipped with the visualization system that provides the core technique. The Viewable Geometry state has been added to represent this converted format. The Render process (R) represents the viewing of this converted geometry on some display device to produce an Image.

This model suggests an architecture that allows phases of the visualization pipeline to be executed on different host systems such as nodes in a computational grid. The model also allows these phases to be implemented in a number of different ways, using visualization APIs, turnkey systems, MVEs, or a combination of these.

4 The SuperVise System

The SuperVise desktop is an implementation of our conceptual model. The user is presented with an interface which allows phases corresponding to our conceptual model to be used to build a visualization which is subsequently executed and displayed by one of a number of viewers.

The system additionally consists of a number of parts that the user is unaware of. These include middleware, grid tools, visualization tools and repositories. The middleware handles the visualization workflow of job submission to appropriate nodes. The grid tools provide facilities for resource management, job submission, data transport, user authentication and encryption. The visualization tools, in our case the MVE IRIS Explorer, are used on each grid node to carry out the visualization processing. The repositories hold visualization pipeline fragments that can then be used by the system.

The complete SuperVise system architecture is presented in figure 2, whilst the remainder of this section describes the core components of the SuperVise system in greater detail.

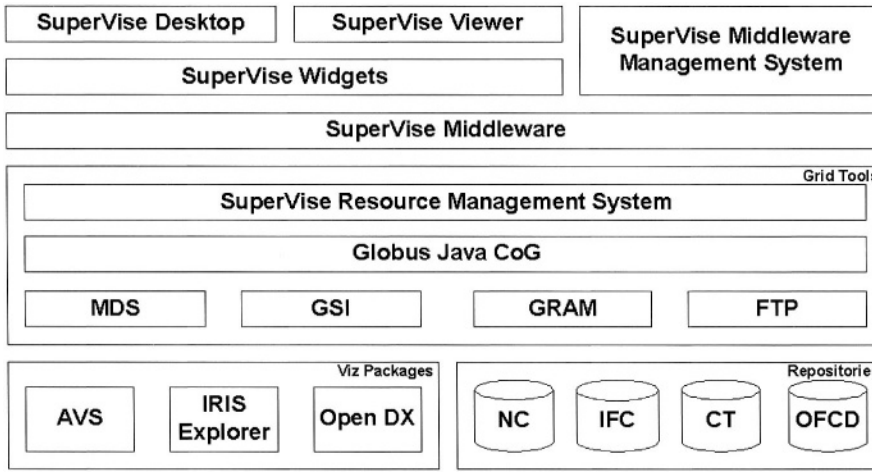


Fig. 2. SuperVise System Architecture

4.1 Desktop

The SuperVise desktop provides the user interface to the system (see figure 3).

The desktop consists of three tabbed panes. The Credentials tab allows a user to enter their Globus grid credentials to allow subsequent access to visualization processing resources. The Processing tab allows the user to enter the location and format of data files and to specify output requirements such as stereo rendering,

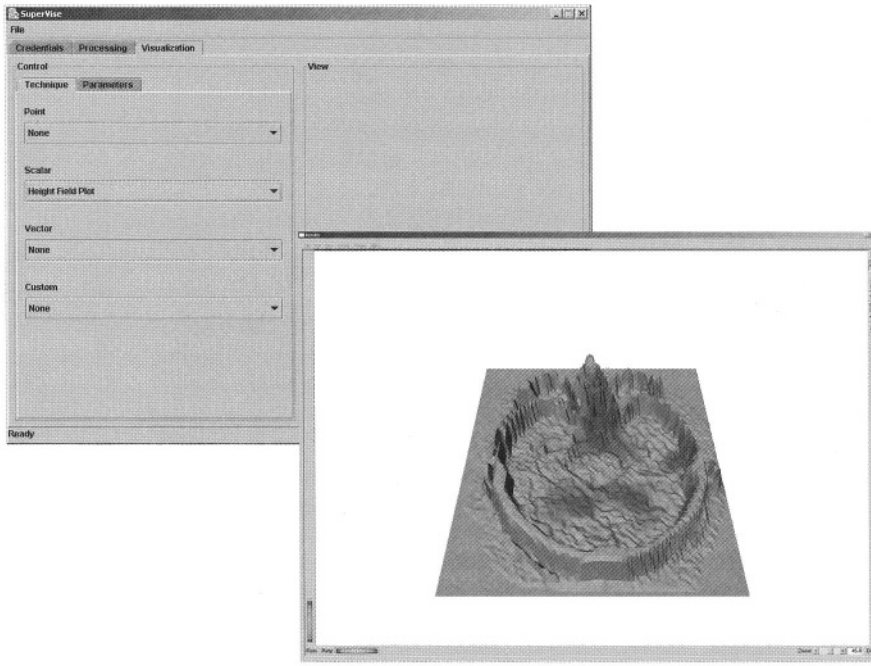


Fig. 3. SuperVise Desktop and Example Visualization

head tracked camera control, haptic interaction and so on. The Visualization tab allows the user to specify the visualization technique and to alter the parameters controlling it. A standalone viewer is invoked to display the resulting geometry.

Currently the user needs to know which nodes can run jobs (grid nodes with visualization systems installed locally), which nodes support stereo output and so on. We envisage that in future SuperVise will perform job execution on suitable grid nodes in autonomous fashion as well as suggest a list of alternatives if the user's local workstation does not support the output requirements they have specified.

4.2 Viewer

In visualization it is common to build specialised viewers to work on specific display systems utilising various devices such as head tracking equipment and stereo glasses. VRML is a commonly-used, but by no means the only, geometry input format for such viewers. In SuperVise the user's Output Requirements, as specified via the Processing tab on the desktop, dictate which viewer is needed in the Output Format Conversion and Display phase and therefore what geometry format must ultimately be produced. Conversion to this format from that produced by the core technique, if required, is also the responsibility of this phase.

4.3 Middleware

The middleware consists of a number of objects that encapsulate visualization pipeline fragments to perform Input Format Conversion (IFC), Output Format Conversion and Display (OFCD), and to execute the Core Technique (CT). These middleware objects are stored in repositories as discussed in section 4.6. Each object also contains methods to generate a proxy based on a Globus certificate that enable access to a particular resource. As well as this authentication step, the middleware manages the generation of Globus scripts and their subsequent submission to a given host to invoke the visualization software that will perform the processing. Scripts to drive the visualization system are then used in order to produce the desired result.

In the future, the middleware objects will be able to probe their environment to decide which underlying visualization package to use. This probing ability, in conjunction with the Node Capabilities Repository (see section 4.6), will then enable autonomous selection of visualization processing and display resources.

4.4 Grid Tools

The Globus grid toolkit developed by Foster & Kesselman [3] provides the underlying grid technology for the system. Globus is used as the facilitating technology to manage a number of tasks and resources involved in a typical visualization workflow in the HIVE.

The Java Commodity Grid (CoG) toolkit developed by von Laszewski [4] is used to connect our Supervise desktop interface, implemented in Java, to the back-end grid services that provide the system with facilities for resource management, job submission, data transport, user authentication and encryption.

MDS, a component of the Globus toolkit, can be used to find out machine details such as internal hardware, operating system and job manager. In the future, to facilitate appropriate visualization processing and display resource selection, we envisage the storage of additional characteristics such as installed visualization software and hardware capability.

We are using the concept of a proxy credential, provided as part of the Globus Security Infrastructure (GSI), to manage authentication and access for the user. Each grid user has a proxy certificate, given out by a trusted certificate authority. These user certificates are then authenticated at each node the user wishes to access.

The Grid Resource Access Management (GRAM) component of Globus is used to manage the submission of visualization jobs, such as executing a range of core techniques on a range of local grid nodes. GRAM makes use of the proxy certificate for authentication.

Grid-FTP is used to provide a mechanism for data transport between grid nodes and uses the proxy certificate for authentication. The original input data may be in a number of different locations, such as on the local hard disk, available on the internet at a given URL or available via FTP.

4.5 Visualization Tools

The system currently uses IRIS Explorer, an MVE, to execute visualization pipeline fragments. In normal usage its interface consists of a map editor and a librarian, used together in a visual programming paradigm. The user picks up modules from the librarian, places them in the map editor (a 2D visual programming area) and wires them together to build a visualization pipeline, or map. The learning curve of the system in this mode is steep and is one of the complexities that we wish to hide from the domain specialist.

In contrast, SuperVise makes use of IRIS Explorer in script mode. Maps are first generated by visioneers and then converted into scripts which specify modules, their parameters and the connections between them using a language called Skm. These Skm scripts constitute the realisation of visualization pipeline fragments. The user therefore does not see the complex IRIS Explorer interface when running in script mode, and does not need to know about the processing that is going on behind the scenes.

4.6 Repositories

The main repositories provide a storage location for middleware objects.

The Core Techniques (CT) repository provides a store of pipeline fragments that implement visualization techniques such as generating an isosurface, contour plot, histogram and so on. These are categorised and stored in a hierarchy based on the E notation developed by Brodlie [5]. The techniques repository also provides a store for visualization techniques named for the user, such as 'John's Fish Stock Visualization'.

The Input Format Conversion (IFC) repository provides a store of pipeline fragments that allow SuperVise to transform raw data in the given format to the format used by the chosen core technique. Similarly the Output Format Conversion and Display (OFCD) repository provides a store of pipeline fragments to transform geometry so that it can be displayed on the chosen device.

The Node Capabilities repository (NC) will provide a store of grid node details such as installed visualization software, as well as details of other machines and their output capabilities. This repository will then be used to enable autonomous selection of visualization processing and display resources. At present the user must have knowledge of the available resources and select them manually.

5 Conclusions and Future Work

SuperVise decouples the visualization processing from the display of results. We have achieved this through the development and implementation of a new conceptual model aimed at describing the process of visualization as three well-defined tasks, namely Input Format specification, Core Technique specification and Output Requirements specification. SuperVise is currently deployed on a test

grid installation, but to gain experience of using Supervise in a more realistic environment we envisage deployment of the system in the future on a larger scale.

The current implementation of Supervise requires that resources be selected by the user. In future, we aim to include facilities for autonomous resource selection at runtime so that Supervise will be aware of changes in the user's environment, the availability of equipment, node load and so on.

The current system architecture is appropriate to support those visualization techniques that do not require extensive interaction on the part of the user, but in due course we will investigate how best to support interaction-oriented techniques as well.

References

1. Upson, C., Faulhaber, T., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., van Dam, A.: The Application Visualization System : A Computational Environment for Scientific Visualization. IEEE Computer Graphics and Applications **9** (1989) 30–42
2. Haber, R.B., McNabb, D.A.: Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. Visualization in Scientific Computing (1990) 74–93
3. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications **11** (1997) 115–128
4. von Laszewski, G.: Java Commodity Grid (CoG) Toolkit. Internet: `http://www.computingportals.org/CPdoc/CGT.doc` (2002)
5. Brodlie, K.: Visualization Techniques. In: Scientific Visualization - Techniques and Applications. Springer-Verlag (1992) 37–85

Metrics for Grid Applicability: A Distributed Elliptic Curve Platform Assessment

Paulo Trezentos[1]* and Arlindo L. Oliveira[2]

[1] UNIDE / ISCTE
Edif. ISCTE, Av. Forças Armadas
1600-082 Lisboa Paulo.Trezentos@iscte.pt
[2] INESC-ID / IST
R. Alves Redol 9
1000 Lisboa aml@inesc-id.pt

Abstract. The distribution of computational load among several nodes is an important step for problems requiring High Performance Computing (HPC). This phase is critical because bad decisions could cost time and money.
The emergence of heterogeneous networks with resources available for a wide group of users as provided by Grid Computing [3] creates the need for formal processes and metrics to evaluate if the application of the problem to the Grid is feasible.
In this article we introduce some auxiliary indicators for measuring the potential applicability of a parallel application to a Grid. Using the measures defined in the internal draft (GWD-I) produced by the Network Measurement WG [6] from the Global Grid Forum and RFC 2330 [7], the authors present some aggregate metrics that are useful in the characterization of the parallel applications that are well adapted to existing grids.
The defined auxiliary metrics were useful in the validation of a concrete application that factorizes numbers using the elliptic curves method (ECM) [5] over a testbed Grid. The results of the application of the metrics to this specific mathematical algorithm are presented.

1 Introduction

Not all parallel implementations are adequate to all grids. We usually face one of these two scenarios: having a parallel application developed and build a grid for running it or having an already existent grid and implementing a parallel solution suitable to that grid.

In the first case, the application is "fixed" and the grid should serve its purposes. This is generally the case of large problems where the necessary funds are put at the team's disposal.

* The author would like to thank ADETTI and CFC by their support to this work, and the means that were put at his disposal.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 864–871, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The second case applies to already existent grids where you can not change its network and hardware capabilities. We then have a "fixed" grid and our application should be developed specifically having in mind the grid specifications.

This work proposes the use of aggregated measurements that can help the algorithm designer in the process of determining if a given grid is suitable for a given problem. The auxiliary ratios hereby presented should not be used in a deterministic way as a "yes or no" oracle suggestion. They are useful to perform grid / application specification analysis and avoid a situation where the definition of requirements is based only in "common sense experience".

The use of combined metrics is not new (See, for example, [2]). Nevertheless, the authors believe that the requirements and specification analysis stage still lacks the help of appropriate tools.

2 Auxiliary Metrics

The auxiliary metrics that will be presented in this section measure the applicability of one distributed application to a Grid. We assume that we have a typical grid and application scenarios:

- **Grid** - a set of computing resources (both hardware and software) combined in units called nodes that are interconnected through a network infrastructure; if a client / server architecture is enforced by the application, some of the nodes can act as clients (processing back-ends) and some as servers (front-ends);
- **Application** - program that runs simultaneously in different nodes of the grid with a specific goal to be achieved in limited time. Some interaction between nodes is supposed to exist. In the rest of this document the analysis will assume that the application is composed of a server program that delivers data to be processed and by the clients that, after processing the data, reply to the server with the results and, optionally, send a request for more data.

2.1 Critical Factors

The first step consists in defining which critical factors should weight in an equation that ponders if the grid is suitable. Due to space requirements, we will not address in detail the process by which these variables are measured.

- **minimum bandwidth required** - what is the minimum bandwidth between nodes that the application requires;
- **maximum latency allowed** - what is the maximum latency / delay between nodes that the application supports;
- **data transmission frequency** - how often there is a need to transmit data between nodes;
- **amount of data to transmit** - which amount of data is transmitted in each communication.

Having these variables in mind, some combined measurements were produced.

2.2 Frequency of Interruptions for Communication (FIC)

This auxiliary metric analyzes the average number of suspensions in application processing due to the need to establish a communication to exchange data.

We will use *minute* as unit but that should be flexible enough to allow to use subunits when more appropriate.

$$\frac{\text{Node processing capacity (FLOPS)} * 60}{\text{Total application floating point instructions}} \quad (1)$$

By **node processing capacity** (FLOPS) we represent the number of floating point instructions that one node of the grid is capable of processing by second, on the average.

Total application floating point instructions means the average number of floating point instructions that a client node application processes between receiving the data for processing and delivering the results to the server. Since the number of floating point instructions might depend of the data received, an average value should be used.

The result of the equation will be the number of *interruptions per minute.*

Moreover, this auxiliary measurement reflects the granularity of the compromise between transmitting more data in each transfer and having a longer processing period or the reverse.

Note that this granularity can, for some type of applications, be flexible but, for the rest, is fixed, and one can not increase or decrease the amount of data to be transmitted.

2.3 Total Time of Data Transmission (TTDT)

The **Total Time of Data Transmission (TTDT)** is the absolute time period that it takes the server to transmit data to the client.

This value is given by the following equation:

$$\frac{\text{Total data to be transmitted (KB)} * 8}{\text{Bandwidth of the connection (Kbps)}} \quad (2)$$

Total data to be transmitted (KB) quantifies - in KBytes - the quantity of data that will be transfered from the server to the client for processing purposes.

By **Bandwidth of the connection** (Kbps) is meant the bandwidth availability as defined in NMWG GWD-I [6]. The result will be in seconds, and multiplication by 8 is required so that both measures work with the same unit (KB).

We assume that:

- in what concerns this ratio, latency is negligible compared to the total time of the transmission; for latency control purposes we will introduce later another ratio (MCO);

- communication is only established between server and client, not taking into account the client-to-client connections and the client-to-server results transmission; this last factor can commonly be neglected since the transmission of results is usually not bandwidth intensive;
- the initial transfer of the client program into the grid node is not pondered since it happens only once.

If any of the previous points is not applicable, then the equation should take into account the changed conditions.

2.4 Maximum Communication Overhead (MCO)

Maximum Communication Overhead is a limit that should be established indicating the maximum overhead that we are willing to accept for the communication.

For instance, if we draw the limit of MCO to 5% this will mean that we are ready to accept that our system will spend at most 5 minutes in communication for each 100 minutes of processing. We can check if an implementation does not cross this limit by using the following equation:

$$= \frac{\text{TTDT (seconds)}}{\frac{1}{\text{FIC}} * 60} * 100 = \frac{\text{TTDT (seconds)} * FIC}{60} * 100 \tag{3}$$

2.5 Maximum Latency Overhead (MLO)

The **Maximum Latency Overhead** represents the ratio between the delay introduced by the network constraints and the absolute time of the overall transmission[1]. This indicator can help us establish a limit (%) for the maximum latency overhead that we are ready to accept.

The MLO can be defined as follow:

$$= \frac{\text{Latency/delay}}{\text{Overall time of data transfer}} * 100 \tag{4}$$

The latency / delay can be defined [6] in two ways: one-way delay and round-trip delay. Moreover, since it is defined by network and protocol characteristics, we can use the RFC 1242 [1] definition In this case, it is defined as the average time that one bit takes to be transfered from the original computer's port into the port of the destination computer.

2.6 "Goodness Frontier" for the Application

Using the previous combined metrics we can think in a Cartesian axis that uses two of the indicators as axis and Maximum Communication Overhead (MCO) as the limit to the use of the application over the grid.

[1] This time is different from the previously introduced TTDT since it already includes latency and other delays reflecting the overall time of the communication.

The line that represents the frontier can be defined using the following coefficients:

- **A** - maximum limit to TTDT above which it does not pay to use the grid, even for small values of FIC. To define its value we can take into account the latency through the MLO ratio;
- **M** - slope of the *goodness* straight line

The slope can be found through:

$$m = \frac{A}{FIC} - \frac{TTDT}{FIC} \tag{5}$$

Thus:

$$TTTD = A - M * FIC \tag{6}$$

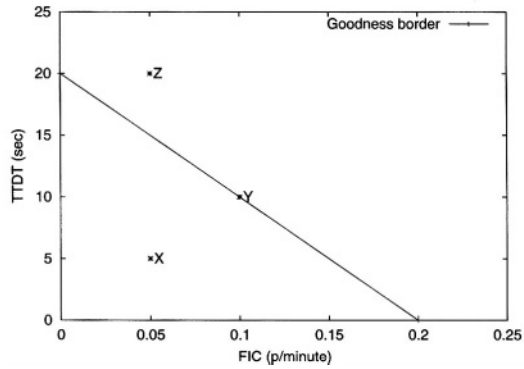


Fig. 1. "Goodness" application ratio

The straight line will then reflect our defined MCO. Figure 1 exemplifies the use of this ratio. The value of FIC is represented in the **X-axis**. In this axis, points closer to the origin represent less interruptions, and, therefore, a better chance of applicability to the grid.

In the **Y-axis** we represent the Total Time of Data Transmission (TTDT). In the same way, points closer to the origin are points better adapted to use the grid.

The ***goodness frontier*** is drawn using $A = 20$ and $M = 100$. This frontier (represented as a dotted straight line) is specially interesting when we have to take a decision about employing or not the grid in our application, because it considers simultaneously variables that characterize both grid and the application.

We can illustrate the use of these measures with some examples:

1. The X point, located at (0.05,5) is below the goodness frontier. As such, we can say that it represents an application that is clearly appropriate to the grid. Its location (0.05,5) comes from an FIC of 0.05, reflecting an interruption in each 20 minutes, and an TTDT of 5 seconds, which corresponds to a good working point.
2. The Y point (0.1,10) is near the threshold of the *goodness frontier.* With an FIC of 0.1 - one interruption in each 10 minutes - and an 10 seconds TTDT, the decision about its feasibility can not be clearly taken.
3. The Z point (0.05,20) has the same FIC as the X point but a much higher TTDT. This fact puts it above the goodness frontier and therefore in the non-applicability zone.

We emphasize that the cited combined metrics take into account several factors, among which the processors performance. One grid with exactly the same specifications of another grid but with slower CPU nodes will have a better result in what concerns the applicability of a given application since it will take more time to process the same set of instructions. Thus, before exercising these indicators we should verify if the CPU computational power is suitable to our problem.

3 ECM over the Grid (DISTECM)

An Elliptic Curve over a field K is the set of solutions that respect:

$$Y^2 = X^3 + AX^2 + BX + C \tag{7}$$

The points in the curve form an Abelian Group on which all operations are well defined. The negative of any point $P1 = (x_1, y_1)$ is defined by $-P1 = (x_1, -y_1)$. The Elliptic Curve Method (ECM) was applied to factorization problem proposed by Hendrik Lenstra [5].

ECM has become more popular than Pollards P-1 since it works well even if $P - 1$ is not smooth. Assume that n>1, gcd(n,6)=1 and that n is not a prime power with power >1. The ECM algorithm for factoring n consists in [5]:

1. Draw a, x, $y \in Z/\mathrm{n}Z$
2. Identify P=(x:y:1) $\in N$ and select an integer K=K(a,b)
3. Calculate K· P
4. If it fails, the d divisor of n it was found. If not, we return to point 1 again and start over.

The platform devised for Distributed ECM uses the Globus Middleware[2]. Globus provided grid services such as Information (GRIS / GIIS), Security (GSI), Resource Management (GRAM) and Data Management (GASS). These services can be used to launch processes, retrieve output files and allow single sign-on on the grid.

[2] http://www.globus.org

At application level, we implemented a client-server program where the communication is performed using standard sockets and not PVM or MPI (MPICH). The server role is to distribute curves among the several clients until one of them finds factors of n.

The platform was tested for the factorization of the Partition Number P(10341). The idea of using partition numbers for factorization was launched by the RSA company[3]. The number P(10341) has 109 digits and was rapidly factored using the distributed platform.

The factorization took 17 hours and 7 minutes with 8 clients over a small SMP cluster with 4 nodes. Table 1 shows some statistics of the load distribution observed.

Table 1. Summary of curves processed by each server

Node	# curves	Aver. Time	Std. Dev.
Gandalf	91	1330 "	3,93
Hobbit	177	682 "	2,95
Aragorn	133	904 "	245,57
Isildur	117	1028 "	194,2

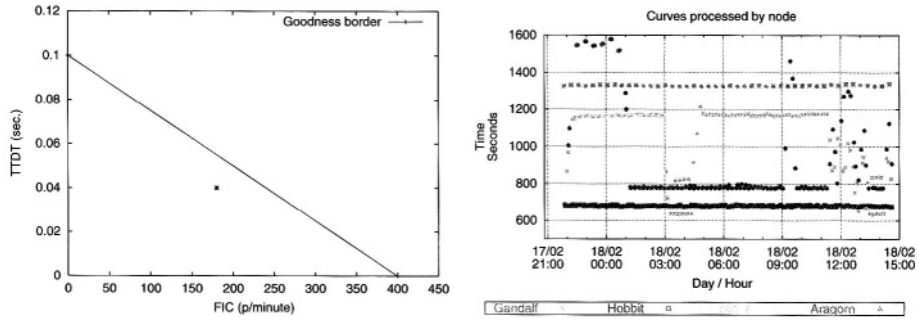


Fig. 2. Result of "Goodness frontier" metric application

4 DISTECM Assessment

In this scenario our grid is based on a Beowulf Cluster [8]. A Beowulf is a cluster of PCs built with commodity components and off-the-shelve hardware solutions [4]. PCs are usually connected to a l00Mbps FastEthernet network or Gigabit. This corresponds to a 100Mbps connection, with an average latency estimated at 6ms. Experimental evaluation of the (average) CPU speed gives a value of 52.4 MFLOPS. The "goodness frontier" for this grid environment is represented in the left of figure 2.

[3] http://www.rsasecurity.com/rsalabs/challenges/factoring/index.html

In this graphic, the "Goodness frontier" limits FIC to 400 interruptions per minute which his acceptable for a FastEthernet network. As the marked point is below the "goodness frontier" we conclude that the DISTECM would be suitable for this kind of grid. The experimental results have indeed shown an average CPU occupation of 97.7%. This indicates a well balanced load distribution, and reasonably small idle times, which validates experimentally the applicability of the metrics proposed.

The rightmost graphic in figure 2 illustrates the distribution of tasks over SMP nodes. Together with the high occupation ratio observed, it shows that, despite the differences in CPU speed, the available processing speed is used up to its full capacity.

5 Conclusions

Mathematical algorithms like ECM can be suitable or not for distributed processing over a grid. The metrics hereby presented can be of help in the process of defining a formal approach to understand what applications are appropriate for use in grids.

The distributed ECM (DISTECM) platform has proved to be a good approach for factoring numbers under 100 digits. An NFS distributed platform based on CWI code that broke an RSA 512 bits key was not presented in this article but was also developed by the authors and is more efficient in the factorization of numbers above 100 digits.

Future work will concentrate in enhancements of these metrics in the Global grid Forum (GGF) scope and application of the same to other fields of problems.

References

1. S. Bradner. RFC 1242: Benchmarking terminology for network interconnection devices, July 1991. Status: INFORMATIONAL.
2. T. Ferrari and F. Giacomini. Network monitoring for grid performance optimization. *Computer Communications, submitted for publication* 2002.
3. I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1999.
4. P.T. José Guimarães: Spino: A distributed architecture for massive text storage. *ICEIS* 1:244–248, 2001
5. H.W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics* 126:649–673, 1987.
6. B. Lowekamp, B. Tierney, L. Cotrell, R. Hughes-Jones, T. Kielmann, and M. Swany. A hierarchy of network measurements for grid applications and services - draft. Technical report, Global Grid Forum - Network Measurements Working Group. July 2002.
7. V. Paxson, G. Almes, J. Mahdavi and M. Mathis. RFC 2330: Framework for IP performance metrics. May 1998. Status: INFORMATIONAL.
8. J. Radajewski and D. Eadline. Beowulf howto. Technical report, Linux Documentation Project. Nov. 1998.

Execution and Migration Management of HLA-Based Interactive Simulations on the Grid*

Katarzyna Zając[1], Marian Bubak[1,2], Maciej Malawski[1], and Peter Sloot[3]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30,30-059 Kraków, Poland
{kzajac, bubak, malawski}@uci.agh.edu.pl
[2] Academic Computer Centre – CYFRONET, Nawojki 11,30-950 Krakow, Poland
[3] Faculty of Sciences, Section of Computational Science, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
sloot@science.uva.nl
phone: (+48 12) 617 39 64, (+48 12) 633 80 54

Abstract. This paper presents the design of a system that supports execution of a HLA distributed interactive simulations in an unreliable Grid environment. The design of the architecture is based on the OGSA concept that allows for modularity and compatibility with Grid Services already being developed. First of all, we focus on the part of the system that is responsible for migration of a HLA-connected component or components of the distributed application in the Grid environment. We present a runtime support library for easily plugging HLA simulations into the Grid Services Framework. We also present the impact of execution management (namely migration) on overall system performance.

1 Introduction

The need for an environment supporting the development of complex distributed interactive simulations is an important issue. Simulation can often replace experiments that are too expensive or dangerous to conduct. For example, a simulation can help a surgeon perform operations through virtual reality training. Such an application is under development within the CrossGrid project [4].

The High Level Architecture (HLA) for Modeling and Simulation was developed as an IEEE standard to facilitate interoperability among simulations and to promote reuse of simulation components. With HLA, a large-scale distributed simulation can be constructed using even a huge number of geographically distributed computing nodes. However, the HLA standard does not provide any mechanisms for managing the execution of a simulation according to the dynamically changing conditions of computing resources. HLA assumes that the distributed environment is a predictable, fault-tolerant system.

The Grid is a promising approach for managing resources according to the demands of interactive applications; however their unreliable behavior is an important aspect which must be taken into account. The first interesting approach

* This research is partly funded by the European Commission IST-2001-32243 Project "CrossGrid".

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 872–879, 2004.
© Springer-Verlag Berlin Heidelberg 2004

to combining the possibilities of HLA and Grid technology was presented in [3] and bases on GT2.

We propose a system that ports existing well-known solutions for interactive applications into the Grid. To build the required fault tolerant Grid environment for HLA applications, we propose to use the OGSA Grid services approach. Preliminary concepts were presented in [16,17]. In this paper, we focus on the part of the system which is responsible for migration of a HLA-connected component or components of the distributed application in the Grid environment.

2 HLA as a Support for Distributed Interactive Simulations

HLA is explicitly designed as a support for interactive distributed simulations. Therefore, HLA provides various services needed for that specific purpose, such as time management useful for time-driven or event-driven interactive simulations. It also takes care of data distribution management and allows all application components to see the entire application data space in an efficient way.

On the other hand, the HLA standard do not provide automatic setup of HLA distributed applications. In order to run the whole application a user has to log into chosen machines and manually start each of the components (federates). In HLA there is no mechanism for migrating federates according to the dynamic changes of host loads or failures. HLA does not provide dynamic discovery.In the HLA-RTI implementation [13] this issue is resolved by means of a naming service that resides in a coordination process called RTIexec. Normally, for a newly joined federate, the endpoint of RTIexec is found either by using multicast or by specifying of the endpoint explicitly in a configuration file. The use of multicast is limited in a WAN and therefore constitutes a significant problem for Grids.

3 Overview of Grid Environment for HLA Simulation

While designing management system for HLA-based simulations one must consider not only host specific issues, but also performance of the network connection. Therefore, choosing an appropriate resource for migration is a nontrivial task. Fig.1 explains the control flow during steering of the HLA simulation execution (namely its migration). **Monitoring Tool** monitors the execution of HLA federates within host and network and makes decisions about migration. If there is a need to migrate, the **Monitoring Service** informs the **Migration Scheduler**(step 1), which then asks the **Information Service** for available resources(step 2). Basing on that information, the **Migration Scheduler** makes a decision on where to migrate the HLA federate and asks the **Monitoring service** to check if the chosen resource satisfies requirements of the application (step 3). If yes, it asks **Local Flow Setup Service** to set up all the necessary local services on that resource (step 4). When this is done, the **Migration Scheduler** asks the **HLA–Speaking service** to save the state (step 5) and transfers data

to the new site. Then, it restarts the new **HLA–Speaking Service** with the available checkpoint file (step 6).

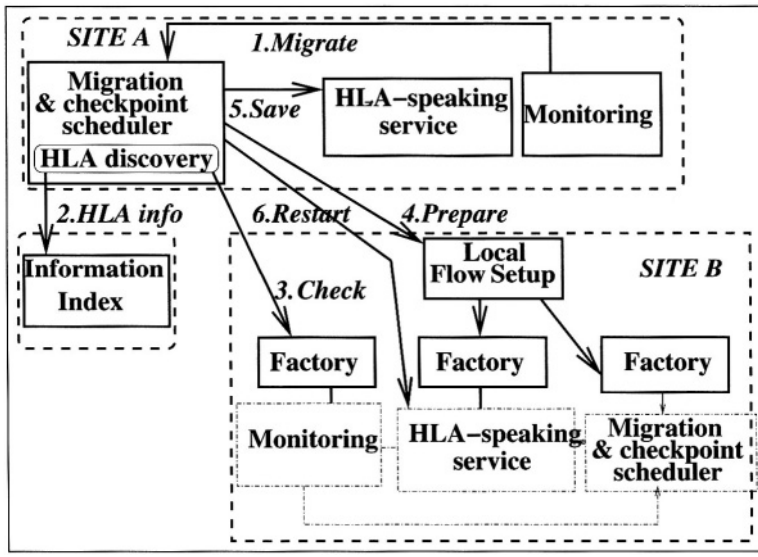


Fig. 1. Migration of s part of sn HLA application within the Problem Solving Environment

Migration Scheduler. An interactive application should use Grid Services in accordance with varying policies of remote sites. There is a need for a component that would schedule invocation of HLA Services in a way that will not harm other users, but still allow for execution of all the components of the interactive distributed application in near-real time. The Migration Scheduler carries the logic of the migration system, it interacts with **Monitoring Tool** to get information about resources available. For optimal decision it needs not only the status of computing node, but also the network connection between new node and nodes, where the other federates of the federation are executed. This information is taken from the monitoring system while making the decision (step 3 in the fig. 1).

Information Services. For our purposes we need an HLA information service that helps the system to find out HLA Services on the Grid. The OGSA team is working on a scalable architecture of the Information Index [14] based on the former MDS (Metacomputing Directory Service) tree architecture. This component should be scalable and we should supply it with information about HLA installation on the Grid.We are investigating to build the network of information services based on scale free network structure [15].

Code and checkpoint transfer. Off-line data transfer is an issue already resolved in GT2. The GridFTP protocol [1] and OGSA support a Reliable File Transfer service [14].

HLA–Speaking Service is created by means of a service factory and the actual federate(s) code is loaded into the service, which, in turn prepares the environment for the federate?s connection to the HLA bus. The HLA-Speaking PortType of the service provides the following operations for the **Migration scheduler Service** to support migration: *start,* which loads the federate code and the required libraries, *requestRun* which starts the federate execution, *requestRunWithRestore* which starts the federate from the checkpoint file and *requestMigrationSave,* which saves the federate and RTI state to a file. The federate code should provide callback functions to be orchestrated by the **Migration Scheduler Service**. HLA-components need to understand migration scheduler requests. In the next section we propose the Migrator Library API to allow the HLA application developers to easily plug their simulation code into the migration services framework.

There are several HLA-specific issues [3] concerning the design of migration support for HLA components of interactive applications. The most important one is how to define means of suspending the federate without generating HLA-specific errors (i.e. by resigning from federation). There are also some other issues like time and data management consistency. In the next section we discuss details of the proposed solution.

Monitoring Tool. The Migration Scheduler requires local information about the resource used by the local federate within an HLA-speaking service. A lot of effort has gone into building monitoring systems. We are considering the use of the monitoring system developed by the CrossGrid project [4].

4 Migration Library – An Interface between HLA and Grid Services

Migration of one of the federates without affecting the behavior of other federates is a nontrivial task. The HLA specification hides the actual location of data and messages, therefore using the HLA Management Object Model (MOM) [12] for obtaining information about a federate?s internal state cannot guarantee that this state is up-to-date.

A different approach is to write a new implementation of the HLA standard. This, however, would be huge task and would also force the developer to use a selected implementation of the library.

We decided to use the HLA specification facility to save and restore the HLA internal

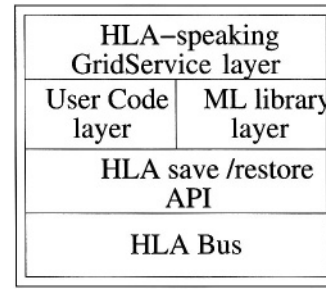


Fig. 2. ML in the context.

state. It is not completely transparent to the developer, since it requires using the HLA API and it does not provide for saving user-specific data. However, it assures state consistency between all federates with relatively little overhead.

We have developed the Migration Library (ML) that adds support for saving user-specific data, simplifies the HLA API and acts as an interface between user code and general HLA-speaking service functionality. The design of ML is based on the communication scheme which strongly depends on HLA specification features. In Fig.2 shows the ML library in the context. It is placed on the same level as user code, between Grid Service layer and HLA save/restore API layer.

5 Functionality Description of ML

The functionality of the ML is encapsulated within the Migrator class. The routines are responsible for startup of the simulation, its connection to RTI bus, checking if the internal and external save and restore requests came and saving user values. Below, we describe the functionality in more detail.

Startup functions. These routines are responsible for starting up the simulation code. Function `static int Migrator: :main (Migrator*,...)` starts the program and should be used instead of the standard `main()` function. It provides a pointer to a Migrator object which is necessary to call other functions of the library. Function `Migrator::register_start()` registers the main simulation loop. This function should be invoked inside the `Migrator::main()` routine.

RTI Connection functions. These routines are responsible for connecting to RTI bus. Routine `Migrator::set_rtiamb_count()` informs the migration engine about the number of federations in which a federate is participating. `Migrator::add_RTIambassador()` provides a reference of those federations to RTIambassadors.

Checking external migration or restore requests. Those functions check if the external request from **Migrator Service** has arrived.
`Migrator:: check_restore_state()` checks if the federate is being restored from the checkpoint file and if so, performs all the necessary steps to restore the internal state of the RTI.
`Migrator:: check_migration_state()` checks if a save request has been received and performs all the necessary HLA functions to save the internal state of RTI as well as the user space state defined by the routines specified below.

Checking internal RTI saving/restoring requests. This type of routines should be used while implementing RTI callbacks of the Federate Ambassador. They pass the information provided by RTI to the Migration Library.
`Migrator::save_on()` notifies ML that RTI has sent a request message.
`Migrator::save_off()` notifies ML that RTI has finished saving internal data.
`Migrator::restore_on()` notifies ML that RTI has sent a restore request message. `Migrator::restore_off()` notifies ML that RTI has finished restoring internal data.

Saving and restoring user-specific values. These routines provide an API for saving and restoring user-specific values. `Migrator::init_save()` opens an output checkpoint file. `Migrator:: save()` saves the object to the checkpoint file. The object must have a defined stream operator. `Migrator::exit_save()` closes the output checkpoint file. `Migrator::init_restore()` opens the input checkpoint file. `Migrator::restore()` fills an object with data saved in the checkpoint file. The object must have a defined stream operator. `Migrator::exit_restore()` closes the input checkpoint file.

6 Performance Results

The current implementation consists of a basic **Migration Component,** the functionality of which is going to be included into the **Migration Scheduler Service.** The main objective of our tests was to measure migration time in different circumstances as well as the impact of OGSA and HLA RTI solutions on the overall system performance. The tests were performed on two different testbeds, on a LAN at the Institute of Computer Science AGH, consisting of 8 PCs with Pentium III 800 MHz CPUs and 500MB of RAM, connected by 100 Mbps Fast Ethernet and on a WAN testbed, based on the DutchGrid DAS2 [5] and the Krakow ACC Cyfronet infrastructure, consisting of 8 PCs with Pentium III 1+ GHz CPUs and 1GB of RAM, connected by a 10 Gbps network (inside DAS2) and a 155Mbps network between DAS2 and Cyfronet.

No of federates	DAS2		LAN	
	Avr	Err	Avr	Err
2	31	2.7	27.4	0.5
3	32	1.1	27.4	0.5
4	34	2.8	32.4	0.5
5	37	2.7	35.5	0.5
6	40.5	0.5	39.8	0.5

Table 1. Publisher migration time as seen by subscribed federates (in seconds).

	DAS2		LAN	
	Avr	Err	Avr	Err
Service creation	10.8	0.1	11.9	0.5
Code loading	8.1	0.1	10.0	0.1
Save/Restore invocation	6.2	0.1	7.2	0.2
Transfer time	2.9	0.1	3.0	0.1

Table 2. Time of invoking **Migration Service** functions (in seconds).

The HLA federation consisted of N federates (one publisher and N-1 subscribers). We have measured the impact of publisher migration on subscribers. The scenario of migration is briefly described in Section 3 and it is as follows: The **Migration Component** creates the **HLA–Speaking Service** on the remote site which it wants to migrate to; then it transfers user code and loads it into the **HLA–Speaking Service** (via JNI). These steps are invisible to the federates and the time of their execution is shown in Tab.2. Then the **Migration Component** requests the local **HLA-speaking Service** to save its state

and transfers the checkpoint file to the remote resource. Finally, it invokes the *restore* operation on the remote service. In Tab.1 we measured the average time of publisher migration as seen by subscribers on both LAN and Grid testbeds. The time depends linearly on the number of federates participating in the federation and includes saving the internal state of RTI (via the HLA standard API), saving user data (via the migration library API), invocation of restore request, and GridFTP transfer of checkpoint files of user data. RTI internal state files reside on the RTIexec machine and therefore do not need to be transferred. These tests were performed for a relatively small user data checkpoint file (10KB), so the most significant factor is the impact of the implementation of HLA RTI methods responsible for saving the internal state of RTI, including the synchronization of federates. In Tab.2 we show the average time of functionality of the **Migration Service**. Here, the most significant factor is the performance of the OGSI implementation, since network latencies are relatively small. As can be seen, invocation of the restore operation on the **HLA-speaking** service is less then one quarter of the whole migration time visible by other federates and it is independent of their number. The design of the system allows for benefiting from OGSA technology without significant overhead on actual migration. Moreover, the time required for setting up the new HLA service is relatively short and the service can be set up before the actual migration starts.

7 Summary and Future Work

This paper presents the overall concept of a Grid management system for HLA-based applications. The goal is to provide the fault tolerant environment for executing HLA applications on the Grid, which itself is very unreliable. We also presented preliminary results of the implementation of a part of the system responsible for federate migration. Thanks to the Grid services technology, all modules of the system can be developed independently of each other. The results show that the OGSA-based migration system does not introduce significant overhead in comparison with actual HLA performance. Currently we are working on scheduling strategies for interactive applications to help the **Migration Service** in deciding about migration.

Acknowledgments. The authors wish to thank Alfredo Tirado-Ramos, Zhiming Zhao and Dick van Albada for valuable discussions about OGSA and HLA, and Piotr Nowakowski for additional remarks.

References

1. Allcock W., Bester J., Bresnahan J., Chervenak A., Liming L., Tuecke S.: GridFTP: Protocol Extensions to FTP for the Grid, http://www.globus.org/

2. Bubak M., Malawski M., Zajac K.: Towards the CrossGrid Architecture. In: D. Kranzlmueller, P. Kacsuk, J. Dongarra, J. Volker (Eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 9th Eurpean PVM/MPI Users' Group Meeting, Linz, Austria, September/October 2002, LNCS 2474, pp. 16-24.
3. Cai W., Turner S.J., Zhao H.: "A Load Management System for Running HLA-based Distributed Simulations over the Grid" in : Proceedings of the 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications 2002 IEEE.
http://csdl.computer.org/comp/proceedings/ds-rt/2002/1853/00/1853toc.htm
4. CrossGrid http://www.eu-crossgrid.org
5. Distributed Ascii Supercomputer http://www.cs.vu.nl/das2/
6. DataGrid http://www.eu-datagrid.org/
7. Foster, I: "What is the Grid? A three checkpoints list". GridToday Daily News And Information For The Global Grid Community July 22, 2002: VOL. 1 NO. 6
8. Foster I, Kesselman C., Nick J., Tuecke S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
9. Gannon D., Ananthakrishnan R., Krishnan S., Govindaraju M., Ramakrishnan L., Slominski A.: Grid Web Services and Application Factories
http://www.extreme.indiana.edu/xgws/afw/appFactory.pdf
10. Grid Ant http://www-unix.globus.org/cog/projects/gridant/
11. GSFL http://www-unix.globus.org/cog/projects/workflow/
12. HLA specification, http://www.sisostds.org/stdsdev/hla/
13. HLA-RTI https://www.dmso.mil/public/transition/hla/rti/
14. OGSA http://www-unix.globus.org/ogsa/
15. Reka A., Barabasi A.L.: "Statistical mechanics of complex networks" in: Review of Modern Physics, vol 74, January 2002.
http://citeseer.nj.nec.com/499527.html
16. Tirado-Ramos A., Zajac K., Zhao Z., Sloot P., van Albada D., Bubak M.: "Experimental Grid Access for Dynamic Discovery and Data Transfer in Distributed Interactive Simulation Systems" to be published in : P.M.A. Sloot, David Abramson, A. Bogdanov, Jack J. Dongarra, Albert Zomaya, Yuriy Gorbachev (eds.): Proceedings of International Conference on Computational Science - ICCS 2003, vol. I, no. 2657, Lecture Notes in Computer Science, Springer, 2003, pp. 284-292.
17. Zajac, K., Bubak, M., Malawski, M., Sloot, P., Tirado-Ramos, A., Zhao, Z., A Proposal of Services For Managing Interactive Grid Applications, in: M. Bubak at al: M., Noga, M., Turala, M. (Eds.), Proceedings of Cracow'02 Grid Workshop, December 11-14, 2002, Krakow, Poland, ACC Cyfronet UMM, 2003, pp. 155-163.

Asymptotically Optimal Algorithm for Laplace Task Graphs on Heterogeneous Platforms

Olivier Beaumont, Pierre Ramet, and Jean Roman

LaBRI UMR CNRS 5800, ENSEIRB, University Bordeaux 1
INRIA Futurs ScAlApplix Project **
{obeaumon|ramet|roman}@labri.fr

Abstract. In this paper, we focus on the scheduling of Laplace task graph on a general platform where both communication links and processing units are heterogeneous. In this context, it is known that deriving optimal algorithm, in the sense of makespan minimization, is NP-Complete, and several inapproximation results have been proved. Nevertheless, we provide an asymtotically optimal algorithm in this general context. Moreover, we expect that this methodolgy can be extended to more general task graphs, especially for nested loops where the innermost loop is parallel.

1 Introduction

Parallel distributed memory machines improve performances and memory capacity but their use induces an overhead due to communications. To obtain programs that perform and scale well, this overhead must be hidden. The choice of a good data distribution can be used to decrease the number and the size of communications, and to obtain large grain computations. Moreover, depending of the dependences within the code, asynchronous communications can be used to overlap communications with computations. In the context of heterogeneous computing, an additional difficulty comes from load balancing.

Pipelined algorithms have been widely studied in homogeneous architectures (see [1] and references therein). We focus on the problem of finding good data distributions in order to minimize the execution time when the size of the task graph is large, on heterogeneous platforms such as clusters of SMP nodes or heterogeneous networks of workstations.

Scheduling general task graphs is known to be NP-Complete in the context of heterogeneous resources (see for instance [2] and [3]). In this paper we consider more specifically the scheduling of Laplace task graph on a platform where both communication links and processing units are heterogeneous. In the general framework of Laplace task graph, each element is updated using its left and upper neighbors. This graph can model for instance the main loop of ADI (Alternative Direction Implicit) algorithm [4].

** http://www.labri.fr/scalapplix

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 880–887, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The remainder of the paper is organized as follows. In Section 2, we present some related work, dealing with close problems (pipelined communications on heterogeneous networks) or using close techniques in order to prove asymptotical optimality. In Section 3, the model of the platform is presented, and the asymptotically optimal algorithm is described and proved. At last, we give some conclusions and future work in Section 4.

2 Related Work

2.1 Pipelined Communications on Heterogeneous Networks

In [1], the authors study the minimization of the latency of a message through a network of store-and-forward stages. The platform is characterized by the number n of pipeline stages, and stage i is characterized by g_i, the fixed per-fragment overhead (its latency) and G_i, the time transfer per-byte (the inverse of the bandwidth). It is worth of interest to note that processing can be included in this model, by adding stages where the $g_i = 0$ and G_i represents the time necessary to process one element.

The main contribution of [1] is to provide a polynomial algorithm when the size and the number of fragments is fixed, i.e. re-fragmentation and reassembling is not possible between the stages. In this context, a linear program (in rational numbers) provides the optimal solution for the size of the fragments.

When the size and the number of fragments is not fixed, i.e. re-fragmentation and reassembling is possible between the stages, the solution is only known under strong assumptions and with limited pipeline sizes.

2.2 Asymptotically Optimal Packet Routing

The packet routing problem is the following: let $G = (V, E)$ be a non-oriented graph modeling the target architectural platform, and consider a set of same-size packets to be routed through the network. Each packet is characterized by a source node (where it initially resides) and a destination node (where it must be located in the end). For each pair of nodes (v_k, v_l) in G, let n_{kl} be the number of packets to be routed from v_k to v_l. Bertsimas and Gamarnik [5] introduce a scheduling algorithm which is asymptotically optimal when $n = \sum n_{kl} \to +\infty$.

The algorithm consists in three main phases. First, a linear program is solved, where, so to speak, temporal constraints have been removed: it is never written than a packet must have reached a node before leaving it. This solution of this linear program provides a lower bound T_{OPT} for the completion time of the packet routing problem. Then, a periodic algorithm is derived from the solution of the linear program. It roughly consists in $\sqrt{T_{\text{OPT}}}$ periods of duration $\sqrt{T_{\text{OPT}}}$, and during each period, the packets are routed using the rounded behavior given by the linear program. At last, the (few) remaining packets are routed sequentially. The makespan of this schedule is $T_{\text{OPT}} + O(\sqrt{T_{\text{OPT}}})$, hence the asymptotic optimality.

2.3 Divisible Load Scheduling

The divisible load model has been widely studied in the last several years, and popularized by the landmark book written by Bharadwaj, Ghose, Mani and Robertazzi [6]. A divisible job is a job that can be arbitrarily split in a linear fashion among any number of processors. The applications of the divisible load model encompass a large spectrum of scientific problems, including among others Kalman filtering, image processing, video and multimedia broadcasting, database searching, and the processing of large distributed files (see [6] for more examples).

In [7], the target computing platform is a heterogeneous master/worker platform, with p worker processors labeled $P_1, P_2, \ldots, P_p$. The master P_0 sends out chunks to workers over a network. The master processor can distribute the chunks to the workers in a single round, (also called installment in [6]), so that there will be a single communication between the master and each worker.

For large workloads, the single round approach is not efficient, because of the idle time incurred by the last processors to receive their chunks. In order to minimize the makespan, i.e. the total execution time, it is proposed in [7] to send the chunks to the workers in multiple rounds: the communications will be shorter (less latency) and pipelined, and the workers will be able to compute the current chunk while receiving data for the next one. As in the case of packet routing, an asymptotically optimal algorithm is derived, using $\sqrt{T_{\text{OPT}}}$ periods of duration $\sqrt{T_{\text{OPT}}}$, whose makespan is of order $T_{\text{OPT}} + O(\sqrt{T_{\text{OPT}}})$, hence the asymptotic optimality.

3 Asymptotically Optimal Algorithm

3.1 Modeling of the Platform

In this section, we consider the scheduling of Laplace task graphs on heterogeneous platforms. The platforms we consider are very general, since both communication links and processing units are heterogeneous. The only assumption we make is the possibility of overlapping communications with computations, i.e. the existence of a "front end". More precisely, we model the platform as a graph $G = (V, E, w_i, c_{i,j})$. The nodes V_i ($|V| = m$) represent the processors, labeled with w_i, their processing power, i.e. the necessary time to process one update. The edges $(V_i, V_j) \in E$ represent the communication links, labeled with $c_{i,j}$, i.e. the necessary time to transfer one element from V_i to V_j.

In order to derive an asymptotically optimal algorithm, we need a cyclic ordering of the nodes of the platform. Since it is known to be NP-Complete to find an edge disjoint Hamiltonian path in G, and that such a path may well not exist, we rely on a very simple heuristic. This heuristic may lead to a bad ordering, inducing a lot of communications. Nevertheless, as we will prove it in Section 3.5, it is always possible to overlap communications with computations, so that this bad ordering has little impact on the overall performance of the algorithm.

The heuristic we rely on is the following. Let $T = (V, E')$ be a spanning tree of G, rooted in V_0. We build the path using depth first order of the tree T. This path is denoted by $P = (V_0, V_{i_1}, \ldots, V_{i_K}, V_0)$. For sake of simplicity, we consider a reordering of the indexes of the nodes, so that the nodes first appear in the path P in lexicographic order, namely, if $f(i)$ denotes the smallest index k such that $f(i) = i_k$, then $\forall i,\ f(i) < f(i+1)$.

In the algorithm we propose in Section 3.5, each node V_i sends a message to node $V_{i+1[m]}$ inside each period. Thus, we need to evaluate the cost of such an exchange on the network topology we have defined. Since communications are handled along the edges of the tree according to depth first order, each edge of the tree is used exactly twice (one for forward and one for backward communications). Moreover, we made the assumption of one-port model for the nodes, so that one processor cannot be involved in more than one communication at the same time.

Thus, for node V_i, the overall communication time is given by $2 \sum_{V_j \in \mathcal{V}(V_i)} c_{i,j}$, where $\mathcal{V}(V_i)$ denotes the neighborhood of V_i in the tree. Thus, the overall communication time for the whole exchange procedure can be bounded by $2\ m\ c_{\max}$, where $c_{\max} = \max c_{i,j}$. This bound is very pessimistic in general (though it may be reached in the case of homogeneous communications and a star shaped network), but as already mentioned, is enough in order to prove the asymptotic optimality of the algorithm we propose in Section 3.5.

3.2 Lower Bound for Processing Time

Let us consider a $n \times n$ Laplace task graph. The algorithm consists in updating all n^2 elements. To establish a lower bound for the complexity, we neglect for a while both communication costs and precedence constraints in order to concentrate on load balancing issues. Let us denote by n_i the number of elements updated in the optimal solution by node V_i, and let us denote by T_{OPT} the optimal time under those simplifying assumptions. The time required for V_i to update all n_i elements is given by $w_i n_i$, where w_i denotes the time to update one element.

Therefore, the following linear program provides a lower bound for overall processing time:

$$\textbf{Minimize } T_{\text{OPT}},$$
$$\textbf{subject to}$$
$$\begin{cases} \forall i, \quad n_i w_i \leq T_{\text{OPT}} \text{ (lower bound for processing time on } V_i) \\ \quad \sum n_i = n^2 \quad \text{(all the elements are processed).} \end{cases}$$

The solution of this linear program is not trivial in integer numbers, but solving it in rational number also provides a (lower) lower bound for processing time and, in this context, the solution T_{OPT} given below is a lower bound for the processing time of the $n \times n$ Laplace task graph.

$$\forall i, \quad n_i = \frac{n^2}{w_i \sum \frac{1}{w_j}}, \text{ so that } T_{\text{OPT}}(n) = \frac{n^2}{\sum \frac{1}{w_j}}$$

3.3 Data Distribution

Let us consider again a $n \times n$ Laplace task graph. We allocate rows to processors in a periodic way. The Laplace task graph is decomposed into roughly $\sqrt{n}$ periods of roughly $\sqrt{n}$ rows, and in each period, each processor receives approximately the amount of rows that corresponds to perfect load balancing.

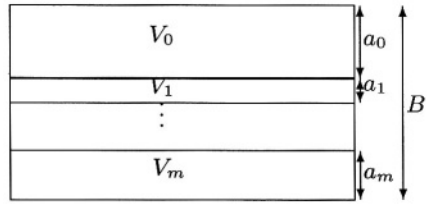


Fig. 1. Data distribution inside a period of size B

More precisely, let us denote by $a_i = \lfloor \frac{\sqrt{n}}{w_i \sum \frac{1}{w_j}} \rfloor$ the number of rows allocated to V_i inside one period, so that the size B of the period is given by $B = \sum a_i$. Inside one period, the rows are allocated to the processors accordingly to their ordering in the path defined in Section 3.1, i.e. a_0 rows to processor V_0, and then a_1 to V_1 an so on, so that a period is as depicted in Figure 1.

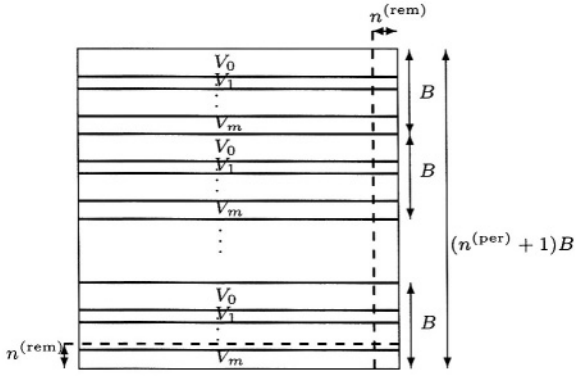


Fig. 2. Data distribution of the filled matrix

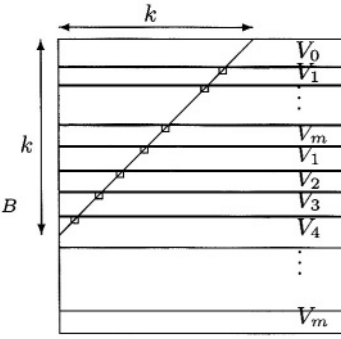


Fig. 3. Step k of the algorithm

Let us denote by $n = n^{(\text{per})}B + n^{(\text{rem})}$ the euclidean division of n by B, where $1 \leq n^{(\text{rem})} < B$. The data distribution of the whole matrix is depicted in Figure 2, and consists of $n^{(\text{per})}+1$ periods of size B. The last $B - n^{(\text{rem})}$ rows and columns are filled with zeros. Clearly, the processing time of the filled matrix is larger than the processing time of the original matrix, so that asymptotical optimal results obtained on the filled matrix can be extended to the original matrix.

3.4 Sketch of the Algorithm

The algorithm we propose consists in updating the elements along a moving diagonal. At step k, the elements (i, j) where $i + j < k$ have already been updated. Step k consists in updating the elements on the diagonal $i + j = k$, as depicted in Figure 3.

In Figure 3, the boxed elements of the k^{th} diagonal are those for which a communication is required from above processor before the update. For all other diagonal elements, both left and upper neighbors are located on the same processor, so that the update can take place without any communication.

This very simple algorithm is in fact asymptotically optimal, as proved in Section 3.5. The proof is based on the following two remarks. First, if n is large, the bands allocated to each processor are large (of order $\sqrt{n}$), and since only one communication is required for the computation of the whole band, then communications can be overlapped with computations. Moreover, since load balancing into each period of size B is almost perfect and (provided k is sufficiently large) the number of full periods is large with respect to the last (partial) period, then the processors receive an amount of work which is roughly proportional to their processing capabilities.

3.5 Proof of the Asymptotic Optimality

Overlapping condition. As already mentioned, we expect that, for sufficiently large values of n and k, communications can be overlapped with computations. Let us consider precisely the k^{th} step of the algorithm, and let $k = pB+q$ denote the euclidean division of k by B. Thus, p denotes the number of full periods, and $q < B$ denotes the number of remaining rows. Let us evaluate the overall volume of communication involved in step k. Inside each period, each processor sends one element to its below neighbor processor. In Section 3.1, we have seen that the cost of this pipelined exchange takes less than $2\ m\ c_{\max}$ time units, so that the overall volume of communication involved at step k is bounded by $2\ (p+1)\ m\ c_{\max}$.

Let us now evaluate the amount of computation that does not require any communication. Inside each period, processor V_i can process $a_i - 1$ updates without communication (see Figure 3). Thus, at step k, processor V_i can process at least $p(a_i - 1)$ elements in $p(a_i - 1)w_i$ time without the need of any communication.

In order to overlap communications with computations, the processing time that does not require communications must be smaller than the overall communication time, on any processor. This condition can be written as

$$\forall i, \quad p(a_i - 1)w_i \geq 2(p+1)m\, c_{\max}\ .$$

Since $\frac{k}{B} - 1 < p \leq \frac{k}{B}$, and $a_i \geq \frac{\sqrt{n}}{w_i \sum \frac{1}{w_j}}$, above condition holds true as soon as

$$(\frac{k}{B} - 1)(\frac{\sqrt{n}}{w_i \sum \frac{1}{w_j}})w_i \geq 2(\frac{k}{B} + 1)\, c_{\max} \iff \frac{k}{B} \geq \frac{\sqrt{n} + 2\, c_{\max} \sum \frac{1}{w_j}}{\sqrt{n} - 2\, c_{\max} \sum \frac{1}{w_j}}\ .$$

Therefore, as soon as $n \geq (2\, c_{\max} \sum \frac{1}{w_j})^2$, the above condition becomes $k \geq B$. Therefore, if n is sufficiently large, communication can be overlapped with computation at any step, except for the B first steps. Thus, we need to consider separately the first B steps, where communications cannot a priori be overlapped, and the next pB steps.

Case where $k \leq B$. Let us first consider the case where $k \leq B$. Since B is relatively small (of order $\sqrt{n}$), we can rely of a very rough bound. We bound the necessary time to process step k by the time necessary to process all the updates on the slowest processor, if we had to pay one communication (on the slowest link) between each pair of rows. In this case, the overall processing time for step k would be $(w_{\max} + c_{\max})k$, where $w_{\max} = \max w_i$. Therefore, if we denote by $T(k)$ the overall processing time of step k, then $T(k) \leq (w_{\max} + c_{\max})k$.

Case where $k > B$. Let us now consider the general case where $k > B$. In this case, the time to process all the rows distributed on V_i is bounded by $(p+1)(a_i - 1)w_i + (p+1)w_i$, i.e. the time to process all the updates that do not require any communication plus the time to process the updates of remaining tasks, once the communications have been done (what is possible because of the overlapping condition). Thus, if we denote by $T(V_i, k)$ the time to process all the tasks distributed on V_i, then

$$\begin{aligned} T(V_i, k) &\leq (p+1)a_i w_i \leq (p+1)(\tfrac{\sqrt{n}}{w_i \sum \frac{1}{w_j}} + 1)w_i \\ &\leq (\tfrac{k}{B} + 2)(\tfrac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max})\ . \end{aligned}$$

The last bound holds true for any V_i, so that $T(k) \leq (\frac{k}{B} + 2)(\frac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max})$.

Asymptotic optimality. We have evaluated the processing time $T(k)$ of any step of the algorithm. By a trivial symmetry argument, the overall processing time $T_{\text{ALG}}(n)$ for the Laplace task graph $T(k)$ is bounded by $2\sum_1^{(n^{(\text{per})}+1)B} T(k)$.

$$\begin{aligned} \text{Thus, } T_{\text{ALG}}(n) &\leq \textstyle\sum_{B+1}^{(n^{(\text{per})}+1)B} (\frac{k}{B} + 2)(\frac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max})) + 2(\sum_1^B (w_{\max} + c_{\max})k \\ &\leq (n^{(\text{per})} + 1)((n^{(\text{per})} + 1)B + 1)(\tfrac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max}) + \\ &\quad 2(n^{(\text{per})} + 1)(\tfrac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max}) + (w_{\max} + c_{\max})B(B+1)\ . \end{aligned}$$

Moreover, since $\sqrt{n} - m \leq B \leq \sqrt{n}$, then, as soon as $\sqrt{n} \geq m$, then

$$\begin{aligned} &T_{\text{ALG}}(n) \leq \tfrac{n^2}{\sum \frac{1}{w_j}} + 2n\sqrt{n}(\tfrac{1}{\sum \frac{1}{w_j}} + 2m + w_{\max}) + O(n) \\ &\text{and therefore, } \tfrac{T_{\text{ALG}}(n)}{T_{\text{OPT}}(n)} \leq 1 + \tfrac{1}{\sqrt{n}}(2 + (2m + w_{\max}) \textstyle\sum \frac{1}{w_j}) + O(\frac{1}{n})\ . \end{aligned}$$

Thus, when n is large, $\frac{T_{\text{ALG}}(n)}{T_{\text{OPT}}(n)}$ becomes arbitrarily close to 1, what achieves the proof of the asymptotical optimality of the algorithm we propose.

4 Conclusion

In this paper, we have shown that it is possible to derive an asymptotically optimal algorithm for Laplace task graph, when the size of the task graph becomes arbitrarily large. The algorithm is based on an heterogeneous block data distribution, that ensures a good load balancing all along the processing. Since the size of the task graph is large, it is possible to make use of large grain distribution and thus, communications can be fully overlapped with computations.

This study may be a good testbed in a more general context. Indeed, Laplace task graph is an example of DO – DO-PAR loops, if the matrix is updated along its diagonals. This context is much more general, and the transformation of a nested loop into a DO – DO-PAR loop may be obtained, for instance, with the algorithm of Allen and Kennedy [8].

The case when the outermost loop is parallel has already been addressed in [9], since it corresponds to the distribution of independent tasks, when data is supposed to be initially located on a single processor and master slave tasking is used. Our study may lead to comparable results for inner DO-PAR loops.

Finally, the algorithm we propose shows that asymptotical optimality is much easier to obtain than classical makespan minimization, especially in the context of heterogeneous computing (see also [7] and [5]). Moreover, considering very large task graphs makes sense in the context of heterogeneous (grid) computing, since the startup time on such a platform may be large, thus restricting its use to very large regular applications.

References

1. Wang, R.Y., Krishnamurthy, A., Martin, R.P., Anderson, T.E., Culler, D.E.: Modeling and optimizing communication pipelines. In: ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM (1998)
2. Hochbaum, D.S.: Approximation Algorithms for NP-Hard Problems. PWS (1997)
3. Shirazi, B.A., Hurson, A.R., Kavi, K.M.: Scheduling and load balancing in parallel and distributed systems. IEEE Computer Science Press (1995)
4. Lowenthal, D.K.: Accurately selecting block size at run time in pipelined parallel programs. International Journal of Parallel Programming **28** (2000) 245–274
5. Bertsimas, D., Gamarnik, D.: Asymptotically optimal algorithm for job shop scheduling and packet routing. Journal of Algorithms **33** (1999) 296–318
6. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society Press (1996)
7. Beaumont, O., Legrand, A., Robert, Y.: Optimal algorithms for scheduling divisible workloads on heterogeneous systems. Technical Report 2002-36, LIP, ENS Lyon, France (2002)
8. Allen, J.R., Kennedy, K.: Automatic loop interchange. In: Proceedings of the SIGPLAN Symposium on Compiler Construction. Volume 19-6., Montreal, Canada (1984) 233–246
9. Banino, C., Beaumont, O., Legrand, A., Robert, Y.: Scheduling strategies for master-slave tasking on heterogeneous processor grids. In: PARA'02. LNCS 2367, Springer Verlag (2002) 423–432

Dynamic Tasks Assignment for Real Heterogeneous Clusters

Marta Beltrán[1], Antonio Guzmán[1], and Jose L. Bosque[1,2]

[1] ESCET, Universidad Rey Juan Carlos, 28933 Móstoles, Madrid, Spain
{mbeltran, aguzman, jbosque}@escet.urjc.es
[2] Universidad San Pablo CEU, Madrid, Spain

Abstract. Heterogeneous clusters are becoming very popular for providing flexible, scalable and cost-effective solutions for high performance computing. An efficient resources utilization can significantly enhance cluster performance, specially in heterogeneous environments. So, load balancing is critical for achieving high performance in this systems. In this paper a dynamic, global, distributed, general purpose and automatically tuning itself off load balancing scheme is proposed for heterogeneous clusters. A new workload index is developed to take into account nodes heterogeneity. To verify and validate our approach, a complete set of experiments on a real heterogeneous cluster is presented too.

1 Introduction

Cluster performance depends mainly on the nodes computational power and the network features. These resources utilization can be improved by assigning each processor a workload proportional to its processing capabilities [1]. In homogeneous systems the problem of load balancing has been extensively studied but heterogeneous clusters are an attractive alternative to the traditional multiprocessor and multicomputer environments due to their good price-performance ratio and scalability [2,3]. Most of load balancing schemes which have been proposed last years cannot be used in these new clusters. Therefore load balancing for heterogeneous clusters is an area of increasing interest. It is necessary to provide efficient, scalable, low overheading and general-purpose strategies capable of handling heterogeneity.

In this paper a new dynamic load balancing scheme for heterogeneous clusters is presented. It is a global, distributed, simple and general-purpose strategy. It can automatically tun itself off, thus, it disappears in global under-loaded or over-loaded situations. The load balancing is implemented as a non-preemptive tasks assignment, by remote execution. Hence processes are transferred from one node to another only when they are started and have no allocated resources. The main contribution of this paper is to define a new load index which considers cluster nodes computing power to handle heterogeneity.

The rest of this paper is organized as follows. Section 2 discusses related work on load balancing for heterogeneous clusters. Section 3 presents our approach for dynamic task assignment and in Sect. 4 the experimental results obtained with this implementation on a real heterogeneous cluster are summarized. And finally, Sect. 5, with some conclusions and future work.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 888–895, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Background

While many techniques for load balancing in homogeneous clusters have been proposed, load balancing for heterogeneous clusters is a relatively new subject of research, but some work has been reported on this issue.

Static load balancing techniques for data assignment are presented in [4,5]. On the other hand dynamic tasks assignment methods have been proposed in [6,7,8]. [6] and [7] are agent-based techniques. These are flexible and configurable approaches but the amount of resources needed for agents implementation is considerably large. In [6], a synchronous remote invocation metode is implemented to perform non-preemptive task allocation. Besides, this work analyses the effect of the load index in the load balancing system performance for a heterogeneous cluster. This analysis is based on the Kunz's work, [9]. On the other hand, [7] proposes a load management framework based on process migration using mobile agents. [8] presents a very different cooperative approach to the load balancing problem, considering it as a game in which each cluster node is a player and must minimize its jobs execution time.

3 Algorithm Design

Dynamic tasks assignment schemes can be decomposed in four policies [1]. Decisions made in the design phase have a huge influence on the load balancing performance and combining different approaches for these policies yields a large amount of dynamic load balancing techniques.

3.1 Load Measurement

Dynamic tasks assignment is based on information about nodes computational power and workload. This information must be quantified by a load index, measured periodically, which should be a good estimate of a node computing capabilities. The choice of a load index has a huge impact on load balancing efficiency [9].

Load Index: One important advantage of the proposed load index is the simplicity of its measurement. More sophisticated workload descriptions typically increase the system overhead [10] and decrease load balancing performance. The number of tasks in the run queue has been used as a workload descriptor in previous works [9,11]. But this index has a limited performance when the incoming tasks are no CPU-intensive.

Our index is based on a new concept, the CPU assignment. In this paper the CPU assignment (A) is defined as the percentage of CPU time that would be available to a new incoming task in a cluster node. If there are N CPU-intensive running tasks in a node, they share the processor in a balanced way, with Round Robin scheduling. The CPU assignment to a new task would be $\frac{1}{N+1}$. But no CPU-intensive tasks could be waiting for I/O operations or memory paging. Thus, CPU assignment to a new arriving job would be larger: all the available CPU.

Therefore, the load index computation is based on three parameters: one static parameter, the node computational power (P) and two dynamic parameters, the number of

tasks in the run queue (N) and the percentage of CPU utilization accumulated for the N tasks in the run queue (U) and is defined as:

$$A = \begin{cases} 1/(N+1) \text{ if } U \geq 1/N \\ 1 - U \qquad \text{otherwise} \end{cases} . \tag{1}$$

Due to cluster heterogeneity, there are different computational powers for cluster nodes. The absolute computational power for a node in a certain moment of time can be defined as:

$$P_{ABS} = P \cdot A \ . \tag{2}$$

But common references are needed to compare the load index on different nodes, so a normalization by the maximum computational power in the cluster is needed. If P_{MAX} is the maximum of all the P values, the load index definition is:

$$Index = \frac{P_{ABS}}{P_{MAX}} \ . \tag{3}$$

This index considers CPU-intensive and no intensive tasks, with no a priori knowledge about their requirements and without losing the desired simplicity.

Three-Status Scheme: The cluster nodes state is determined by their load index values. And the node state determines its capability for handling new arriving jobs. A three-state scheme has been defined [12]:

- *Receiver state:* High load index value, thus, under-loaded nodes. They can start new local tasks and accept new remote tasks execution.
- *Neutral state:* Medium load index value, not very well defined behavior. These nodes can accept new local tasks but they are not allowed to begin load balancing operations or to accept remote executions because they could be near receiver or sender states.
- *Sender state:* Low load index value, over-loaded nodes. They cannot accept new remote tasks. And for new local tasks they must begin load balancing operations.

The node current state determines completely its load balancing decisions. State changes are caused by load index variations through certain thresholds. Two algorithm parameters related to this three-status scheme are the neutral state width and the neutral middle point, which give the state change thresholds situation. Neutral width will be discussed in Sect. 4.

3.2 Information Exchange Rule

For making load balancing decisions it is necessary to exchange state information between the cluster nodes. Our approach is a global algorithm, so all the cluster nodes maintain updated information about the global system state. A on-state-change driven policy has been implemented. Processors broadcast their workload information when they suffer a state change. Load balancing operations can only take place between receiver and sender nodes. Hence, only state changes to or from receiver state are significant enough to communicate and thus, the number of messages is significantly reduced.

Each node maintains a state-queue with the information received from other nodes. Only a receiver-queue is needed, because only receiver nodes can accept remote executions.

3.3 Initiation Rule

It determines when to begin a new load balancing operation. A sender-initiated policy have been selected ([13]): only sender nodes can submit tasks to other cluster nodes. The load balancing decision is completely local. A receiver or neutral node can accept new local tasks. But a sender node must begin a load balancing operation.

This load balancing scheme is a conservative approach because neutral nodes are not allowed to take part in load balancing operations. The variation of the neutral state width provides a mechanism to control the maximum imbalance level allowed in the cluster. Load balancing operations are always performed between a receiver and a sender node. Thus, with a wide neutral state, load balancing operations would rarely occur. On the other hand, with a narrow neutral state, balancing operations would be frequently performed.

3.4 Load Balancing Operation

The load balancing operation is defined by three more rules, the *Location, Distribution* and *Selection* rules.

The Location rule finds the cluster node to balance the workload. In the proposed algorithm is solved with the state-queue. When there is a new arriving task on a sender node it begins a negotiation stage with the first receiver node in the queue which is the oldest one, so its state is the most stable. Our approach is automatically tuning itself off because when the state-queue is empty or when there are not sender nodes.

The Distribution and Selection rules are easily implemented because in every load balancing operation only the task which has just started to run is involved.

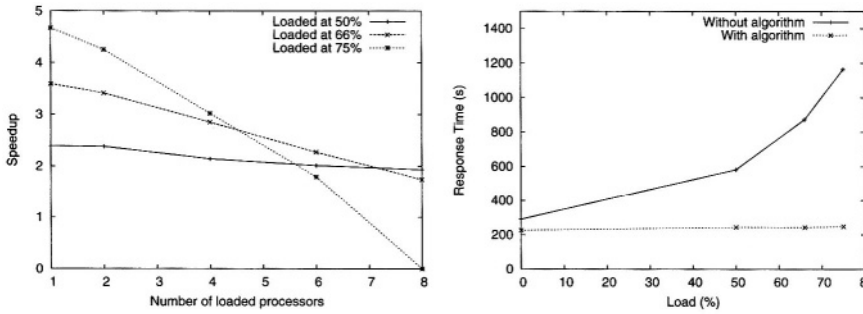
4 Experimental Results

The proposed algorithm has been implemented on a real heterogeneous cluster to verify its performance. In this implementation each cluster node runs concurrently three conventional processes which compose the load balancing application. They use the MPI/LAM 6.5.6 library [14] for communication issues.

The heterogeneous cluster is composed of 8 PC nodes, four of them based on 733 MHz Pentium III processors and the remaining four based on 550 MHz Pentium III processors, interconnected by a Fast Ethernet 100 Mbps network. A CPU-intensive workload (a matrix multiplication) was used in all the measurements. In each experiment, virtual users were simulated submitting new tasks with a fixed arrival frequency. Additionally, the cluster nodes could have external workload which the load balancing algorithm cannot manage.

Table 1. Response time with and without load balancing algorithm (without external workload).

Number of tasks	Time without alg. (s)	Time with alg. (s)	Asked op.
100	244	226	0
200	517	509	0
300	730	677	0
400	952	902	0



(a) Speedup evolution with the number of additionally loaded nodes

(b) Response time evolution with one loaded node

Fig. 1. Results with external load

4.1 Tests without External Workload

This first experiment compares cluster performance with and without the load balancing strategy. Cluster nodes have not additional external workload and the virtual users tasks arrival frequency is fixed in 2 seconds. An optimal balance for these tasks is statically calculated taking into account nodes computational power and its heterogeneity. Tasks are shared between nodes according to this optimal assignment.

Table 1 shows that response times are always better with load balancing algorithm than without it. Although there is not external load, the algorithm overhead is hidden by a better resources utilization. With dynamic tasks assignment, when nodes become senders, they do not accept new tasks. Hence, the processor does not execute all tasks simultaneously and each task has guaranteed a minimum CPU assignment. Finally, no load balancing operations are performed. The algorithm has automatically tuned itself off because the initial tasks assignment is perfectly balanced.

4.2 Tests with External Workload

This experiment makes a detailed comparison of the heterogeneous cluster performance with and without load balancing with different workload conditions. Besides virtual users tasks, there is an external workload on some cluster nodes. Users tasks are assigned

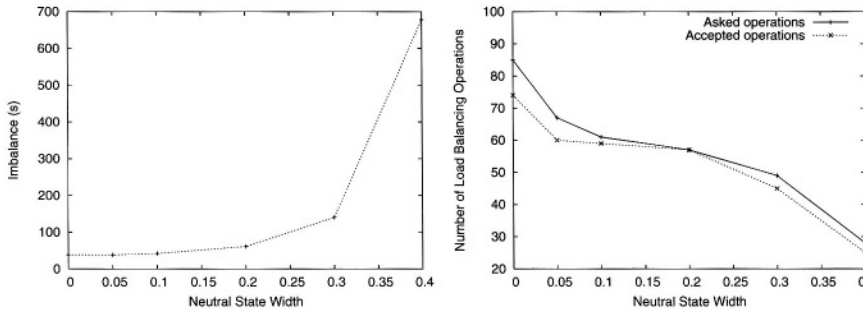
Table 2. Results with External Workload

Load	Loaded nodes	T without alg. (s)	T with alg. (s)	Speedup	Asked op.	Accepted op.	Dif.
50%	1	582	244	2,39	9	7	2
50%	2	582	245	2,38	12	11	1
50%	4	582	272	2,14	12	10	2
50%	6	582	289	2,01	16	13	3
50%	8	582	302	1,93	11	10	1
66 %	1	873	243	3,59	16	15	1
66 %	2	873	256	3,41	22	17	5
66 %	4	873	306	2,85	45	31	14
66 %	6	873	384	2,27	46	27	19
66 %	8	873	502	1,74	28	20	8
75 %	1	1164	249	4,67	19	18	1
75 %	2	1164	274	4,25	63	43	20
75 %	4	1164	385	3,02	75	48	27
75 %	6	1164	650	1,79	81	58	23
75 %	8	1164	∞	0,00	∞	0	∞

uniformly to the cluster nodes, hence the initial assignment is an unbalanced distribution. A variable percentage of cluster nodes has been additionally loaded with a variable percentage of external workload. The external load varies form 1 to 3 additional tasks (external load from 50% to 75%). All these results are shown in table 2 and Fig. 1(a).

As expected, the load balancing algorithm obtains a significant speedup in all the experiments. This speedup decreases when the number of nodes with external load increases because almost all processors are sender nodes and it is more difficult to assign tasks dynamically (Fig. 1(a)). And the speedup increases when the external load increases because an adequate tasks assignment is more critical for performance. On the other hand, table 2 shows that the number of asked and accepted load balancing operations increases with the loaded nodes increase, because there are more sender nodes. The same effect has the external load increase, when processors are more loaded the amount of asked operations increases too. Accepted operations increase in the same way but not so fast because it is more difficult to find a receiver node. Hence, the difference between asked and accepted operations increases with the cluster load. In high overloaded situations, the algorithm progressively tunes itself off, so the number of asked and accepted operations begins to decrease.

Another interesting experiment is progressively loading only one cluster node to understand load balancing advantages and to probe its efficiency (Fig. 1(b)). As can be seen, the external load in one cluster node has a tremendous impact on system response time, because all cluster nodes must wait for the loaded one. With the load balancing scheme, the loaded node can reassign its tasks to under-loaded nodes and system response time keeps constant indepently of the external load in the loaded node.



(a) System imbalance variation with the neutral state width

(b) Load balancing operations variation with the neutral state width

Fig. 2. Results for neutral width variation

4.3 Influence of the Neutral State Width

In this section the effect of neutral width variation is evaluated. Virtual users tasks are assigned uniformly and there is not external load. Response times for the slowest and fastest nodes are measured and system imbalance is calculated as the difference between these two times. These results verify the possibility of controlling the imbalance allowed in the system with the neutral state width (Fig. 2(a)). When this width is 0, there is not a neutral state and no imbalance is allowed. Hence, a lot of load balancing operations are performed to obtain a perfectly balanced assignment (Fig. 2(b)). On the other extreme, with a wide neutral state (for example 0.4), the number of load balancing operations considerably decreases but at expenses of response time and system balance.

The medium values for neutral width obtain a compromise between the number of load balancing operations and the response time. But the difference between asked and accepted operations is an important concern (the refused operations introduce an important overhead in the system). So the optimal neutral width is 0.2, which obtains the minimum response time matching asked and accepted load balancing operations.

5 Conclusions

In this paper a global, distributed, based on remote execution, with on-state-change driven information exchange and sender-initiated approach is presented to implement general purpose dynamic tasks assignment on heterogeneous clusters.

A new load index has been proposed based on CPU assignment and nodes computing power. A load index based on the first concept can manage CPU-intensive and no intensive tasks. And introducing nodes computing powers in the load index computation includes the cluster heterogeneity effects in this index.

Experimental results reported in this paper probe the benefits of this approach. Response times with load balancing are always better than without the algorithm, this

scheme does not perform load balancing operations when they are not necessary or efficient and values for the algorithm parameters are determined to optimally tun the load balancing system. Neutral state width, neutral state middle point and the load index update frequency have been discussed.

Future work will concentrate on evaluating different load metrics taking into account memory, I/O and network resources. Another important concern is to study process migration behavior in a scheme like the one presented here.

Acknowledgements. This work has been partially funded by the Spanish Commission for Science and Technology (grants CYCIT TIC2001-0175-03-03 and TIC2002-04486-C02-02).

References

1. Xu C. and Lau F. : *Load Balancing in Parallel Computers: Theory and Practice.* Kluwer Academic Publishers, Boston, (1997).
2. Pfister G. F. : *In search of clusters: The Ongoing Battle in Lowly Parallel Computing, 2nd ed.* Prentice Hall, Englewood Cliffs, NJ, (1995) edition, (1998). ISBN 0-13-899709-8, IBM.
3. Bell G. and Gray J. : What's next in high-performance computing? *Communications of the ACM,* 45(2):91–95, February (2002).
4. Snell Q., Judd G., and Clement M. : Load balancing in a heterogeneous supercomputing environment. In *Proc. 1998 International Conference on Parallel and Distributed Processing Techniques and Applications,* pages 951–957, (1998).
5. Bohn C. A. and Lamont G. B. : Asymmetric load balancing on a heterogeneous cluster of PCs. *Future Generation Computer Systems,* 18(3):389-00, January (2002).
6. Rajagopalan A. and Hariri S.: An agent based dynamic load balancing system. In *International Workshop on Autonomous Decentralized Systems,* pages 164–171. IEEE, (2000).
7. Obeloer W., Grewe C., and Pals H.: Load management with mobile agents. In *24th Euromicro Conference, vol. 2,* pages 1005–1012. IEEE, (1998).
8. Grosu D., Chronopoulos A., and Leung M. : Load balancing in distributed systems: an approach using cooperative games. In *16th International Parallel and Distributed Processing Symposium IPDPS '02,* pages 52–53. IEEE, April (2002).
9. Kunz T. : The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering,* 17(7):725–730, July (1991).
10. Eager D. L., Lazowska E. D., and Zahorjan J. : Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Eng.,* 12:662–675, May (1986).
11. Shen K., Yang T., and Chu L. : Cluster load balancing for fine-grain network services. *Proceedings of the International Parallel an Distributed Processing Symposium (IPDPS),* pages 51–58, (2002).
12. Willebeek-LeMair M. H. and Reeves A. P. : Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems,* 4(9):979–993, September (1993).
13. Eager D. L., Lazowska E. D., and Zahorjan J. :a comparison of receiver initiated and sender-initiated dynamic load sharing. Technical Report 85-04-01, University of Washington, (1985).
14. MPI Forum. : A message-passing interface standard. (1995).

Messages Scheduling for Data Redistribution between Clusters*

Johanne Cohen[1], Emmanuel Jeannot[2], and Nicolas Padoy[3]

[1] CNRS LORIA, Vandœuvre les Nancy, France jcohen@loria.fr
[2] LORIA, Université H. Poincaré, Vandœuvre les Nancy, France ejeannot@loria.fr
[3] École Normale Supérieure de Lyon, France npadoy@ens-lyon.fr

Abstract. In this paper we study the general problem of parallel data redistribution over a network. Given a set of communications between two parallel machines interconnected by a backbone, we wish to minimize the total time required for the completion of all communications assuming that communications can be preempted and that preemption comes with an extra cost. Our problem, called *k-Preemptive bipartite scheduling (KPBS)* is proven to be NP-Complete. Moreover we prove that approximating KPBS problem within a ratio number smaller that $\frac{4}{3}$ is impossible unless P = NP. In spite of this negative result, we study a lower bound on the cost of KPBS problem in terms of its parameters, and we propose an approximation algorithm with ratio 2 and fast heuristics.

1 Introduction

With the emergence of grid computing many scientific applications use code coupling technologies to achieve their computations where parts of the code are distributed among parallel resources interconnected by a network. Code coupling requires data to be redistributed from one parallel machine to another. For instance the NxM ORNL project [1] has for objective to specify a parallel data redistribution interface and CUMULVS [2] (which uses MxN) supports interactive and remote visualization of images generated by a parallel computer. In this paper we concentrate on the scheduling of the messages when a parallel data redistribution has to be realized on a network, called a backbone. Two parallel machines are involved in the redistribution : the one that holds the data and the one that will receive the data. If the parallel redistribution pattern involves a lot of data transfers, the backbone can become a bottleneck. Thus, in order to minimize the parallel data redistribution time and to avoid the overloading of the backbone it is required to schedule each data transfer.

In this paper, we revisit the problem of packet switching (in wavelength-division multiplexed (WDM) optical network [3,4,5,6,7] or in satellite-switched time division multiple access (SS/TDMA) [8,9,10]) in the context data redistribution.

* This work is partially supported by the ARC INRIA redGRID

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 896–906, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Data redistribution has mainly been studied in the context of high performance parallel computing [11,12,13]. In this paper we study a generalization of the parallel data redistribution. Indeed, contrary to some previous works that only deal with block-cyclic redistribution [14,13], here, no assumption is made on the redistribution pattern. Moreover, contrary to other works which assume that there is no bottleneck [11,12], we suppose that the ratio between the throughput of the backbone and the throughput of each of the n nodes of the parallel machines is k. Hence, no more than k communications can take place at the same time. We study the problem for all values of k. We focus on the case $k < n$ (the backbone is a bottleneck) whereas the case $k \geq n$ has been tackled in [11,12].

The contribution of this paper is the following. We prove that the problem of scheduling any parallel data redistribution pattern is NP-Complete for any value of k ($< n$) and that approximating our problem (called KPBS) within a factor smaller that $\frac{4}{3}$ is impossible unless P = NP. We exhibit a lower bound for the number of steps of the redistribution as well as a lower bound for the sum of the duration of each step and prove that both lower bounds are tight. Next, we propose two algorithms: a pseudo-polynomial approximation algorithm with ratio 2, and polynomial approximation algorithm with ratio 2. Finally, we study simple and fast heuristics that achieve a good average performance.

2 The Problem

2.1 Modelization of the Problem

We consider the following heterogeneous architecture made of two clusters of workstations $\mathcal{G}_1$ and $\mathcal{G}_2$ connected together by a backbone of throughput D. Let n_1 be the number of nodes of $\mathcal{G}_1$ and n_2 be the number of nodes of $\mathcal{G}_2$. All the nodes of the first cluster have a throughput d_1 and the nodes of the second have a throughput d_2.

Let us consider a parallel application that must execute the first part of its computation on $\mathcal{G}_1$ and the second part on $\mathcal{G}_2$. This is the case where an application is made of two parallel components such that each code is only available (for security/license reason) on one cluster.

During the execution of the application parallel data must be redistributed from the first cluster to the second one.

We assume that the communication pattern of the redistribution is computed by the application. This pattern is modeled by a *traffic matrix* $T = (t_{i,j})_{1 \leq i \leq n_1, 1 \leq j \leq n_2}$, where $t_{i,j}$ represents the amount of information that must be exchanged between node i of cluster $\mathcal{G}_1$ and node j of cluster $\mathcal{G}_2$.

For a given traffic pattern and for a particular architecture our goal is to minimize the total transmission time. In order to do this, we need to optimize the scheduling of the messages such that the available bandwidth is used without generating congestion. In this work, we do not rely completely on the network transport layer (i.e. TCP). Indeed, due to the control of the flow, TCP tends to use only a fraction of the total available bandwidth when congestion occurs.

Here, thanks to our knowledge of the underlying architecture, a large part of the congestion control is performed at the application level..

Let us consider the constraints relative to the communications. A transmitter (resp. receiver) cannot transmit (resp. receive) more than one message at a time (1-port model). However, we allow several messages between different transmitters and receivers to be transmitted simultaneously as long as the backbone is not saturated. A parallel transmission *step* is a communication phase in which there can be simultaneous transmissions between several transmitters and receivers. We denote by k the maximum number of simultaneous transmissions that can take place during one step. This number depends on the number of nodes (n_1 and n_2) of each cluster as well as on the bandwidth of the network card of each node (d_1 and d_2) and on the bandwidth of the backbone (D). We denote by d the speed of each communication.

For instance let us assume that $n_1 = 200$, $n_2 = 100$, $d_1 = 10\text{Mbit/s}$, $d_2 = 100\text{Mbit/s}$ and $D = 1\text{GBbit/s}$($D = 1000\text{Mbit/s}$). In that case, $k = 100$ because $\mathcal{G}_1$ can send 100 outgoing communications at 10 Mbit/s generating a total of 1 Gbit/s aggregated bandwidth (which is supported by the backbone) and each network card of $\mathcal{G}_2$ can receive the data at $d = 10$ Mbit/s.

A common approach to minimize the overall transmission time is to allow preemption, i.e. the possibility to interrupt the transmission of a message and complete it later. In practice, this involves a non-negligible cost, called *set-up delay* and denoted here by β, which is the time necessary to start a new *step.*

2.2 Formulation of the Problem

Let T be a traffic matrix, k be the maximum number of communications at each step, β be the startup delay and d be the speed of each communication.

We can normalize the problem by d and β as follows: (1) The traffic matrix *T,* can be replaced by the matrix $Q = (q_{i,j}) = (\frac{t_{i,j}}{d})_{1 \leq i \leq n_1, 1 \leq j \leq n_2}$ that represents the communication times for each messages . (2) The matrix Q can be replaced by the matrix $M = (m_{i,j}) = (\frac{q_{i,j}}{\beta})_{1 \leq i \leq n_1, 1 \leq j \leq n_2}$ that represents the fraction of setup delay required for sending each messages.

In the following we will always consider the normalized problem ($\beta = 1$).

The matrix M can be represented by a bipartite graph $G = (V_1, V_2, E)$ and a positive edge-weight function $w : E \to \mathbb{Q}$. Each node of cluster $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) is represented by a node of V_1 (resp. V_2). Hence, $|V_1| = n_1$ and $|V_2| = n_2$. The weight of an edge between node i and j is equal to $m_{i,j}$.

We use the 1-port model for the communication and at most k communications can occur during one step. Hence, a communication step is a weighted matching of G with at most k edges. The weights refer to preemption. We denote the matching corresponding to a communication step by a *valid weighted matching* (for the remainding, a valid weighted matching contains at most k edges).

We call this problem *k-Preemptive bipartite scheduling (KPBS),* formally defined as follows:

Given a weighted bipartite graph $G = (V_1, V_2, E, w)$ where $w : E \to \mathbb{Q}$, an integer[1] $k \geq 2$ find a collection $\{(M_1, W_1), (M_2, W_2), \ldots, (M_s, W_s)\}$ of valid weighted matchings such that:

1. Let w_i be the edge weight function of each matching M_i. It must respect the following inequalities: for any $e \in E$, $\sum_{i=1}^{s} w_i(e) \geq w(e)$. If $e \notin M_i$ then $w_i(e) = 0$.
2. For any $1 \leq i \leq s$, matching M_i has at most k edges ($|M_i| \leq k$) and its cost is equal to the rational number $W_i = \max_{e \in M_i} w_i(e)$.
3. $(\sum_{i=1}^{s} W_i) + s$ is minimized. In the normalized form of the problem, each step has a cost equal to W_i plus 1 for the setup cost.

In the remainder of this paper, note that for any solution S of *KPBS,* if the cost of S is $\alpha + s$, the number of steps is s and the useful transmission cost equals α. See Figure 1 for an example.

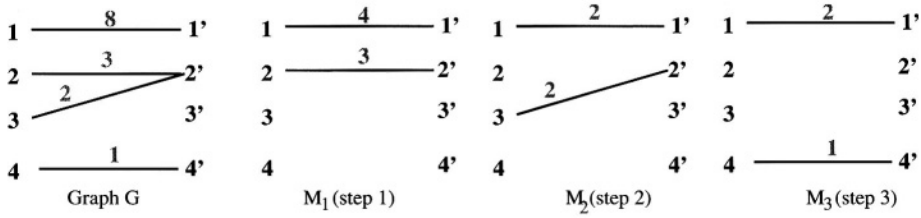


Fig. 1. An example for KPPS problem ($k = 2$). The cost of the solution is 8 + 3 = 11

3 Complexity Results

This problem has already been proven NP-complete for the particular case where $k \geq \min(n_1, n_2)$ [15,10]. We prove that it remains NP-complete for any fixed $k \geq 2$ (with a different reduction than in [15,10]).

Theorem 1 *Let $k \geq 2$ be a fixed integer. KPBS is NP-complete.*

Moreover, we improve the result in [12]. We prove that one cannot approximate the problem KPBS within a factor smaller than 4/3 if $P \neq$ NP.

Theorem 2 *If $P \neq NP$, there is no polynomial time approximation algorithm for the problem KPBS with an approximation ratio smaller than* 4/3.

Proof. Theorem 1 and 2 are proven in [16].

[1] The case $k = 1$ is not interesting: the backbone is saturated by one communication

4 Lower Bounds

Before giving a lower bound for the optimal solution, we give some graph notations. We define the weight $w(v)$ of a node v of G to be the sum of weights of all edges incident to vertex v. We denote the maximum of $w(v)$ over all vertices by *W(G)*. Let *P(G)* be the sum of the weights of all edges of graph *G*. We denote the maximum degree of the bipartite graph G by $\Delta(G)$, its number of edges by $m(G)$ and its number of vertices by $n(G)$.

Proposition 1 *Let $G = (V_1, V_2, E, w)$ be a weighted bipartite graph. Let k be an integer. The cost of the optimal solution for the instance $\langle G, k, \beta \rangle$ of KPBS is at least $\eta(G) = \eta_d(G) + \eta_s(G)$ where*

$$\eta_d(G) = max\left(W(G), \left\lceil \frac{P(G)}{k} \right\rceil\right) \quad and \quad \eta_s(G) = max\left(\Delta(G), \left\lceil \frac{m(G)}{k} \right\rceil\right)$$

Proof. $\eta_s(G)$ is a lower bound for the number of steps. The first term of the maximum accounts for the fact that two edges incident to the same node cannot appear in the same step and the second term for the fact that a step contains at most k edges. $\eta_d(G)$ is a lower bound for the useful transmission cost and is obtained similarly. The total cost is therefore minimized by $\eta_d(G) + \eta_s(G)$. □

Next, we study the quality of these lower bounds. The remainder of this section is to prove that there are polynomial time algorithms to optimize the number of steps (see Proposition 3) or the useful transmission cost (see Proposition 2).

Proposition 2 *Let G be a weighted bipartite multigraph. Then G can be decomposed such that the total transmission cost is $\eta_d(G)$.*

Proposition 3 *Let G be a weighted bipartite multigraph. Then G can be decomposed in $\eta_s(G)$ valid weighted matchings in polynomial time.*

Propositions 3 and 2 are equivalent. Indeed by setting all the weights to 1, Proposition 2 minimizes the number of steps because, in that case it is equal to the total transmission cost. On the contrary, by splitting all the edges into edges of weight 1, Proposition 3 gives a solution that minimizes the total transmission cost. We present a similar polynomial-time algorithm for Proposition 3 that will be used later.

The previous propositions can be seen as a consequence (see [17]) of a coloration theorem (given in [18] pages 132-133). Moreover, a proof of proposition 3 can be found in [9]. However, an other proof can be found in [16].

The decomposition is achieved in $\mathcal{O}(n(G)^{3/2} \times m(G)^3)$. The authors of article [8] provide a polynomial time algorithm that proves Proposition 2 for matrices, and shows that the number of steps is bounded by a polynomial in $n(G)$. We use it in section 5.

We separately studied η_s and η_d, what about η? There are quite simple graphs [17] (with all the edges having the same weight) such that η is not reached, and we can exhibit class of graphs (for instance graphs with edges having the same weight and with $k|m(G)$) for which it is.

5 Algorithms

The following algorithm approximates KPBS with a constant ratio.

Algorithm 1

Input: A weighted bipartite graph $G = (V_1, V_2, E, w)$ and an integer k a rational number α
Output: A set of valid weighted matchings.

1. Split every edge e of G into $\lceil \frac{w(e)}{\alpha} \rceil$ edges having each a weight equal to α, which leads to a multigraph H.
2. Find $\eta_s(H)$ valid weighted matchings whose union is H.
3. Every matching represents a communication step of length α.

In each matching of the solution the edges have the same weight, and in order to evaluate the solution, we decide that all steps have the same length α, where α is a constant that will be fixed to 1. The algorithm splits each edge in edges of weight α (it is an idea used in [11]) to make a multigraph H, then we find a solution such that the number of matchings is minimum (thanks to Proposition 3).

Its complexity is $\mathcal{O}\left(n(H)^{3/2} \times m(H)^3\right) = \mathcal{O}\left(n(G)^{3/2} \times m(G)^3 \times W(G)^3\right)$ and therefore pseudo-polynomial since the running time of Algorithm 1 depends linearly on the weights of G.

Proposition 4 *Let $cost(G, \alpha)$ be the cost of the solution given by Algorithm* 1. *$cost(G, 1) \leq 2 \times \eta(G)$. Therefore, Algorithm 1 is a* 2*-approximation algorithm.*

Let us first consider a particular class of graphs such that the parameter η_s is equal to 1. Let G be a graph such that $\eta_s(G) = 1$. By definition, we have $\Delta(G) = 1$ and $m(G) \leq k$. Thus, the scheduling is composed of 1 step and the cost of this scheduling corresponds to the lower bound. For the remainder of the proof, we only consider graphs G such that $\eta_s(G) \geq 2$.

Proof of Proposition 4: Assume first, that the weights of the edges of G are multiple of α. The definitions of η_s and η_d imply $\alpha \times \eta_s(H) \leq \eta_d(G) + \alpha$ and therefore:

$$cost(G, \alpha) = \alpha \times \eta_s(H) + \eta_s(H) \leq \eta_d(G) + \frac{1}{\alpha} \times \eta_d(G) + \alpha + 1 \qquad (1)$$

Since only graphs G such that $\eta_s(G) \geq 2$ are considered, we have $\eta(G) \geq \eta_d(G) + 2$. From equation 1, we get

$$cost(G, 1) \leq 2\eta_d(G) + 2 \leq 2\eta(G) - 2 \qquad (2)$$

Therefore, the approximation ratio is 2 with $\alpha = 1$.

When the weights are not multiple of α, they are rounded up to the first multiple of α, to make a graph $G^{'}$, then the previous algorithm is applied to $G^{'}$. So, from equation 1, we get

$$cost(G, \alpha) = cost(G', \alpha) \leq \eta_d(G') + \frac{1}{\alpha} \times \eta_d(G') + \alpha + 1 \tag{3}$$

We compare $\eta(G)$ to $\eta(G^{'})$. We have $\eta_s(G^{'}) = \eta_s(G)$, but $\eta_d(G^{'})$ differs:

$$\eta_d(G^{'}) = max\left(W(G^{'}), \left\lceil \frac{P(G^{'})}{k} \right\rceil\right) \tag{4}$$

$$\leq max\left(W(G) + (\alpha - 1)\Delta(G), \left\lceil \frac{P(G) + (\alpha - 1)m(G)}{k} \right\rceil\right)$$

$$\leq \eta_d(G) + (\alpha - 1) \times \eta_s(G) \tag{5}$$

Hence, from in-equations 3 and 5 we get:

$$cost(G, 1) \leq \eta_d(G')(1 + \frac{1}{\alpha}) + \alpha + 1 \leq 2\eta(G) + 2(1 - \eta_s(G)) \tag{6}$$

Since we only consider graphs G such that $\eta_s(G) \geq 2$, Algorithm 1 is a pseudo-polynomial time algorithm for KPBS with an approximation ratio 2. □

We use now this algorithm to describe a polynomial-time algorithm for KPBS with an approximation ratio 2. Given a graph *G,* we evaluate an expression depending on *P(G)* that represents roughly the average cost of a step (expressed in the number of set-up delays), then depending on the result of its comparison with the number of set-up delays, we branch on the previous algorithm or on another one.

Algorithm 2

Input: A bipartite graph G.
*Output:*A set of valid weighted matchings.

1. Calculate $\gamma = \frac{P(G)}{k \times (n(G)^2 + n(G) + 1)}$
2. If $\gamma \leq 1$, branch on Algorithm 1 with G and $\alpha = 1$ as input
3. Otherwise, branch on the algorithm which find the valid weighted matchings such that the useful transmission cost is minimized

When $\gamma \leq 1$ all the weights of G are bounded, therefore Algorithm 1 is polynomial. Indeed $W(G) \leq P(G) \leq k(n^2(G) + n(G) + 1)$. This yields to a complexity of $\mathcal{O}(kn^{15/2}(G) \times m^3(G))$

We need to determine the approximation ratio in the second case (when executing line 3). The paper [8] gives (with a matrix formulation) a polynomial

algorithm for optimizing the useful transmission cost with in the worst case a number of steps lower than $(n(G)^2 + n(G) + 1)$. For this algorithm, we have: (*cost(G)* being the cost of the solution given by Algorithm 2, when executing line 3).

$$cost(G) \leq \eta_d(G) + (n(G)^2 + n(G) + 1) \leq \eta_d(G) + \frac{P(G)}{k} \leq 2 \times \eta(G)$$

Therefore, we can deduce that:

Theorem 3 *There is a polynomial-time 2-approximation algorithm for KPBS.*

6 Heuristics

Algorithm 2 has a large complexity. In this section, we concentrate on fast heuristics that we have studied in practice.

Heuristic on weights

Input: A bipartite graph *G*.
Output: A set of valid weighted matchings.

1. Find a maximal matching.
2. Keep only the k (or less if there are less than k edges) edges whose weights are the biggest.
3. Set all the weights of the matching equal to the lowest one.
4. Subtract the matching from *G*.
5. Loop until there is no more edge left in *G*.

Here are two heuristics that appear to work well in practice (a heuristic on weighs and a heuristic on degrees). The heuristic on degrees is the same as the heuristic on weights except that line 2. is changed into "2. *Keep only the k (or less if there are less than k edges) edges with highest degrees*".

Complexity: We use the Hungarian method of complexity $\mathcal{O}(m(G) \times n(G)^{1/2})$ for finding a maximum cardinality matching in a bipartite graph. For both heuristics, at each step, at least one edge is removed from *G*. Therefore, the complexity of both heuristics is $\mathcal{O}(m(G)^2 \times n(G)^{1/2})$ which is better than the complexity of algorithm 2.

Experiments: We have tested each heuristic (with k fixed) on a sample of 100 000 random graphs (the number of edges, the edges, and finally the weights were chosen randomly with a uniform distribution). We made a difference between lightly and heavily weighted graphs. Small weights were taken between 1 and

20, whereas large weights were taken between 1 and 100 000. The result of a heuristic is calculated as the solution cost divided by the lower bound η. The plots show the average and the maximum calculated over the samples.

For these tests, the maximum is always below 2.5, even 1.8 for small weights, and the average is always below 2, and even 1.3 in case of large weights. Unfortunately, we didn't succeed into giving an approximation ratio for these two heuristics.

We explain the convex shape of the plots as follows:

- when $k = 1$ the two heuristics obtain the optimal solution which consists in one communication per steps;
- when k is greater than 2 and lower than a certain value (close to $n/2$), the quality of the solution degrades (compared to the lower bound); We believe that this is due to the fact that, at each step, the number of valid matchings increases;
- When k is greater than $n/2$ the quality of the solution tends to improve. At each stage of the two heuristics the choice of valid matchings decreases, therefore the heuristics are less likely to select bad valid matchings.

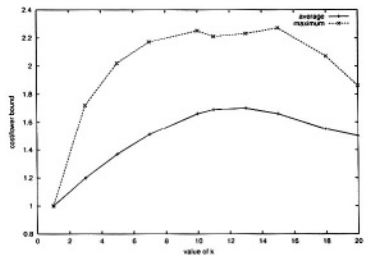


Fig. 2. Heuristic on weights. $n = 20$. Simulation on 100000 graphs with small weights per point.

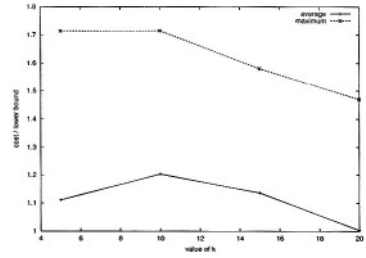


Fig. 3. Heuristic on edges. $n = 20$. Simulation on 100000 graphs with large weights per point.

7 Related Work

Up to our knowledge, there is no work on the KPBS problem in its generality ($n_1 \neq n_2$ and k can have any value, etc.).

This problem partially falls in a field originated by packet switching in communication systems for optical network called wavelength-division multiplexed (WDM) broadcast network [3,4,5,6,7].

The problem of minimizing the number of steps is studied in [9,4], and the problem of minimizing the total cost is studied in [5].

In [3] and in [6], the author consider a version of the KPBS problem where the number of receivers is equal to the number of messages that can be transmitted

at the same time ($k = n_2$) and where the set-up delay can be overlapped by the communication time (In [6] authors also assume that the size of all messages are the same). In that case, a list-scheduling algorithm is proven to be a 2-approximation algorithm in [3].

The case where the backbone is not a constraint ($k \geq \min(n_1, n_2)$) has been studied in [11,12] and it is known as the *preemptive bipartite scheduling (PBS)*. PBS was proven to be NP-complete in [15,10]. In [12], two different polynomial time 2-approximation algorithms for PBS have been proposed and in [11], an improvement of this result is given.

In the context of block cyclic redistribution several works exist [13,14]. In this case the communication pattern is not arbitrary and and the backbone is not a constraint.

8 Conclusions

In this paper we have formalized and studied the problem (called KPBS) of redistributing parallel data over a backbone. Our contribution is the following. We have shown that KPBS remains NP-Complete when k is constant. We have shown that approximating the KPBS problem within a ratio number smaller that $\frac{4}{3}$ is impossible unless P = NP. We have studied lower bounds related to KPBS. We have proposed a polynomial time approximation algorithm with ratio 2. We have studied two fast and simple heuristics that have good properties in practice.

Our future work is directed towards studying the problem when the throughput of the backbone varies dynamically, when the redistribution pattern is not completely known in advance or when the network cards on each cluster are not all identical. We would also like to perform real tests on real architectures in order to compute a realistic value of the startup time and to be able to build a library for parallel redistribution.

References

1. Labs, O.R.N.: Mxn. (`http://www.csm.ornl.gov/cca/mxn`)
2. Geist, G.A., Kohl, J.A., Papadopoulos, P.M.: CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. International Journal of High Performance Computing Applications **11** (1997) 224–236
3. Choi, H., Choi, H.A., Azizoglu, M.: Efficient Scheduling of Transmissions in Optical Broadcast Networks. IEEE/ACM Transaction on Networking **4** (1996) 913–920
4. Ganz, A., Gao, Y.: A Time-Wavelength Assignment Algorithm for WDM Star Network. In: IEEE INFOCOM'92. (1992) 2144–2150
5. Mishra, M., Sivalingam, K.: Scheduling in WDM Networks with Tunable Transmitter and Tunable Receiver Architecture. In: NetWorld+Interop Engineers Conference, Las Vegas, NJ (1999)
6. Pieris, G.R., G.H., S.: Scheduling Transmission in WDM Broadcast-and-Select Networks. IEEE/ACM Transaction on Networking **2** (1994)

7. Rouskas, N., Sivaraman, V.: On the Design of Optimal TDM Schedules for Broadcast WDM Networks with Arbitrary Transceiver Tuning Latencies. In: IEEE INFOCOM'96. (1996) 1217–1224
8. Bongiovanni, G., Coppersmith, D.,Wong, C.K.: An Optimum Time Slot Assignment Algorithm for an SS/TDMA System with Variable Number of Transponders. IEEE Transactions on Communications **29** (1981) 721–726
9. Gopal, I.S., Bongiovanni, G., Bonuccelli, M.A., Tang, D.T., Wong, C.K.: An Optimal Switching Algorithm for Multibean Satellite Systems with Variable Bandwidth Beams. IEEE Transactions on Communications **COM-30** (1982) 2475–2481
10. Gopal, I., Wong, C.: Minimizing the Number of Switching in an SS/TDMA System. IEEE Transactions on Communications (1985)
11. Afrati, F., Aslanidis, T., Bampis, E., Milis, I.: Scheduling in switching networks with set-up delays. In: AlgoTel 2002, Mèze, France (2002)
12. Crescenzi, P., Xiaotie, D., Papadimitriou, C.H.: On Approximating a Scheduling Problem. Journal of Combinatorial Optimization 5 (2001) 287–297
13. Desprez, F., Dongarra, J., Petitet, A., Randriamaro, C., Robert, Y.: Scheduling Block-Cyclic Array Redistribu- tion. IEEE Transaction on Parallel and Distributed Systems **9** (1998) 192–205
14. Bhat, P.B., Prasanna, V.K., Raghavendra, C.S.: Block Cyclic Redistribution over Heterogeneous Networks. In: 11th International Conference on Parallel and Distributed Computing Systems (PDCS 1998). (1998)
15. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problem. SIAM J. Comput. **5** (1976) 691–703
16. Cohen, J., Jeannot, E., Padoy, N.: Parallel Data Redistribution Over a Backbone. Technical Report RR-4725, INRIA (2003)
17. Padoy, N.: Redistribution de données entre deux grappes d'ordinateurs. Rapport de stage, de l'École Normale Supérieure de Lyon (2002)
18. Berge, C.: Graphs. North-Holland (1985)

Multidimensional Static Block Data Decomposition for Heterogeneous Clusters

Alexey Kalinov and Sergey Klimov

Institute for System Programming of Russian Academy of Sciences,
25, Bolshaya Kommunisticheskaya str., Moscow 1090045, Russia,
{ka,sergey}@ispras.ru

Abstract. We propose general static block and block-cyclic heterogeneous decomposition of multidimensional data over processes of parallel program mapped onto multidimensional process grid. The decomposition is compared with decomposition of two-dimensional data over two-dimensional process grid of Beaumont et al and with natural decomposition of three-dimensional data over three-dimensional process grid.

1 Introduction

A lot of parallel algorithms are based on homogeneous static block or block-cyclic decomposition of multidimensional data over multidimensional process grid. Those algorithms provide perfect load balancing for homogeneous parallel systems. But the load balancing that can be achieved using those algorithms on heterogeneous parallel systems is not good enough. In this paper we address to heterogeneous cluster consisting of processors of different performance interconnected with homogeneous communication equipment. The most common example of such systems is local networks in dedicated mode.

For a lot of algorithms mapping processes into multidimensional grid is superior to one-dimensional grid. In these cases the situation with mapping processes into multidimensional grid and subsequent distributing data over the process grid is much more difficult. It is proved in [1] that in the case of two-dimensional process grid the optimal solution for the problem is NP-complete. So, for multidimensional process grid we are forced to use heuristic solution in any case.

The decomposition proposed in this paper is further development of natural multidimensional data decomposition proposed in [2]. More advanced algorithms of processes mapping into process grid and data distribution over it are proposed.

The rest of the paper is organized as follows. In Section 2 we discuss problem of multidimensional heterogeneous static block-cyclic data decomposition. In section 3 we introduce a heuristic solution of the problem. In Section 4 we compare the proposed multidimensional decomposition with decomposition of 2D data over 2D process grid of Beaumont et al [1] and with natural decomposition of 3D data over 3D process grid [2].

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 907–914, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Problem of Multidimensional Block-Cyclic Decomposition on Heterogeneous Clusters

We call mDnD data decomposition a pair ξ, ζ where ξ specifies the *mapping* of processes onto m-dimensional (mD) grid and ζ specifies the *distribution* of n-dimensional (nD) data over the process grid. In this paper we consider only the case one process per processor and thus we don't distinguish process and processor.

Let set of processes is logically considered as m-dimensional process grid with sizes of edges $e_0, e_1, ..., e_{m-1}$ respectively. Mapping ξ assigns to each process its composite index$\rho = \{\rho_0, ..., \rho_{m-1}\}$, $\rho_k \in [0, e_k)$ in the grid.

1D1D homogeneous block-cyclic distribution partitions 1D data space into blocks of size a and distribute these blocks in a cyclic manner along the e processes. This means, that data element k is stored in process $\lfloor (k-1)/a \rfloor \bmod (e)$. In other terms, 1D1D block-cyclic distribution is partition of 1D data space onto *generalized* blocks of size $s = a \cdot e$, which in its turn is distributed over e processes. So, the task of block-cyclic distribution can be considered as task of block distribution of generalized block. In heterogeneous case 1D1D distribution is parameterized also by set $R = \{r_i\}$, $i \in [0, e)$ of process performances and amount of data of generalized block distributed on a process depend on *R*.

The m-dimensional block-cyclic distribution can be regarded as combination of m 1D1D block-cyclic distributions applied to dimensions of n-dimensional data space ($n \geq m$). The distribution with block size $a_0 \times a_1 \times ... \times a_{m-1}$ partitions the data space of size $N_0 \times N_1 \times ... \times N_{n-1}$ into generalized blocks of size $s_0 \times s_1 \times ... \times s_{n-1}$, where $s_i = a_k \cdot e_k$ if k-th 1D distribution is applied to i-th dimension of data and $s_i = N_i$ otherwise. Each generalized block in its turn is partitioned into $e_0 \cdot e_1 \cdot ... \cdot e_{m-1}$ blocks with size depending in common case on process performances. Such definition of m-dimensional block-cyclic distribution introduces "true" grid when every process has border only with one neighbor in all axis directions. This minimizes communication overheads but leads to imbalance in computational load.

Let the 1D1D distribution corresponding to k-th dimension of process grid is applied to η_k dimension of data grid. On process with composite index ρ is distributed block of data with volume $V_\rho = l^{\eta_0}_{0,\rho_0} \cdot ... \cdot l^{\eta_{m-1}}_{m-1,\rho_{m-1}} \cdot s_{\eta_m} \cdot ... \cdot s_{\eta_{n-1}}$ where $\eta = \{\eta_0, ..., \eta_{n-1}\}$ is permutation of numbers $\{1, ..., n-1\}$ and $\sum_{\rho_k=0}^{e_k-1} l^{\eta_k}_{k,\rho_k} = s_{\eta_k}$. So, the distribution ζ is specified by set $\{s_i\}, i \in [0, n)$ and sets $\{e_k\}$, $\{\eta_k\}$, and $\{l^{\eta_k}_{k,\rho_k}\}$, $l^{\eta_k}_{k,\rho_k} \in N$, $\sum_{\rho_k=0}^{e_k-1} l^{\eta_k}_{k,\rho_k} = s_{\eta_k}$, $k \in [0, m)$. Figure 1 presents a 2D3D data distribution with $\eta_0 = 1$, $\eta_1 = 0$.

Let process with composite index ρ has performance p_ρ. Then time of the block processing gets from the formula $t^{\xi,\zeta}_\rho = V_\rho / p_\rho$, time of parallel blocks processing is determined by $\max_\rho(t^{\xi,\zeta}_\rho)$, and objective of the task of data decomposition can be formulated as

$$\text{Objective 1} = \min_{\xi,\zeta}[\max_{\rho}(t_{\rho}^{\xi,\zeta})].$$

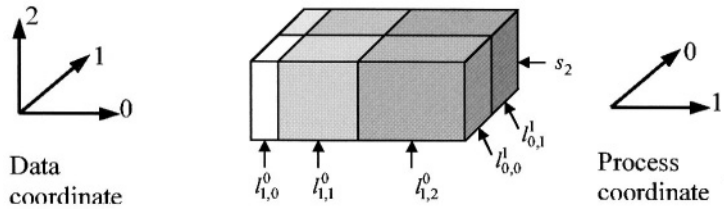


Fig. 1. 2D distribution of 3D generalized block. 1D1D distribution along first dimension of process grid is applied to zero dimension of the data ($\eta_0 = 1$) and 1D1D distribution along zero dimension of process grid is applied to first dimension of data ($\eta_1 = 0$).

3 Heuristic Solution

For simplicity we separate determination of mapping ξ and distribution ζ and use the following objective

$$\text{Objective 2} = \min_{\xi}\{\min_{\zeta}[\max_{\rho}(t_{\rho}^{\xi,\zeta})]\}.$$

3.1 Mapping of Processes onto Process Grid

We propose three heuristics for process mapping. First one is natural heuristics *NAT* introduced in [2]. Let total amount of processes is $E = e_0 \cdot ... \cdot e_{m-1}$, set of processes $\{p_j\}$ is sorted in ascending order according to process performances and $e_0 \leq e_1 \leq ... \leq e_{m-1}$. According to natural mapping, processes are mapped onto grid in column-wise order that is j-th process has the following coordinates in the grid:

$$\rho_k = \left\lfloor \frac{j - \sum_{l=k+1}^{m-1} \rho_l \cdot \prod_{i=0}^{l-2} e_i}{\prod_{i=0}^{k-1} e_i} \right\rfloor, \quad k \in [0, m).$$

Natural mapping is good enough for relatively "homogeneous" heterogeneous networks. For heterogeneous networks with heterogeneity essentially shifted to field of weak processes (for example, 1, 10, 11, 12, 13, ...) natural mapping leads to overloading of weak processes and to under loading of powerful ones. For such network we propose modification of natural mapping *NAT1*. Informally this modification can be introduced in the following way. First, we fill according to natural mapping all hyperplanes passing through grid node with coordinates $(0,\ldots,0)$. After that we fill with natural mapping the rest of the process grid.

More formal description is following. On i-th $i \in [1, m]$ step of mapping we select $(m-1)$-dimensional process grid of size $\{\tilde{e}_0, ..., \tilde{e}_{m-2}\}$: $\{\tilde{e}_0 = e_0, \ldots,$

$\tilde{e}_{m-i-1} = e_{m-i-1}, \tilde{e}_{m-i} = (e_{m-i+1}-1), \ldots, \tilde{e}_{m-2} = (e_{m-1}-1)\}$ such that, $\rho_0 \in [0, e_0), \ldots, \rho_{m-i-1} \in [0, e_{m-i-1}), \rho_{m-i} = 0, \rho_{m-i+1} \in [1, e_{m-i+1}), \ldots, \rho_{m-1} \in [1, e_{m-1})$. Processes are mapped onto this process grid according to natural mapping starting from process with lowest performance that was not mapped on the previous steps. On $m+1$ step of algorithm processes are mapped according to natural mapping onto reminder - m-dimensional process grid of size $\tilde{e}_0 = (e_0-1), ..., \tilde{e}_{m-1} = (e_{m-1}-1)$ such that $\rho_0 \in [1, e_0), \rho_1 \in [1, e_1), ..., \rho_{m-1} \in [1, e_{m-1})$. Figure 2 presents sequence of steps for 3D *NAT1* mapping. Numbers on I-IV refers to process subgrids onto which processes are mapped on the steps.

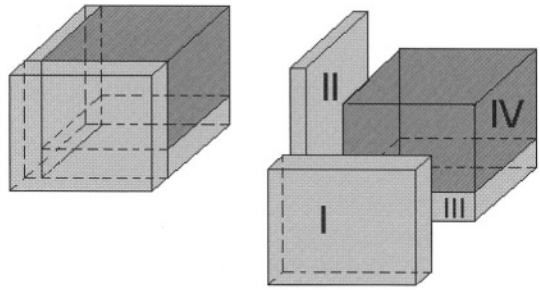


Fig. 2. Sequence of steps for 3D *NAT1* mapping. Numbers on I-IV refers to process subgrids onto which processes are mapped on the steps.

Modification *NAT2* for case of heterogeneity shifted to the field of powerful processes is symmetric to *NAT1*. The only difference is that process set is sorted in descending order and mapping is started with the most powerful process.

Table 1 presents mappings according to mentioned heuristics of processes with performance 1, 2, 3, 4, 5, 6, 7, 8, and 9 onto 2D grid 3x3.

We propose to solve optimization problem with all three mappings and choose the best one.

Table 1. Mapping of processes with performance 1, 2, 3, 4, 5, 6, 7, 8, and 9 onto 2D grid 3x3 according to

NAT1	NAT	NAT2
3 7 9	3 6 9	7 3 1
2 6 8	2 5 8	8 4 2
1 4 5	1 4 7	9 6 5

3.2 Distribution of Data over Process Grid

On this stage we have fixed mapping ξ and we are looking for optimal ζ.

Let $Sl_{k,i}$ is set of processes with composite index $\rho : \rho_0 \in [0, e_0), \ldots, \rho_{k-1} \in [0, e_{k-1}), \rho_k = i, \rho_{k+1} \in [0, e_{k+1}), \ldots, \rho_{m-1} \in [0, e_{m-1})$ and $f_{k,i}^{\xi,\zeta} = \max_{\rho \in Sl_{k,i}} (t_\rho^{\xi,\zeta})$, $k \in [0, m)$, $i \in [0, e_k)$. Suppose $l_{k,i}^{\eta_k} \in R$. The requirement of local minimum of function $\max_\rho(t_\rho^{\xi,\zeta})$ at ζ_0 in this case is

$$\forall \{k \in [0, m), i \in [0, e_k)\} \Rightarrow f_{k,i}^{\xi,\zeta_0} = const(\zeta_0)$$

We propose this requirement as objective for solving optimization problem of data distribution and propose to solve this optimization problem for every k independently. So, we try to reach equality of all $f_{k,i}^{\xi,\zeta}$ for every $k \in [0, m)$ independently.

To reach equality of $f_{k,i}^{\xi,\zeta}$ we solve task of moving of borders between blocks of data distributed on $Sl_{k,i}$ on the force of difference $f_{k,i}^{\xi,\zeta} - f_{k,i+1}^{\xi,\zeta}$ similarly to moving partition between two volumes of gas on the force of pressure difference in them. It is the physical analogy that is original for the proposed distribution.

As first approximation for optimization problem we use natural distribution introduced in [2]. According to this distribution $l_{k,i}^{\eta_k}$ are computed using the formula:

$$l_{k,i}^{\eta_k} = \frac{\sum_{\rho \in Sl_{k,i}} p_\rho}{\sum_\rho p_\rho} \cdot s_{\eta_k}, \quad k \in [0, m), \quad i \in [0, e_k).$$

After optimization $l_{k,i}^{\eta_k}$ are rounded. The sum $\sum_{i=0}^{e_k-1} l_{k,i}^{\eta_k} = s_{\eta_k}$ may be less than s_{η_k}. In that case $l_{k,i}^{\eta_k}$ with greater difference $(s_{\eta_k} \cdot \sum_{\rho \in Sl_{k,i}} p_\rho - l_{k,i}^{\eta_k} \cdot \sum_\rho p_\rho)$ are iteratively incremented to achieve equality.

4 Experimental Results

Proposed data decomposition was compared with 2D2D data decomposition of Beaumont et al [1] and with 3D3D natural decomposition introduced in [2].

4.1 Comparison with 2D2D Decomposition of Beaumont et al.

For 2D2D case we have conducted two computational experiments. As factor of comparison we use the ratio of time of computation with proposed decomposition to time of computation with decomposition of Beaumont et al. The size of generalized block is 1000x1000. A factor characterizing heterogeneity of the network we use *heterogeneity level* computed as the ratio of maximal to minimal process performance values $(\max_j p_j \Big/ \min_j p_j)$. Figure 3 (a) presents plot of this factor against heterogeneity level (axis X) and size of square process grid (axis Y). Every point of the plot is computed as average value of the factor computed

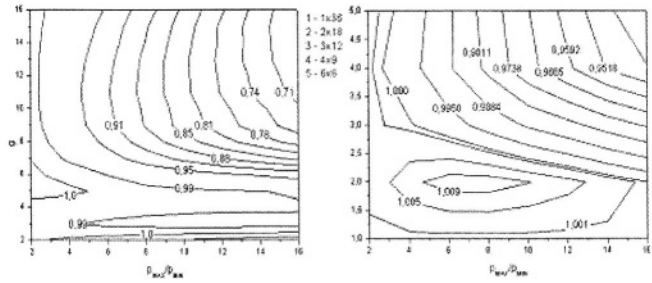


Fig. 3. The average ratio of time of computation with proposed decomposition to time of computation with decomposition of Beaumont et al against the ratio of maximal and minimal process performance values and (a) – size of square process grid, (b) - different variants of process grid consisting of 36 processes (1 – 1x36, 2 – 2x18, 3 – 3x12, 4 - 4x9, 5 – 6x6)

from 32000 random variants of the network with the heterogeneity level and the size. Presented results shows that data decomposition of Beaumont et al is better then proposed decomposition (ratio greater then 1) only in limited region. Of course, it is better for size equal to 2 when it is proved optimal solution (the maximal benefit is 1,01762 in case 2x2 grid and heterogeneity 16). Figure 3 (b) presents plot of the average times ratio against the heterogeneity level (axis X) and different process grid: 1x36, 2x18, 3x12, 4x9, 6x6 consisting of 36 processes (axis Y). One can see that for essentially different sizes of 2D process grid and low heterogeneity results are practically the same but in the remainder proposed decomposition is a bit better. We for purpose choose the case of 36 processes because for 6x6 grid the both decompositions have advantage over other. It is interesting to see what are the results provided by the both decomposition for different variants. The heterogeneity level $\max_j p_j \Big/ \min_j p_j$ does not fully characterize performance heterogeneity. For the estimation of "distribution of heterogeneity" of heterogeneous network we introduce two functions:

$$Fhet_\min = -\frac{1}{(E-1)} \ln \left(\frac{\left(\min_j p_j \right)^E}{p_0 \cdot \ldots \cdot p_{E-1}} \right),$$

$$Fhet_\max = -\frac{1}{(E-1)} \ln \left(\frac{\left(\max_j p_j \right)^E}{p_0 \cdot \ldots \cdot p_{E-1}} \right).$$

Former characterizes shift of heterogeneity to the field of lower performances and the latter characterizes shift of heterogeneity to the field of

higher performances. We call *internal heterogeneity* of the network the value max(*Fhet*_min, *Fhet*_max).

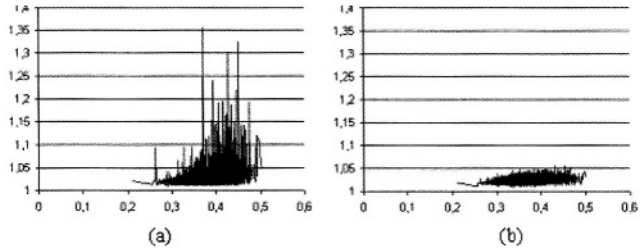


Fig. 4. Ratio of time of computation with (a) – decomposition of Beaumont et al and (b) – proposed decomposition to time of computation with ideal decomposition against internal heterogeneity of the network for grid 6x6 and heterogeneity level equal to 2

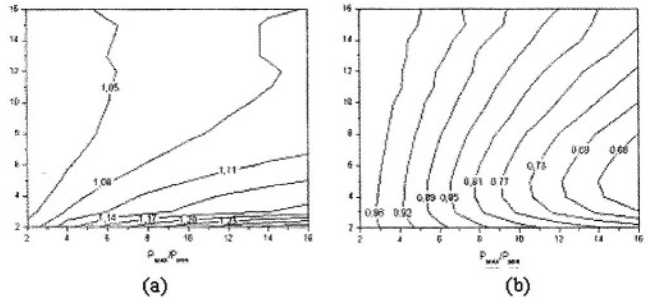


Fig. 5. The average ratio of time of computation with proposed decomposition to time of computation with (a) – ideal decomposition and (b) – natural decomposition against the ratio of maximal and minimal process performance values and size of square process grid

Let examine variant – grid 6x6 and heterogeneity level equal to 2 with a bit better average value for decomposition of Beaumont et al. Figure 4 presents ratio of time of computation with the both decompositions to time of computation with ideal decomposition against internal heterogeneity of the network for that variant. Time of ideal decomposition is computed as $\sum_\rho V_\rho \Big/ \sum_\rho p_\rho$. One can see that proposed decomposition has is less spread in results than decomposition of Beaumont et al.

We did not inspect time of computation of data decompositions. But for all cases except 2x2 process grid proposed decomposition is computed faster then the decomposition of Beaumont et al and for not squire grids it is several orders faster.

4.2 3D3D Case

For 3D3D we examine efficiency of proposed decomposition relative to ideal and natural ones. Figure 5 presents plots of average ratio of time of computation with proposed decomposition to time of computation with (a) ideal decomposition and (b) natural decomposition against heterogeneity level (axis X) and size of square process grid (axis Y). The figure 5(a) shows that proposed decomposition is essentially worse than ideal one in region of small networks. The figure 5(b) shows that for high heterogeneity level proposed decomposition essentially better than natural one.

5 Conclusion

We proposed general heterogeneous block data decomposition of multidimensional data over multidimensional process grid that is further development of the simplest general heterogeneous multidimensional decomposition - natural block data decomposition. We showed that proposed decomposition in general is better than specialized decomposition of two-dimensional data over two-dimensional process grid of Beaumont et al. We also showed for three-dimensional case that proposed decomposition in most cases is close to ideal one and that it is much better than natural one in the case of high heterogeneity level.

References

[1] Olivier Beaumont, Vincent Boudet, Antoine Petitet, Fabrice Rastello, and Yves Robert: A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers). IEEE Trans. Computers. Vol.50, 10 (2001) 1052-1070

[2] Y.Dovolnov, A.Kalinov, and S.Klimov: Natural Block Data Decomposition for Heterogeneous Clusters. Proceedings of HCW'03, IEEE CS Press, Nice, France, 22 April 2003

A Job Scheduling Strategy for Heterogeneous Multiprogrammed Systems

Piyush Maheshwari

School of Computer Science and Engineering, The University of New South Wales, Sydney NSW 2052 Australia piyush@cse.unsw.edu.au

Abstract. Mapping and scheduling in multiprogrammed environment has recently attracted more attention of the researchers. Most of the past algorithms use the First Come First Serve (FCFS) strategy, using time-sharing, space-sharing or the combination of both. However, there are limitations when FCFS is implemented in a real system. In order to overcome those drawbacks, we propose a new scheme, called First Input First Output–Best Fit (FIFO-BF), which is used in a mapping policy, Adaptive Multiprogrammed Mapping (AMM). Taking heterogeneity into account, another mapping policy, Heterogeneous Multiprogrammed Mapping (HMM), is presented. Due to its complex nature, little work has been done in this area.

1 Introduction

With the growing needs of multiprogrammed general-purpose parallel systems, different policies dealing with job assignment are emerging. A good job scheduler in a multiprogrammed parallel processing environment balances the user's desire to complete his jobs as quickly as possible with the system's desire to service as many jobs as possible [6]. Scheduling schemes can be classified as *time-sharing, space-sharing* or a mixture of both. In purely time-shared systems, all jobs have a certain quantum of time served by processors. Space-sharing tends to provide each job more constant allocation of a fewer number of processors than does time-sharing [4].

Most work has been done using space-sharing based on First Come First Serve (FCFS) policy [5] [6], which present simplified problem since schedulers do not compare jobs and assume that all (size of) jobs arrive at equal opportunity. However, in a realistic environment, it may happen that a large job in a queue will occupy all the processors and thus can cause blockade.

In order to avoid overheads occurred before scheduling is done, we first present our Adaptive Multiprogram Mapping (AMM) policy based on First In First Out–Best Fit (FIFO-BF) strategy. FIFO-BF deals with jobs according to the arrival sequence. This strategy is same as FCFS, however, when a job cannot be assigned due to shortage of processors, the scheduler continues searching the next proper job instead of stopping scheduling at all. We then extend the algorithm to multiprogrammed heterogeneous environment and present a Heterogeneous Multiprogrammed Mapping (HMM).

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 915–920, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The rest of the paper is organized as follows. Section 2 presents our first proposed job assigned scheme. The second scheme for heterogeneous systems is presented in Section 3. Extensive simulations and experiments have been done to verify the first policy, however, due to space limitations they are not presented here. Section 4 presents the simulation results of the HMM scheme. Section 5 concludes the paper.

2 Adaptive Multiprogrammed Mapping

In order to adaptively utilize the system resources and maintain the dynamic load balancing of processors, it is critical to adjust the system environment according to the jobs arrival and departure. Thus, the focus should be on to allocate programs on processors in a way that maximize throughput and minimizes response time.

In order to reduce the context switching rate, a common approach is to divide the processors into several independent partitions and then execute each job on its own partition. This type of space-sharing eliminates the competition between jobs for processors, and is often used in massively parallel systems [1][3].

We assume that when a job arrives, it brings information about how many processors are needed for an optimal assignment. Presume the number of needed processors is k and the total number of processors is p. If $k<p/2$, it means that the application (or job) needs rather little resources, so put it into the *small* queue. Similarly, *large* queue includes programs' valid assigned processor number larger than $p/2$.

For both the queues, First Input First Output–Best Fit (FIFO-BF) is employed to decide the next assigned job.

When a job from *large* queue comes, the algorithm checks the system load. If currently available processors p_{avai} are larger than $p/2$, it assigns the job. It then checks the load again and if p_{avai} is still less than $p/2$, it inputs the next job from *large* queue. But if this $k \gg p_{avai}$, it skips to the following one. The algorithm recursively does this until $p_{avai} < p/2$ or there are no more jobs, gets programs from *small* queue and assigns. Note that when a job is finished, occupied processors would be released to processor pool and system load is changed. So at any time the system load may be different according to the arrival and depe of jobs.

Similarly if first comes a small job, the algorithm just assigns it and checks the system load. Under the condition of $p_{avai} > p/2$, input *large* one if any available. If no, consider next *small* job.

The primary goal of this policy is to allocate relatively small jobs to processors under heavy system loads and relatively large jobs during conditions of light load.

3 Assignement Strategy for Heterogeneous Processors

Recently many policies have been presented for multiprogrammed parallel machines. However, only a few deal with heterogeneous processors. In heterogeneous systems, the optimization for job scheduling must be carried out at both macro and micro levels [2]. In multiprogrammed environment, the basic idea is to utilize idle processors serving more jobs, which would be different if we take heterogeneity into account. So besides at the macro level, scheduler makes decision of finding proper job allocated to available processors, even inside one job, different task types should be put on corresponding individual processor. Our algorithm, presented below, takes care of allocation of both jobs and subtasks.

3.1 Heterogeneous Multiprogrammed Mapping (HMM)

The problem addressed is as follows. Given some parallel jobs and a heterogeneous distributed computing system with n processors, assign each job to the processors in a way that overall load is balanced and all the processing nodes are fully utilized.

We set up a queue for each processor to specify task nodes assigned to it. Inside each queue, there are three attributes to identify every node: the order of the task node, its start and finish time unit on this processor. For example, a queue of p_0:{{0,0,2},{2,3,7}} means nodes 0 and 2 are to allocated the processor p0 and node 0 starts at 0, finishes at unit 2; node 2 starts at unit 3, finishes at 7. By using this queue, we can supervise the idle time of a processor and it becomes easy to insert next job to fill in the holes according to the current load.

A first-come first-served queue is maintained for the jobs. Any job in this queue has non-preemptive priority over others. Upon the arrival of the first job, the system allocates it and gets queues for each processor. In order to see its necessity of inputing next one, we need to find the longest idle time of processors and compare it with average load of next application. The point is that if the processors have got proper workload, no second input is needed.

If a second one is needed, assign nodes of the job one by one since there would be adjustment due to conflict between assignment of jobs in the same set of processors. Firstly, assign the node according to certain criteria. Then check whether the processor is available at this time, if not, compare time for waiting for this processor plus execution time and execution time from an available processor. Based on this, all the nodes should be assigned to individual processors. A point to be noted is that there is no interrupt during the execution of node. Always the task node with earlier start time has higher priority and once it starts, it cannot be stopped. If next one's original time is earlier than this finished time, the start time will be changed to last task's finish time automatically. The above step is applied iteratively until no more job is left.

4 Simulation Results for HMM

We used three jobs and three processors as shown in Figure 1.

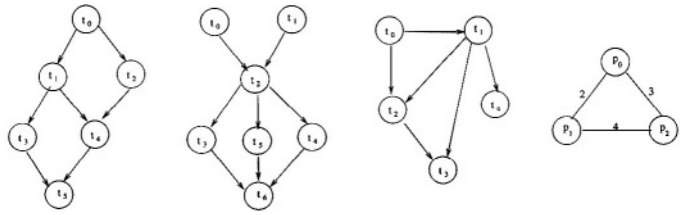


Fig. 1. The input of simulation 1

Inside the job queue, there are $\{T_1, T_2, T_3\}$ representing T_1 comes first, then T_2 and so on.

We first assign T_1 on the system. In order to let problem be more general and minimize the run time of mapping algorithm, we use algorithm in [7]. The same notation is used as in Section 3. We get the results like Figure 2.

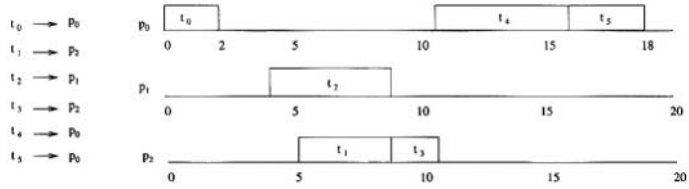


Fig. 2. The result of first assignment

A queue for p_0, p_1, p_2 can be set up as: p_0:{{0,0,2},{4,11,16},{5,16,18}}; p_1:{{2,4,8}}; p_2:{{1,5,8},{3,8,11}}. To see whether the processors are busy all the time, calculate within the finish time of last task node the longest idle time *IdP* for each processor. In this case, node 0 starts at unit time 0, finishes at 2, and node 4 runs from time 11 to 16. So between time 2 and 11, processor 0 is idling. Nodes 4 and 5 are executed continuously, no idle time. Thus, IdP_1 equals to 9. Similarly, we can get IdP_2=10, IdP_3=7 and average equals 8.6. Now all the calculations related to Job 1 are completed. Next, input the second job.

Table 1. Task Execution Time Matrix of Job 2

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
p_0	5	7	4	2	8	2	8
p_1	1	3	3	5	3	2	4
p_2	4	2	2	4	7	1	3

From the execution matrix (Table 1), we need to work out average node execution time. Because this will be only worked as reference, we just do some calculation roughly. Since we do not know final assignment at this time, which

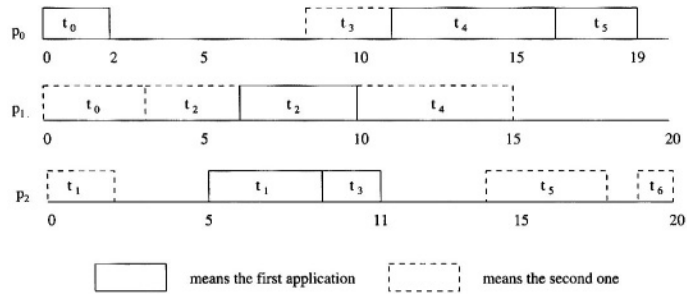


Fig. 3. The result of second assignment

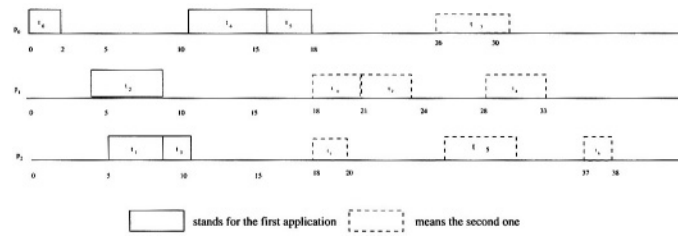


Fig. 4. The result of serial assignment

Table 2. Processors' Utilization in Different Systems

	p_0	p_1	p_2
Single program system	34.2%	39.4%	23.7%
Multiprogrammed system using HMM	68.4%	75%	70%

means the actual execution time for each task is not fixed yet, we get average execution time *(Ave)* for every task. For instance, in Table 1, Ave_0=3.3, Ave_1=4 and so on. Finally calculate the whole average node execution time equalling to 3.8, less than average processor idle time, so it can be chosen as input. This time, the task assignment is different from the first one. From the ready tasks, one by one, we compare the execution time on different processors and select the best one to assign. Before adding to the processor queue, see whether this assigned processor is available, if not, compare the time of waiting the processor available plus execution time and best execution time from available processor, to decide whether it needs to be transfered. In this example, t_0 assigned to p_1 and p_1 is available, so simply add it to the head of p_1 queue; t_1 is assigned to p_2 and same as t_0; t_2 allocated to p_1 and start time is unit 3, earlier than first one in the queue, insert it before the others. The execution time for t_2 is 3 and finish at unit 6, later than the next start time, so all the other nodes in the queue compare start time with last one's finish time and shift it automatically, like t_2 of the first application changed to {2,6,10}, then t_4 {4,10,15}. The assignement continues until the nodes are completely allocated, as shown in Figure 3.

Then, calculate average idle time again. It equals to 3.7, while the average task execution time of the third one is 4. So now the assignment for the system of three processors is finished.

Now we discuss the efficiency of this multiprogrammed algorithm compared to a single program system. We list the allocation for Jobs 1 and 2 separately with the same methodology in Figure 4. From the simulation results, speedup S and efficiency E compared with single programmed system can be worked out. In this case, the run time for two jobs is 18 and 21 respectively and in multiprogrammed environment is 20. So S=38/20 and E=0.65. We define processor's utilization as the ratio of actual CPU running time to the whole time processor being occupied. The great difference through the analysis from our result is shown in Table 2.

5 Conclusion

In this paper, we have introduced two different kinds of mapping strategies in a multiprogrammed multiprocessor environment. The first one, the Adaptive Multiprogrammed Mapping (AMM) is an improved methodology based on our past work for homogeneous systems. In reality, most applications contain more than one type of embedded parallelism and running them on homogeneous processors can bring decreased speedup and/or decreased parallelism. We have extended the AMM algorithm to heterogeneous multiprogrammed environment. The HMM algorithm is about mapping multijobs in heterogeneous systems. Early simulations and analysis have given satisfactory results. In future, we would like to implement this strategy on real parallel heterogeneous machines.

References

1. *Connection Machine CM5 Technical Summary.* Cambridge, Mass.: Thinking Machnes Corp., 1992.
2. Hanh, P.H., Simonenko, V.: Objective-oriented algorithm for job scheduling in parallel heterogeneous systems. *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science,* pp193-213, Apr. 1997.
3. Kessler, R., Schwarzmeier, J.: CRAY T3D: A new dimension for cray research *Proc. COMPCON,* pp 176-182, 1993.
4. Mccann, C., Vaswani, R., Zahorjan, J.: A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors *ACM Trans. on Computer systems,* 11(2), May, 1993.
5. Naik, V.K., Setia, S.K., Squillante, M.S.: Processor Allocation in Multiprogrammed Distributed Memory Parallel Computer Systems *J. of Parallel and distributed computing,* 46, pp28-47, 1997.
6. Wang, F., Franke, H., Papaefthymiou, M., Pattnaik, P., Rudoph, L., Squillante, M.S.: A gang scheduling design for multiprogrammed parallel computing enviionments. *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science,* pp111-125, Apr. 1996.
7. Yang, W., P. Maheshwari, P.: Mapping precedence tasks onto a heterogeneous distributed system. *Int'l Conf. on Parallel and Distributed Computing and Networks,* pp 596-600, Dec. 1998.

Classification of Partitioning Problems for Networks of Heterogeneous Computers

Alexey Lastovetsky and Ravi Reddy

Department of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland {alexey.lastovetsky, manumachu.reddy}@ucd.ie

Abstract. The paper presents a classification of mathematical problems encountered during partitioning of data when designing parallel algorithms on networks of heterogeneous computers. We specify problems with known efficient solutions and open problems. Based on this classification, we suggest an API for partitioning mathematical objects commonly used in scientific and engineering domains for solving problems on networks of heterogeneous computers. These interfaces allow the application programmers to specify simple and basic partitioning criteria in the form of parameters and functions to partition their mathematical objects. These partitioning interfaces are designed to be used along with various programming tools for parallel and distributed computing on heterogeneous networks.

1 Introduction

Parallel solution of regular and irregular problems on a heterogeneous network of computers typically consists of two macro-steps:

- Decomposition of the whole problem into a set of sub-problems that can be solved in parallel by interacting processes;
- The mapping of these parallel processes to the computers of the network.

An *irregular* problem is characterized by some inherent coarse-grained or large-grained structure. This structure implies a quite deterministic decomposition of the whole problem into relatively small number of subtasks, which are of different size and can be solved in parallel. Correspondingly, a natural way of decomposition of the whole program, which solves the irregular problem on a network of computers, is a set of parallel processes, each solving its subtask and all together interacting via message passing. As sizes of these subtasks are typically different, the processes perform different volumes of computation. Therefore, the mapping of these processes to the computers of the executing HNOC should be performed very carefully to ensure the best execution time of the program.

The most natural decomposition of a *regular* problem is a large number of small identical subtasks that can be solved in parallel. As those subtasks are identical, they are all of the same size. Multiplication of two $n \times n$ dense matrices is an example of a regular problem. This problem is naturally decomposed

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 921–929, 2004.
© Springer-Verlag Berlin Heidelberg 2004

into n^2 identical subtasks, each of which is to compute one element of the resulting matrix. The main idea behind an efficient solution to a regular problem on a heterogeneous network of computers is to transform the problem into an irregular problem, the structure of which is determined by the structure of the executing network rather than the structure of the problem itself. So, the whole regular problem is decomposed into a set of relatively large sub-problems, each made of a number of small identical subtasks stuck together. The size of each subproblem, that is, the number of elementary identical subtasks constituting the subproblem, depends on the speed of the processor, on which the subproblem will be solved. Correspondingly, the parallel program, which solves the problem on the heterogeneous network of computers, is a set of parallel processes, each solving one subproblem on a separate physical processor and all together interacting via message passing. The volume of computations performed by each of these processes should be proportional to its speed.

Thus, while the step of problem decomposition is trivial for irregular problems, it becomes key for a regular problem. In fact, at this very step the application programmer designs a heterogeneous data parallel algorithm by working out a generic decomposition of the regular problem parameterized by the number and speed of processors. Most typically the generic decomposition takes the form of data partitioning.

Existing programming systems for heterogeneous parallel computing [1]- [4] support the mapping of parallel algorithms to the executing network but provide very poor support for generic heterogeneous decomposition of regular problems implied by the number and speed of processors. The application programmers need to solve corresponding data partitioning problems and design and implement all supportive code from scratch. Our own experience with using mpC and HMPI for parallel solution regular problems on networks of computers has shown how tedious and error-prone this step of application development can be.
This motivated us to try and automate the step of heterogeneous decomposition of regular problems by designing a library of functions solving typical partitioning problems for networks of heterogeneous computers. Our original approach was to do it by just collecting existing algorithms, designing an API to these algorithms and implementing the API. The main problem we came across on this way was that no classification of partitioning problems was found that might be used as a basis of API design. Existing algorithms created a very fragmented picture. Therefore the main goal of our research became to classify partitioning problems for networks of heterogeneous computers. Such classification had to help to specify problems with known efficient solutions and identify open problems. Then based on this classification an API would have to be designed and partially implemented (for problems that have known efficient solutions). An additional requirement to this classification was that it had to be useful for distributed computing on networks as well.

Our approach to classification of partitioning problems is based on two corner stones:

- A realistic performance model of networks of heterogeneous computers,
- A natural classification of mathematical objects most commonly used in scientific, engineering and business domains for parallel (and distributed) solving problems on networks of heterogeneous computers.

This paper is structured as follows. In section 2, we describe the realistic performance model of networks of heterogeneous computers. In section 3, we identify the mathematical objects. In section 4, we classify the problems encountered during partitioning of sets. Based on this classification, we suggest an API for partitioning sets. Due to limitations on the length of the paper, we only briefly outline the classification of partitioning problems for matrices, graphs, and trees, and the corresponding API.

2 Performance Model of Networks of Heterogeneous Computers

Most algorithms of data partitioning for networks of heterogeneous computers use performance models where each processor is represented by a single positive number that characterizes its relative speed. Data partitioning is performed such that the volume of computations executed by the processor be proportional to its speed.

It is a well known fact that the absolute speed of a processor is a decreasing function of data simultaneously stored in the memory of the processor and used by the processor in computations. The memory typically has a hierarchical structure with levels of fixed sizes. Higher levels are substantially faster and smaller than lower ones. Therefore, as more processed data are stored in the memory, the more levels of the memory hierarchy they fill. As a result more data become stored in slow memory. This increases the average execution time of a single arithmetic operation decreasing the speed of the processor. Figure 1(a) illustrates this fact using matrix multiplication on two computers: modern Dell computer csserver (Linux, main memory of 513960 KB, cache of 512 KB), and relatively old Sun computer csultra01 (Solaris, main memory of 524888 KB, cache of 1024KB).

Nonetheless the above simple performance model is suitable in many real life situations where the relative speeds of the processors involved in the execution of the application are a constant function of the size of the problem and thus can be approximated by single numbers. Figure 1(b) gives an example of such a situation. The relative speed of computers csserver and csultra01 demonstrated on matrix multiplication may be approximated by a single number, 3, with sufficient accuracy. However if the processors have significantly different sizes at each level of their memory hierarchies, they may demonstrate significantly different relative speeds dependent on the size of the problem. Figure 2 gives us relative speeds of different pairs of computers experimentally obtained during multiplication of matrices of different sizes. If we use such networks of heterogeneous

computers for execution of parallel or distributed algorithms, we cannot represent their relative speeds by single numbers. Realistically in this case we must represent the speed by a function of the size of the problem.

Therefore, we suggest using a more realistic model that takes into account the impact of heterogeneity of memory and memory hierarchies on performance.Under this model, each processor is represented by a decreasing function of the problem size that characterizes its speed. In practice, the function is obtained by interpolation of a relatively small number of experimental results for different problem sizes. Constant functions will be just a special case. In addition, the model takes account of memory limitations and characterizes each processor by the maximal size of problem it can solve. The latter feature makes little sense when computing on a local network because in this case the user has some idea about the power of available computers and the size of problem that can be solved on the network. This feature does make sense when the user solves problems on a global network. In that case, the user may have no idea of the number and configurations of computers that may be involved in computations. Therefore if the problem size is big enough, some computer whose speed is estimated based on a small number of experiments may be assigned to solve a subproblem of the size that cannot be solved on the computer at all.

3 Classification of Partitioning Problems

The core of scientific, engineering or business applications is the processing of some mathematical objects that are used in modeling corresponding real-life problems. In particular, partitioning of such mathematical objects is a core of any data parallel algorithm. Our analysis of various scientific, engineering and business domains resulted in the following short list of mathematical objects commonly used in parallel and distributed algorithms: **sets** (ordered and non-ordered), **matrices** (and multidimensional arrangements), **graphs** and **trees.**

These mathematical structures give us the second dimension for our classification of partitioning problems. In the next section, we present our approach

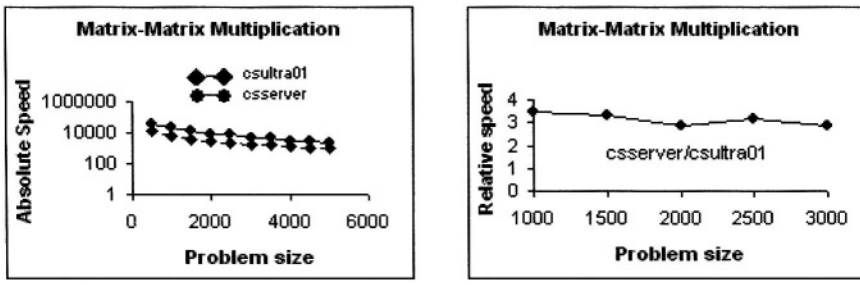


Fig. 1. (a) Absolute Speeds of **csserver** and **csserver01** against the size of the problem in matrix manipulation. (b) The relative speed of these computers against the size of these problems.

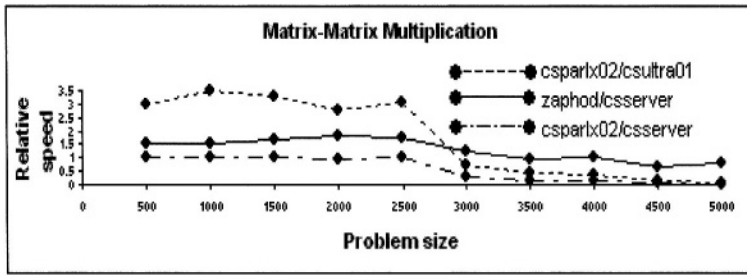


Fig. 2. Relative speeds of computers against the size of the problem in matrix multiplication. Computers involved are: **zaphod**(main memory of 254576 KB, cache of 512 KB), **csparlx02**(126176 KB, 512 KB), **csserver**(513960 KB, 512 KB), **csultra01**(524288 KB, 1024 KB).

to classification of partitioning problems using sets as mathematical objects. We also suggest an API based on the classification.

4 Partitioning Problems for Sets and Ordered Sets

There are two main criteria used for partitioning a **set:**

a) The number of elements in each partition should be proportional to the speed of the processor owning that partition.
b) The sum of weights of the elements in each partition should be proportional to the speed of the processor owning that partition.

Additional restrictions that may be imposed on partitioning of an **ordered set** are:

- The elements in the set are well ordered and should be distributed into disjoint contiguous chunks of elements.

The most general problem of partitioning a set can be formulated as follows:

- Given: (1) A set of **n** elements with weights $\mathbf{w_i}$ (i=0,…,n-1), and (2) A Well ordered set of **p** processors whose speeds are functions of the size of the problem, $\mathbf{s_i = f_i(x)}$, with an upper bound $\mathbf{b_i}$ on the number of elements stored by each processor (i=0,... ,p-1),
- Partition the set into **p** disjoint partitions such that: (1) The sum of weights in each partition is proportional to the speed of the processor owning that partition, and (2) The number of elements assigned to each processor does not exceed the upper bound on the number of elements stored by it.

The most general partitioning problem for an ordered set can be formulated as follows:

- Given: (1) A set of **n** elements with weights $\mathbf{w_i}$ (i=0,... ,n-1), and (2) A Well ordered set of **p** processors whose speeds are functions of the size of the problem, $\mathbf{s_i} = \mathbf{f_i(x)}$, with an upper bound $\mathbf{b_i}$ on the number of elements stored by each processor (i=0,... ,p-1),
- Partition the set into **p** disjoint contiguous chunks such that: (1) The sum of weights of the elements in each partition is proportional to the speed of the processor owning that partition, and (2) The number of elements assigned to each processor does not exceed the upper bound on the number of elements stored by it.

The most general partitioning problems for a set and an ordered set are very difficult and open for research. At the same time, there are a number of important special cases of these problems with known efficient solutions. The special cases are obtained by applying one or more of the following simplifying assumptions:

- All elements in the set have the same weight. This assumption eliminates **n** additional parameters of the problem.
- The speed of each processor is a constant function of the problem size.
- There are no limits on the maximal number of elements assigned to a processor.

Table 1. Special cases of partioning of a set

Mode of Parallel Computation	**Weights of elements are the same**	**Weights of elements are different**
Speeds are functions of problem size & no limits on number of elements stored by each processor.	**Complexity**	No Known Results
	$O(p \times \log n)$	
Speeds are single constant numbers and an upper on number of elements stored that each processor can hold.	**Complexity**	NP-Hard?
	O(p)	
Speeds are single constant numbers & no limits on number of elements stored that each processor can hold.	**Complexity**	NP-Hard?
	O(p)	

One example of a special partitioning problem for a set is:

- Given: (1) A set of **n** elements, and (2) A well-ordered set of **p** processors whose speeds are represented by single constant numbers, $s_0, s_1, \ldots, s_i$.
- Partition the set into **p** disjoint partitions such that the number of elements in each partition is proportional to the speed of the processor owning that partition.

This problem is trivial of the complexity O(p). Another example of a special partitioning problem for a set is:

- Given: (1) A set of **n** elements, and (2) A well-ordered set of **p** processors whose speeds are functions of the size of the problem, $\mathbf{s_i} = \mathbf{f_i(x)}$ (i=1,...,p-1).
- Partition the set into **p** disjoint partitions such that the number of elements in each partition is proportional to the speed of the processor owning that partition.

Table 2. Special cases of partioning of an ordered set

Mode of Parallel Computation	Weights of elements are the same	Weights of elements are different	
		Rearrangement of Processors	
		Allowed	Not allowed
Speeds are functions of the size of the problem & no upper bound exists on number of elements that each processor can hold.	**Complexity** O(p× log n)	No Known Results	No known Results
Speeds are single constant numbers & an upper bound exists on number of elements that each processor can hold.	**Complexity** O(p)	No Known Results	No known Results
Speeds are single constant numbers & no limits exist on number of elements stored that each processor can hold.	**Complexity** O(p)	No Known Results	No known Results

An algorithm of the complexity O (p $\times \log n$) solving this problem is given in [5].

Table 1 and Table 2 summarize specific partitioning problems for a set and an ordered set respectively and their current state to the best knowledge of the authors.

Based on this classification, we suggest the following API to application programmers for partitioning a set into **p** disjoint partitions:

typedef double (*User_defined_metric)(**int** p, **const double** *speeds, **const double** *actual, **const double** *ideal);
int Partition_set (**int** p,**int** pn, **const double** *speeds, **const int** *psizes,
const int *mlimits, **int** n, **const double** *w **int** ordering, **int** processor_reordering,
int type_of_metric, User_defined_metric umf, **int** *metric, **int** *np)

Parameter **p** is the number of partitions of the set. Parameters **speeds** and **psizes** specify speeds of processors for **pn** problem sizes. These parameters are

1D arrays of size **p**×**pn** logically representing 2D arrays of shape [**p**][**pn**]. The speed of the **i**-th processor for **j**-th problem size is given by the [**i**] [**j**]-th element of **speeds** with the problem size itself given by the [**i**][**j**]-th element of **psizes.** Parameter **mlimits** gives the maximum number of elements that each processor can hold.

Parameter **n** is the number of elements in the set, and parameter **w** is the weights of its elements. If **w** is **NULL**, then the set is partitioned into **p** disjoint partitions such that criterion (a) is satisfied. If parameters **w**, **speeds** and **psizes** are all set to **NULL,** then the set is partitioned into **p** disjoint partitions such that the number of elements in each partition is the same. If **w** is not **NULL,** then the set is partitioned into **p** disjoint partitions such that criterion (b) is satisfied. If **w** is not **NULL** and **speeds** is **NULL,** then the set is partitioned into **p** equally weighted disjoint partitions.

Parameter **ordering** specifies if the set is well ordered (=1) or not (=0).

Parameter **type_of_metric** specifies whose metric should be used to determine the quality of the partitioning. If **type_of_metric** is **USER_SPECIFIED,** then the user provides a metric function **umf,** which is used to calculate the quality of the partitioning. Otherwise, the system-defined metric is used which is the weighted Euclidean metric. The output parameter **metric** gives the quality of the partitioning, which is the deviation of the partitioning achieved from the ideal partitioning satisfying the partitioning criteria.

If **w** is **NULL** and the set is not ordered, the output parameter **np** is an array of size **p**, where **np[i]** gives the number of elements assigned to the **i**-th partition. If the set is well ordered, processor **i** gets the contiguous chunk of elements with indexes from **np**[**i**] upto **np[i]+np[i+1]-1.**

If **w** is not **NULL** and the set is well ordered, then the user needs to specify if the implementations of this operation may reorder the processors before partitioning (Boolean parameter **processor_reordering** is used to do it). One typical reordering is to order the processors in the decreasing order of their speeds.

If **w** is not **NULL,** the set is well ordered and the processors cannot be reordered, then the output parameter **np** is an array of size **p**, where **np[i]** gives the number of elements of the set assigned to the **i**-th partition. Specifically, processor **i** gets the contiguous chunk of elements with indexes from **np[i]** upto **np[i]+np[i+1]-1**.

If **w** is **NULL,** the set is well ordered and the processors may be reordered, then **np** is an array of size **2**×**p**, where **np[i]** gives index of a processor and **np[i+1]** gives the size of the contiguous chunk assigned to processor given by the index **np[i].**

If **w** is not **NULL** and the set is not ordered, then **np** is an array of size **n**, containing the partitions to which the elements in the set belong. Specifically, **np[i]** contains the partition number in which element **i** belongs to.

Some of the typical examples where the partitioning interfaces for sets can be used are striped partitioning of a matrix and simple partitioning of a graph. In striped partitioning of a matrix, a matrix is divided into groups of complete

rows or complete columns, the number of rows or columns being proportional to speeds of the processors. In simple partitioning of an unweighted graph, the set of vertices are partitioned into disjoint partitions such that the criterion (a) is satisfied. In simple partitioning of a weighted graph, the set of vertices are partitioned into disjoint partitions such that criterion (b) is satisfied.

5 Conclusion

The same approach is applied to classification of partitioning problems for matrices, graphs, and trees. More information on partitioning these mathematical objects and related API can be found in [6].

References

1. Arapov, D., Kalinov, A., Lastovetsky, A., Ledovskih, L: A Language Approach to High Performance Computing on Heterogeneous Networks. Parallel and Distributed Computing Practices 2(3), pp.87-96, 1999
2. Lastovetsky, A., Arapov, D., Kalinov, A., Ledovskih, I.: A Parallel Language and Its Programming System for Heterogeneous Networks. Concurrency: Practice and Experience 12(13), pp.1317-1343, 2000
3. Lastovetsky, A.: Adaptive Parallel Computing on Heterogeneous Networks with mpC. Parallel Computing 28(10), pp.1369-1407, 2002
4. Lastovetsky, A., Reddy,R.: HMPI: Towards a Message-Passing Library for Heterogeneous Networks of Computers. In Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), CD-ROM/Abstracts Proceedings, IEEE Computer Society 2003
5. Lastovetsky, A., Reddy, R.: Towards a Realistic Model of Parallel Computation on Networks of Heterogeneous Computers. Technical Report, University College Dublin, April 2003
6. Lastovetsky, A., Reddy, R.: Classification of Partitioning Problems for Networks of Heterogeneous Computers. Technical Report, University College Dublin, December 2003

Load-Balancing Iterative Computations on Heterogeneous Clusters with Shared Communication Links

Arnaud Legrand, Hélène Renard, Yves Robert, and Frédéric Vivien

LIP, UMR CNRS-INRIA-UCBL 5668, École normale supérieure de Lyon, France
{Arnaud.Legrand,Helene.Renard,Yves.Robert,Frederic.Vivien}@ens-lyon.fr

Abstract. We focus on mapping iterative algorithms onto heterogeneous clusters. The application data is partitioned over the processors, which are arranged along a virtual ring. At each iteration, independent calculations are carried out in parallel, and some communications take place between consecutive processors in the ring. The question is to determine how to slice the application data into chunks, and assign these chunks to the processors, so that the total execution time is minimized. A major difficulty is to embed a processor ring into a network that typically is not fully connected, so that some communication links have to be shared by several processor pairs. We establish a complexity result assessing the difficulty of this problem, and we design a practical heuristic that provides efficient mapping, routing, and data distribution schemes.

1 Introduction

We investigate the mapping of iterative algorithms onto heterogeneous clusters. Such algorithms typically operate on a large collection of application data, which is partitioned over the processors. At each iteration, some independent calculations are carried out in parallel, and then some communications take place. This scheme encompasses a broad spectrum of scientific computations, from mesh based solvers to signal processing, and image processing algorithms. An abstract view of the problem is the following: the iterative algorithm repeatedly operates on a rectangular matrix of data samples. This matrix is split into vertical slices that are allocated to the computing resources. At each step of the algorithm, the slices are updated locally, and then boundary information is exchanged between consecutive slices. This geometrical constraint advocates that processors be organized as a virtual ring. Then each processor only communicates twice, once with its predecessor in the ring, and once with its successor. There is no reason to restrict to a uni-dimensional partitioning of the data, and to map it onto a uni-dimensional ring of processors. But uni-dimensional partitionings are very natural for most applications, and we show that finding the optimal one is already very difficult.

The target architecture is a fully heterogeneous cluster, composed of different-speed processors that communicate through links of different bandwidths. On

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 930–937, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the architecture side, the problem is twofold: (i) select the processors that participate in the solution and decide for their ordering (which defines the ring); (ii) assign communication routes between each pair of consecutive processors in the ring. One major difficulty of this ring embedding process is that some of the communication routes will (most probably) have to share some physical communication links: indeed, the communication networks of heterogeneous clusters typically are far from being fully connected. If two or more routes share the same physical link, we have to decide which fraction of the link bandwidth is assigned to each route. Once the ring and the routing have been decided, there remains to determine the best partitioning of the application data. Clearly, the quality of the final solution depends on many application and architecture parameters.

Section 2, is devoted to the precise and formal specification of our optimization problem, denoted as SHAREDRING. We show that the associated decision problem is NP-complete. Then, section 3 deals with the design of polynomial-time heuristics to solve the SHAREDRING problem. We report some experimental data in Section 4. Finally, we state some concluding remarks in Section 5. Due to the lack of space, we refer the reader to [6] for a survey of related papers.

2 Framework

2.1 Modeling the Platform Graph

Computing costs. The target computing platform is modeled as a directed graph $G = (P,E)$. Each node P_i in the graph, $1 \leq i \leq |P| = p$, models a computing resource, and is weighted by its relative cycle-time w_i: P_i requires w_i time-steps to process a unit-size task. Of course the absolute value of the time-unit is application-dependent, what matters is the relative speed of one processor versus the other.

Communication costs. Graph edges represent communication links and are labeled with available bandwidths. If there is an oriented link $e \in E$ from P_i to P_j, b_e denotes the link bandwidth. It takes L/b_e time-units to transfer one message of size L from P_i to P_j using link e. When several messages share the link, each of them receives a fraction of the available bandwidth. The fractions of the bandwidth allocated to the messages can be freely determined by the user, except that the sum of all these fractions cannot exceed the total link bandwidth. The eXplicit Control Protocol XCP [5] does enable to implement a bandwidth allocation strategy that complies with our hypotheses.

Routing. We assume we can freely decide how to route messages between processors. Assume we route a message of size L from P_i to P_j, along a path composed of k edges $e_1, e_2, \ldots, e_k$. Along each edge e_m, the message is allocated a fraction f_m of the bandwidth b_{e_m}. The communication speed along the path is bounded by the link allocating the smallest bandwidth fraction: we need L/b time-units to route the message, where $b = \min_{1 \leq m \leq k} f_m$. If several messages simultaneously circulate on the network and happen to share links, the total bandwidth capacity of each link cannot be exceeded.

Application parameters: computations. W is the total size of the work to be performed at each step of the algorithm. Processor P_i performs a share $\alpha_i.W$, where $\alpha_i \geq 0$ and $\sum_{i=1}^{p} \alpha_i = 1$. We allow $\alpha_j = 0$, meaning that processor P_j do not participate: adding more processors induces more communications which can slow down the whole process, despite the increased cumulated speed.

Application parameters: communications in the ring. We arrange the participating processors along a ring. After updating its data slice, each active processor sends a message of fixed length H to its successor. To illustrate the relationship between W and H, we can view the original data matrix as a rectangle composed of W columns of height H, so that one single column is exchanged between consecutive processors in the ring.

Let $\text{succ}(i)$ and $\text{pred}(i)$ denote the successor and the predecessor of P_i in the virtual ring. There is a communication path $\mathcal{S}_i$ from P_i to $P_{\text{succ}(i)}$ in the network: let $s_{i,m}$ be the fraction of the bandwidth b_{e_m} of the physical link e_m that is allocated to the path $\mathcal{S}_i$. If a link e_r is not used in the path, then $s_{i,r} = 0$. Let $c_{i,\text{succ}(i)} = \frac{1}{\min_{e_m \in \mathcal{S}_i} s_{i,m}}$: P_i requires $H.c_{i,\text{succ}(i)}$ time-units to send its message of size H to its successor $P_{\text{succ}(i)}$. Similarly, we define the path $\mathcal{P}_i$ from P_i to $P_{\text{pred}(i)}$, the bandwidth fraction $p_{i,m}$ of e_m allocated to $\mathcal{P}_i$, and $c_{i,\text{pred}(i)} = \frac{1}{\min_{e_m \in \mathcal{P}_i} p_{i,m}}$.

Objective function. The total cost of one step in the iterative algorithm is the maximum, over all participating processors (whose set is denoted $\mathcal{P}$), of the time spent computing and communicating:

$$T_{\text{step}} = \max_{P_i \in \mathcal{P}} \left(\alpha_i.W.w_i + H.(c_{i,\text{pred}(i)} + c_{i,\text{succ}(i)}) \right).$$

In summary, the goal is to determine the best way to select q processors out of the p available, to assign them computational workloads, to arrange them along a ring, and to share the network bandwidth so that T_{step} is minimized.

2.2 The SharedRing Optimization Problem

Definition 1 (**SharedRing**(G,W,H)). *Given p processors P_i of cycle-times w_i and $|E|$ communication links e_m of bandwidth b_{e_m}, given the total workload W and the communication volume H at each step, minimize*

$$T_{step} = \min_{1 \leq q \leq p} \min_{\substack{\sigma \in \Theta_{q,p} \\ \sum_{i=1}^{q} \alpha_{\sigma(i)} = 1}} \max_{1 \leq i \leq q} \left(\alpha_{\sigma(i)}.W.w_{\sigma(i)} + H.(c_{\sigma(i),\sigma(i-1 \bmod q)} + c_{\sigma(i),\sigma(i+1 \bmod q)}) \right) \tag{1}$$

In Equation (1), $\Theta_{q,p}$ denotes the set of one-to-one functions $\sigma : [1..q] \to [1..p]$ which index the q selected processors that form the ring, for all candidate values of q between 1 and p. For each candidate ring represented by such a σ function, there are constraints hidden by the introduction of the quantities $c_{\sigma(i),\sigma(i-1 \bmod q)}$ and $c_{\sigma(i),\sigma(i+1 \bmod q)}$, which we gather now. There are $2q$ communicating paths: the path $\mathcal{S}_i$ from $P_{\sigma(i)}$ to its successor $P_{\text{succ}(\sigma(i))} = P_{\sigma(i+1 \bmod q)}$ and the path $\mathcal{P}_i$

from $P_{\sigma(i)}$ to its predecessor $P_{\text{pred}(\sigma(i))} = P_{\sigma(i-1 \bmod q)}$, for $1 \leq i \leq q$. For each link e_m in the interconnection network, let $s_{\sigma(i),m}$ (resp. $p_{\sigma(i),m}$) be the fraction of the bandwidth b_{e_m} that is allocated to the path $\mathcal{S}_{\sigma(i)}$ (resp. $\mathcal{P}_{\sigma(i)}$). We have the equations:

$$\begin{cases} 1 \leq i \leq q, \quad 1 \leq m \leq E, \quad s_{\sigma(i),m} \geq 0, \quad p_{\sigma(i),m} \geq 0, \quad \sum_{i=1}^{q} (s_{\sigma(i),m} + p_{\sigma(i),m}) \leq b_{e_m} \\ 1 \leq i \leq q, \quad c_{\sigma(i),\text{succ}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{S}_{\sigma(i)}} s_{\sigma(i),m}}, \quad c_{\sigma(i),\text{pred}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{P}_{\sigma(i)}} p_{\sigma(i),m}} \end{cases}$$

Since each communicating path $\mathcal{S}_{\sigma(i)}$ or $\mathcal{P}_{\sigma(i)}$ will typically involve a few edges, most of the quantities $s_{\sigma(i),m}$ and $p_{\sigma(i),m}$ will be zero. In fact, we have written $e_m \in \mathcal{S}_{\sigma(i)}$ if the edge e_m is actually used in the path $\mathcal{S}_{\sigma(i)}$, i.e. if $s_{i,m}$ is not zero (and similarly, $e_m \in \mathcal{P}_{\sigma(i)}$ if $p_{i,m}$ is not zero). Note that, when q and σ are known, the whole system of (in)equations is quadratic in the unknowns α_i, $s_{i,j}$, and $p_{i,j}$ (we explicit this system on an example in [6]).

From Equation (1), we see that the optimal solution involves all processors as soon as the ratio $\frac{W}{H}$ is large enough: then the impact of the communications becomes small in front of the cost of the computations, and the computations should be distributed to all resources. Even in that case, we have to decide how to arrange the processors along a ring, to construct the communicating paths, to assign bandwidths ratios and to allocate data chunks. Extracting the "best" ring seems to be a difficult combinatorial problem.

2.3 Complexity

The following result states the intrinsic difficulty of the SHAREDRING problem (see [6] for the proof):

Theorem 1. *The decision problem associated to the* SHAREDRING *optimization problem is NP-complete.*

3 Heuristics

We describe, in three steps, a polynomial-time heuristic to solve SHAREDRING: (i) the greedy algorithm used to construct a solution ring; (ii) the strategy used to assign bandwidth fractions during the construction; and (iii) a final refinement.

3.1 Ring Construction

We consider a solution ring involving q processors, numbered from P_1 to P_q. Ideally, all these processors should require the same amount of time to compute and communicate: otherwise, we would slightly decrease the computing load of the last processor and assign extra work to another one (we are implicitly using the "divisible load" framework [6]). Hence (see Figure 1) we have for all i (indices being taken modulo q):

$$T_{\text{step}} = \alpha_i . W . w_i + H . (c_{i,i-1} + c_{i,i+1}). \tag{2}$$

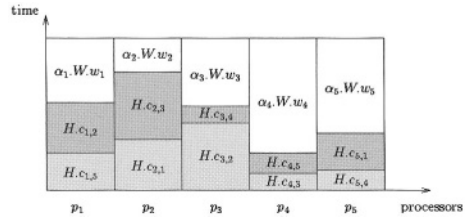


Fig. 1. Summary of computation and communication times with $q = 5$ processors.

Since $\sum_{i=1}^{q} \alpha_i = 1$, $\sum_{i=1}^{q} \frac{T_{\text{step}} - H.(c_{i,i-1}+c_{i,i+1})}{W.w_i} = 1$. With $w_{\text{cumul}} = \frac{1}{\sum_{i=1}^{q} \frac{1}{w_i}}$:

$$T_{\text{step}} = W.w_{\text{cumul}} \left(1 + \frac{H}{W} \sum_{i=1}^{q} \frac{c_{i,i-1} + c_{i,i+1}}{w_i} \right) \tag{3}$$

We use Equation (3) as a basis for a greedy algorithm which grows a solution ring iteratively, starting with the best pair of processors. Then, it iteratively includes a new node in the current solution ring. Assume we already have a ring of r processors. We search where to insert each remaining processor P_k in the current ring: for each pair of successive processors (P_i, P_j) in the ring, we compute the cost of inserting P_k between P_i and P_j. We retain the processor and pair that minimize the insertion cost. To compute the cost of inserting P_k between P_i and P_j, we resort to another heuristic to construct communicating paths and allocate bandwidth fractions (see Section 3.2) in order to compute the new costs $c_{k,j}$ (path from P_k to its successor P_j), $c_{j,k}$, $c_{k,i}$, and $c_{k,i}$. Once we have these costs, we can compute the new value of T_{step} as follows:

- We update w_{cumul} by adding the new processor P_k into the formula.
- In $\sum_{s=1}^{r} \frac{c_{\sigma(s),\sigma(s-1)} + c_{\sigma(s),\sigma(s+1)}}{w_{\sigma(s)}}$, we suppress the terms corresponding to the paths between P_i to P_j and we insert the new terms $\frac{c_{k,j}+c_{k,i}}{w_k}$, $\frac{c_{j,k}}{w_j}$ and $\frac{c_{i,k}}{w_i}$.

This step of the heuristic has a complexity proportional to $(p - r).r$ times the cost to compute four communicating paths. Finally, we grow the ring until we have p processors. We return the minimal value obtained for T_{step}. The total complexity is $\sum_{r=1}^{p} (p - r)rC = O(p^3)C$, where C is the cost of computing four paths in the network. Note that it is important to try all values of r, because T_{step} may not vary monotonically with r (for instance, see Figure 5).

3.2 Bandwidth Allocation

We now assume we have a r-processor ring, a pair (P_i, P_j) of successive processors in the ring, and a new processor P_k to be inserted between P_i and P_j. Together with the ring, we have built $2r$ communicating paths to which a fraction of the initial bandwidth has been allocated. To build the new four paths involving P_k, we use the graph $G = (P, E, b)$ where $b(e_m)$ is what has been left by the $2r$ paths

of the bandwidth of edge e_m. First we re-inject the bandwidths fractions used by the communication paths between P_i and P_j. Then to determine the four paths, from P_k to P_i and P_j and vice-versa:

- We independently compute four paths of maximal bandwidth, using a standard shortest path algorithm in G.
- If some paths happen to share some links, we use an analytical method to compute the bandwidth fractions minimizing Equation 3 to be allocated.

Then we can compute the new value of T_{step} as explained above, and derive the values of the α_i. Computing four paths in the network costs $C = O(p + |E|)$.

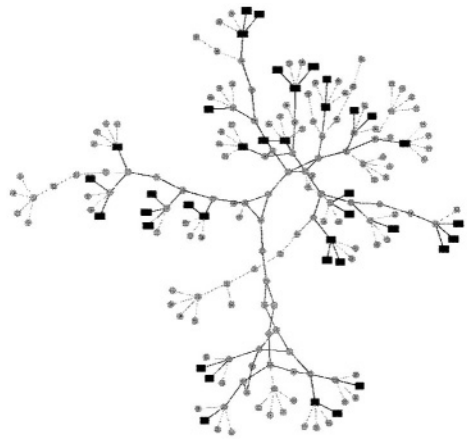


Fig. 2. Boxed nodes are computing nodes: there are 37 of them, connected through 47 routers, and 91 communication links.

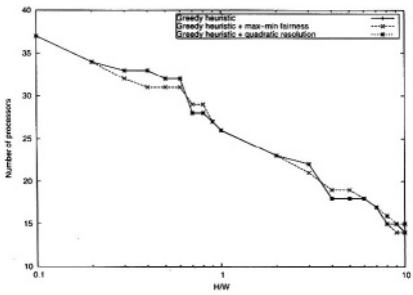


Fig. 3. Size of the optimal ring as a function of the ratio *H/W*.

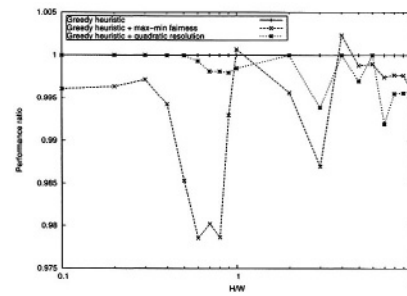


Fig. 4. Impact of the refinements on the quality of the solution.

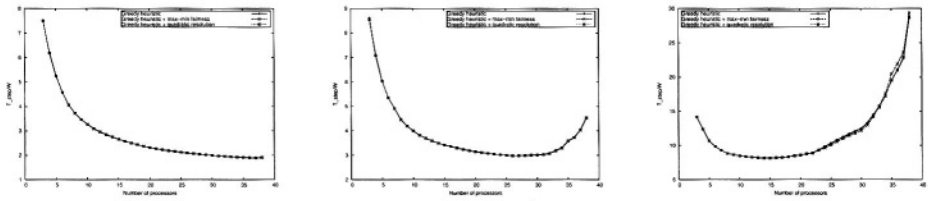


Fig. 5. Value of T_{step}/W as a function of the size of the solution ring, with a communication-to-computation ratio *H/W* equal from left to right to: 0.1, 1, and 10.

3.3 Refinements

Schematically, the heuristic greedily grows a ring by peeling off the bandwidths to insert new processors. To diminish the cost of the heuristic, we never recalculate the bandwidth fractions that have been previously assigned. When the heuristic ends, we have a q-processor ring, q workloads, $2q$ communicating paths, bandwidth fractions and communication costs for these paths, and a feasible value of T_{step}. As the heuristic could appear over-simplistic, we have implemented two variants aimed at refining its solution. The idea is to keep everything but the bandwidth fractions and workloads. Once we have selected the processor and the pair minimizing the insertion cost in the current ring, we perform the insertion and recompute all the bandwidth fractions and workloads. We can re-evaluate bandwidth fractions using a global approach (see [6] for details):

Method 1: Max-min fairness. We compute first the bandwidths fractions using the traditional bandwidth-sharing algorithm [1] maximizing the minimum bandwidth allocated to a path, then the α_i so as to equate all execution times (computations followed by communications), thereby minimizing T_{step}.

Method 2: Quadratic resolution. Once we have a ring and all the communicating paths, the program to minimize T_{step} is quadratic in the unknowns α_i, $s_{i,j}$ and $p_{i,j}$. We use the KINSOL library [7] to numerically solve it.

4 Experimental Results

4.1 Platform Description

We experimented with two platforms generated with the Tiers network generator [3]. Due to lack of space, and as the results are equivalent, we only report on the first platform. All results can be found in [6]. The Tiers generator produces graphs having three levels of hierarchy (LAN, MAN, and WAN). The platforms are generated by selecting about 30% of the LAN nodes (the boxed nodes in Figure 2) which are the computing nodes: the other nodes are simple routers. The processing powers of the computing nodes are randomly chosen in a list corresponding to the processing powers (evaluated using a LINPACK benchmark [2]) of a wide variety of machines. The link capacities are assigned, using the classification of the Tiers generator (LAN, MAN, and WAN), with values measured by `pathchar` [4] between machines scattered in France, USA, and Japan.

4.2 Results

Figure 3 plots the number of processors used in the solution ring. As expected, this number decreases as the ratio *H/W* increases: additional computational power does not pay off the communication overhead. Figure 5 presents the normalized execution time as a function of the size of the solution ring for various communication-to-computation ratios: the optimal size is reached with fewer processors as the ratio increases. Finally, we try to assess the usefulness of the two variants introduced to refine the heuristic (Figure 4). Surprisingly enough, the impact of both variants is not significant: the best gain is 3%. Thus the plain version of the heuristic turns out to be both low-cost and efficient.

5 Conclusion

The major limitation to programming heterogeneous platforms arises from the additional difficulty of balancing the load. Data and computations are not evenly distributed to processors. Minimizing communication overhead becomes a challenging task. In this paper, the major emphasis was towards a realistic modeling of concurrent communications in cluster networks. One major result is the NP-completeness of the SHAREDRING problem. Rather than the proof, the result itself is interesting, because it provides yet another evidence of the intrinsic difficulty of designing heterogeneous algorithms. But this negative result should not be over-emphasized. Indeed, another important contribution of this paper is the design of an efficient heuristic, that provides a pragmatic guidance to the designer of iterative scientific computations. Implementing such computations on commodity clusters made up of several heterogeneous resources is a promising alternative to using costly supercomputers.

References

1. D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, 1987.
2. R. P. Brent. The LINPACK Benchmark on the AP1000: Preliminary Report. In *CAP Workshop 91.* Australian National University, 1991. Website http://www.netlib.org/linpack/.
3. Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine,* 35(6): 160–163, June 1997.
4. Allen B. Downey. Using pathchar to estimate internet link characteristics. In *Measurement and Modeling of Computer Systems,* pages 222–223, 1999.
5. D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM* 2002, pages 89–102. ACM Press, 2002.
6. A. Legrand, H. Renard, Y. Robert, and F. Vivien. Load-balancing iterative computations in heterogeneous clusters with shared communication links. Research Report RR-2003-23, LIP, ENS Lyon, France, April 2003.
7. A.G. Taylor and A.C. Hindmarsh. User documentation for KINSOL. Technical Report UCRL-ID-131185, Lawrence Livermore National Laboratory, July 1998.

Large Scale Peer to Peer Performance Evaluations, with Gauss-Jordan Method as an Example

Serge G. Petiton and Lamine M. Aouad

Laboratoire d'Informatique Fondamentale de Lille, Ecole Polytechnique Universitaire de Lille, Université des Sciences et Technologies de Lille, France
{petiton, aouad}@lifl.fr

Abstract. This paper presents a large scale block-based Gauss-Jordan algorithm to invert very large dense matrices. This version proposes to exploit peer-to-peer (P2P) platforms with increasingly large sets of distributed heterogeneous resources. We assume that we have access to a scheduler that proposes strategies allowing data nailing and data migration anticipation heuristics. Under given hypotheses, we present the up bounds of theoretical evaluation results, using different P2P platforms, with sufficient number of peers interconnected by different networks. Nevertheless, we discuss that, in these cases, the classical evaluation model is not well-adapted to this P2P computing paradigm for large scale scientific applications.

1 Introduction

The availability of powerful personal computers or workstations and high-speed network technologies has changed the way of using computing resources in the last decade. A number of scientific applications that have traditionally performed on supercomputers or on NOW's with traditional tools, such as MPI or PVM, would be running on a variety of heterogeneous resources geographically distributed.

The architecture model we considered is a peer-to-peer platform such as:

- Several thousands of computers with heterogeneous power and various memory sizes,
- Different interconnection networks, Web-based and connected to high-speed networks which provide reliable multicast communications [5].

We assume that it exists an interface which enables the user to program and execute applications, and that a language is proposed to describe task graphs and data communications.

This paper proposes a large scale peer-to-peer block based Gauss-Jordan algorithm. A parallel adaptive implementation of this method on a metacomputing middleware is presented in [1] and a classical large granularity parallel version has been proposes in [2].

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 938–945, 2004.
© Springer-Verlag Berlin Heidelberg 2004

This paper is organized as follows: section 2 presents a brief introduction to the peer-to-peer systems and section 3 describes our hypotheses. In section 4 we describe the block-based Gauss-Jordan algorithm and propose a parallel and distributed version well-adapted to targeted platforms. Section 5 presents performance evaluations for matrix sizes from 0.27×10^6 to 0.9×10^6. The number of peers is from 8100 to 90000 and there are connected by different targeted networks. Finally, we conclude and present research perspectives on section 6.

2 State of the Art

Peer-to-Peer and Grid computing platforms enable the sharing, selection, and aggregation of a wide variety of heterogeneous resources geographically distributed, such as computers and data sources, to solve large-scale problems in science, engineering and business, which cannot be effectively dealt using the current generation of supercomputers or which are less expensive or accessible with this approach. In a peer-to-peer architecture, computers that have traditionally been used alone as clients communicate directly among themselves and can act as both clients and servers. It takes advantage of existing computing power and networking connectivity, allowing users to leverage their collective power to benefit other users. However, parallel and distributed application developments and resource managements in these environments are a new and complex undertaking. In scientific computation, for example, the validity of calculations, the numerical stability, the choices of methods and software's are depends of properties of each peer and its software and hardware environments; which are known only at run time and are indeterminists.

The vast majority of applications deployed today are designed around the centralized or client/server models. The great failure of these models lies in their inability to recognize, let alone exploit, the growing power and information resources at the edges of the network. With P2P computing, each participating computer, referred to as peer, functions as a client with a layer of server functionality. A peer can initiate requests, and it can respond to requests from other peers in the network. The ability to make direct exchanges with other users liberates P2P users from the traditional dependence on central servers. Users have a higher degree of autonomy and control over the services they utilize. So far, the two major classes of P2P applications are the direct Exchange of Services and the distributed Information Infrastructure.

3 Hypotheses

The algorithm introduced in this paper makes some hypotheses about the scheduling strategies and the communication services. First, we suppose to have access to a P2P platform with a large number of computers. Each of them can compute, receive and send data, and take part in the organization of the system. We also assume that we have access to a middleware that provides scheduling strategies with smart "owner computes", which allow data nailing, and migration

anticipation heuristics. Second, our application requires that blocs are delivered from a source to a set of receivers, so we must have a reliable multicast communications between peers.

We consider an average throughput multicast data flow between peers in different networks (slow and fast Internet, and very high performance connection networks). In [5], the average throughput reached is 3.2MB/s in cluster of PC with Ethernet LAN (100Mb/s) using ALC/MPICH-V[3] and data compression. In this case, sending data among peers allows a low bandwidth consumption but need higher resources usage. In WAN, communications between peers and the muticast network can become the bottleneck (unless the bandwidth of the network interface increases). Furthermore, the purpose of our scheduling strategies is to reduce communication latency and increases effective bandwidth. We also assume that other scheduling criteria, such as the execution model, the type of tasks, the power, the memory size, and the proposed scientific libraries of each peer are considered.

Nevertheless, resources failure is the rule rather the exception in these platforms. We must take this into account to use the available resources and services efficiently. To accomplish this, reliable tools and robust environments that monitor resources and applications need to be deployed. Our work presented here is a tentative to extrapolate the use of these ones to predict a new programming paradigm and to propose a performance evaluation for class of scientific computing such as dense linear algebra.

4 Block-Based Gauss-Jordan Method

As a linear algebra method example, we study a block version of the Gauss-Jordan method. Let A and B be two dense matrices of dimension N, and let B be the inverted matrix of A, i.e. $AB = BA = I$. Let A and B be partitioned into a matrix of $p \times p$ blocks of dimension n. The method computes $PAB = PI$, where P is a matrix associated to the method such as $PA = I$ and, then, $B = PI$. At the beginning of the method, we start with the matrix A stored in an array and with the matrix identity stored in another one which will store the inverted matrix PI at the end of the computation. These matrices are distributed on several peers.

Each of the p steps of the algorithm has 3 parts. The method is the block version of classical Gauss-Jordan one and doesn't have a global pivot strategy. The pivoting is done at the block level. Let $C_{i,j}^k$ be the block (i, j) of the matrix C modified at the step k. At each step $k(k = 1...p)$, the first part is to invert the pivot block $A_{k,k}^{k-1}$. The second one is to compute the blocks of the row k of the matrix A and B with indexes above k for A and below k for B, and the blocks of the column k of the matrix B except $B_{k,k}^k$. The last part is to calculate the blocks of all the columns of the matrix A and B with indexes above k for A and below k for B, which was not already modified at this fixed step. Figure 1 illustrates the dependencies between block calculations at the steps $k = 1$ and $k = 2$. It shows the intra-step dependencies. Then, at a fixed step, each block which have to

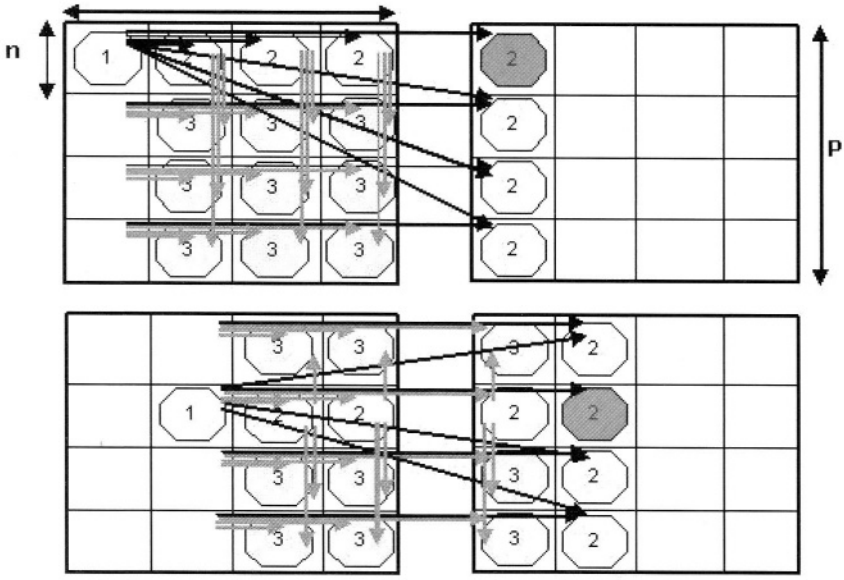


Fig. 1. The intra-step data dependencies $k = 1$ and $k = 2$

be modified need the block-parents on the dependencies graph as an input. The matrices A and B are partitioned into blocks which are represented by numbered squares. Thus, these numbers designate the blocks update order. Task farming scheduling [4] is possible and well-adapted if we assume that we have as many peers as the maximum number of tasks necessary to compute each part.

Nevertheless, on large scale peer to peer platform we often have peers which leave the system for many reasons and, then, this task farming scheduling will generate large delays because we will have to wait the completion of all tasks of each part before to start other computation. The fault tolerant system management of these failures will increase the global time of each front of this task farming scheduling. Another reason for large variation of computation times between several tasks with a same fork-join part can be theirs heterogeneous hardware (frequencies, caches, etc), software (compilers, libraries, etc) or networks; even if any peer doesn't leave the platform. Then, to optimize this global computation time, we have to propose another dependencies graph and scheduling strategy. If we remark that we are not always obliged to wait the completion of all tasks of each part of one step to compute children-tasks on the graph, we can schedule each task as soon as the necessary computations which generate its input blocks are done ; cf. [1,2] for more details. Then, it is possible to have in parallel different tasks from different parts or steps of the algorithm on various peers. The figure 2 illustrates some of these inter-steps dependencies between the second and the third steps ($k = 2$ and $k = 3$); when the computation of blocks (1,3), (4,4) and (2,5) of the second step are not completed. It shows that the following step can be started as soon as calculations at the preceding step

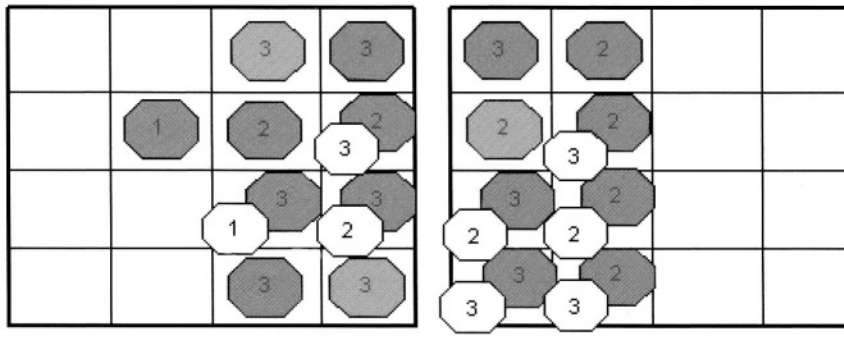


Fig. 2. The inter-steps data dependencies $k = 2$ to 3

are partly finished. Parts of calculation of the third step are done whereas 3 blocks are not yet updated at the step 2. For each computing task, we need 1 to 3 blocks of dimension n. Then, the square of the maximum dimension of each block must be lower than the third of the central memory size of a peer. We can also remark that $(p-1)^2$ peers at least can be necessary for calculation.

The intra and inter step task dependencies graph is shown in the figure 3. Each arrow represents n^2 data (64bits) migration from one task to another. The complexity of each of the three operations at each step (element Gauss-Jordan, matrix product and matrix triadic) is approximately equal to $2n^3$. Then, each task computes $2n^3$ floating point operations and migrates from $2n^2$ to $4n^2$ data, without data migration optimizations. The total complexity is approximately equal to $2N^3$.

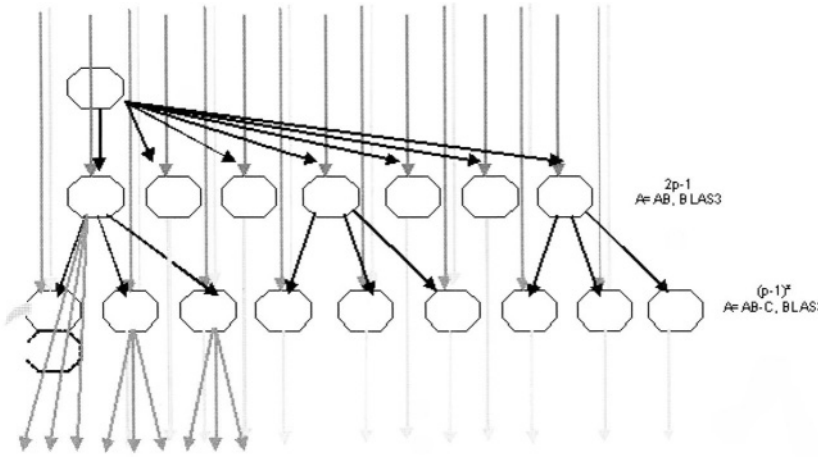


Fig. 3. Data dependences graph of one fixed step

To optimize the global time, a crucial issue is thus to optimize the data migrations. Let $A_{(i,j+p)}$ be equal to $B_{(i,j)}$. Then, for example, according to the data nailing heuristic, at the second step of the method, the block update in the second column of B must be done in the peers storing the blocks of the column 2 of A; i.e. the blocks $A^2_{i,2+p}; i = 1, p$ must be updated in the peers having calculated $A^1_{i,2}; i = 1, p$, and the diffusion of $A^1_{i,k}; i = 1, p (i \neq k)$ on the peers having calculated $A^1_{i,j}; j = k+1, k+p-1$ must be made as soon as possible. In a volatile environment this optimization implies a redundant process managed by the scheduler and the middleware.

The large scale peer to peer algorithm, using the assumptions described above, is done in p steps in parallel with respect of intra and inter steps dependencies. This algorithm is well adapted to the large P2P platforms with volatile distributed resources and is not deterministic. indeed, its behavior cannot be predicted from the input, especially because of volatility. This problem can be solved by redundant computation, as described above. Each work should be processed multiple times in order to envisage the peers which leave the platform. This also makes it possible to detect and discard results from faulty processors and from malicious users.

5 Performance Theoretical Evaluation Results

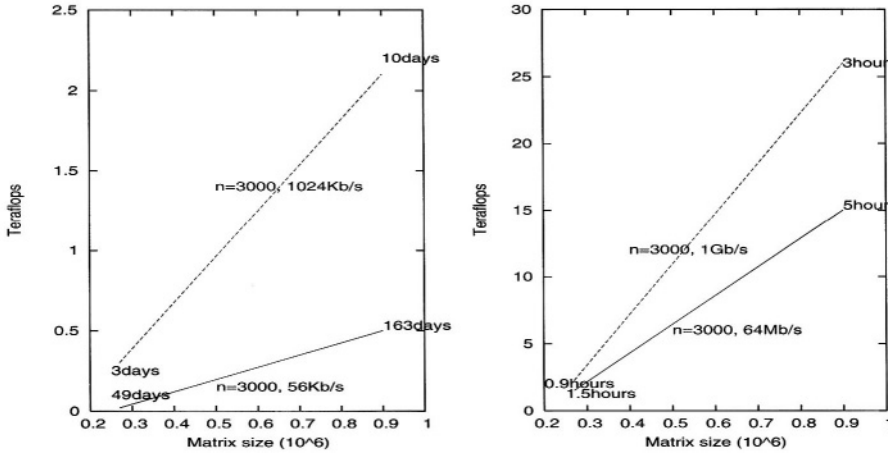
To have a deterministic algorithm and be able to evaluate the global time, we assume that the peers never leave the system for any reason. Then, we can remark that the communications cost for a step k is: 1) two times the diffusion of $p-1$ blocks, in parallel, towards $p-1$ ($A_{i,k}; i \neq k$ and $A_{k,j}; j \neq k$), 2) the diffusion of 1 block towards $2(p-1)$, let $A_{k,k}$ be this block. All the other communications can be overlapped by task computations, anticipated, or virtualized or other communications. Then, we can already conclude that the global time formula t(N), under given hypotheses, is equal to:

$$\mathrm{t(N)} = \mathrm{p}\ [2\mathrm{t}(n^2 \text{ data diffusion}) + 3\mathrm{t}(2n^3 \text{ operations})], \text{ with } N = p \times n$$

Assuming that each peer has no any failures is a strong hypothesis as we discuss above. Nevertheless, we are obliged to accept this assumption because it is too difficult to evaluate performances otherwise. Then, we will obtain an up bound of possible performances. If we remark that experimentations on fault tolerant systems can show performances decrease by a large factor, we can also expected that on close future this factor will be optimized.

To evaluate theoretical performance of this block Gauss-Jordan method, we estimate the above t(N) formula under the following assumptions:

- The block size n is equal to 3000,
- The number of peers is from 8100 ($N = 0.27 \times 10^6$) to 90000 ($N = 0.9 \times 10^6$),
- The different targeted networks are :
  - Internet-like with 56Kbits/s,
  - Faster Internet-like, up to 1024kbits/s,



- Network with 64Mbits/s,
- Very high connection network, up to 1Gbits/s.

– Memory sizes are approximately 32Mwords,
– The average sustained power of peers for basic linear algebra is approximately 500 Megaflops.

Results are presented on previous graphs. We first remark that the efficiency depends, as expected, on the links speeds. The absolute efficiency can reach 30% for matrices on a platform with, up to 90000 peers interconnected with a 64Mbits/s network. Moreover, the efficiency obtained with the slower Internet connected platform is very low (smaller than 0.5). That means that the major part of the total execution time is spent in slow communications.

These results show also that we can invert a matrix as large as 0.9×10^6 in a few hours and obtained several Teraflops. The results with Internet are more modest, 163 days with a platform interconnected by 56Kbits/s, and 10 days for the faster targeted Internet. The obtained results show that the communications management and their speeds are crucial points for performances evaluation for such platforms.

Nevertheless, the classical evaluation model is not well-adapted in these cases. The fact to be able to compute a so large problem with only cheap computers on Internet is the more important results in this case. Thus, as soon as P2P middleware, language and tools will be available these new computing facilities will be possible because platforms, such as Internet, already exists.

6 Discussion and Conclusion

We first have to remark that, on one hand these results are interpreted as up bound of expected performance, as explained above in this paper, and on another hand, it is just a first evaluation. Performances can be evaluated with respect to

many other parameters values. For example, if we assume that the average central memory size of peers is larger, we can choose a larger block size n and, then, reduce the number of peers for a fixed matrix size. In this case we also increase the data to migrate between peers and the number of arithmetic operations of each task. Then, tradeoffs will have to be proposed. All these evaluations are under study.

We have then presented performance simulations for a parallel version of block-based Gauss-Jordan algorithm that exploits the intra and inter-steps parallelism on a large scale peer-to-peer systems. We have to use scheduling strategies with data migration optimization techniques. With this policy, we optimize the data migrations in order to minimize communications time. However, this time is strongly related to the network interconnections. This statement highlights the interest of communications beyond their optimization and the computing time which is relatively short. The algorithms must be evaluated according to their communications not according to their calculations. Nevertheless, that allows to make more calculations without really penalizing the performances; to increase stability and to validate calculations, for example. Further work will consider the possible local numerical stability problems, such as singular block matrices to be invert at a fixed step.We can conclude that large scale peer to peer performance evaluation show that we can reach good theoretical throughputs, using heterogeneous computers. Nevertheless, we did strong assumptions on the future of middleware, scheduler and communication networks. It is new programming paradigm for large scientific applications that still need to be validated on real platforms and systems.

Acknowledgments. This research was supported by ACI-GRID grant and as part of the "Grand Large" INRIA project.

References

1. S. Petiton, N. Melab, E.-G. Talbi. *A Parallel Adaptive Gauss-Jordan Algorithm,* The journal of supercomputing, 17, 167-185, 2000.
2. S. Petiton. *Parallelisation on an MIMD computer with real-time scheduler, Gauss-Jordan example.* In M. H. Wright, ed., Aspects of computation on Asynchronous Parallel Processors. Elsevier Science, IFIP, 1989.
3. F. Cappello, S. Djilali, G. Fedak, C. Germain, G. Bosilca, A. Bouteiller, T. Herault, P. Lemarinier, O. Lodygensky, F. Magnette, V. Neri, A. Selikhov. *MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes.* SuperComputing 2002, Baltimore USA, November 2002.
4. H. Casanova, M. H. Kim, J. S. Plank, J. J. Dongarra. *Adaptive Scheduling for Task Farming with Grid Middleware.* The International Journal of High Performance Computing Applications, 1999.
5. B. Hudzia. *Use of multicast in P2P Network thought integration in MPICH-V2.* Internship report, Pierre and Marie currie university, LIP6, September 2003.

Anticipating Performance Information of Newly Portable Computers on the WLAN for Load Balancing*

David Sánchez, Elsa M. Macías, and Álvaro Suárez

Grupo de Arquitectura y Concurrencia (GAC)
Department of Ingeniería Telemática
University of Las Palmas de Gran Canaria
Campus Universitario de Tafira, 35017 Las Palmas de Gran Canaria, Spain
{dsanchez, emacias, asuarez}@dit.ulpgc.es

Abstract. Parallel and distributed computing over heterogeneous networks is a hot topic nowadays. We have used a combination of a Local Area Network (LAN) and a Wireless LAN (WLAN) to implement Master-Slave parallel distributed programs which exhibit dependences among iterations and parallel calculation inside an iteration. In this environment it is necessary to consider the unbalance communication bandwidths and computing power of fixed and portable computers to avoid long idle times. To solve this problem, in our previous work, we developed an efficient load balancing strategy considering that no new processes are spawned in new portable computers at run time. Due to it is not realistic, in this paper we present a new method for balancing the parallel calculations when parallel processes are spawned on portable computers which uses performance information supplied by a framework based on the standard Simple Network Management Protocol (SNMP). In this way, idle times are minimized and better execution times are obtained.

1 Introduction

In the last decade fixed networks of computers constitute the lowest cost as well as the most available parallel computer. However, the proliferation of high performance portable computers and the recent advances in wireless technologies allow combining WLAN with traditional LAN for parallel and distributed computing [1][2].

We use a computing environment formed by fixed and portable computers to implement Master-Slave applications that exhibit strict data dependences among iterations and parallel calculations inside each iteration. The intrinsic heterogeneity of this natural but currently used computing environment (different processing power and communication bandwidths) makes that the efficient

* Research partially supported by Spanish CICYT under Contract: TIC2001-0956-C04-03, by ULPGC under Contract: UNI 17/2002 and by the Fundación Canaria Universitaria de Las Palmas, Lopesan, S.A. and Unelco, S.A.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 946–953, 2004.
© Springer-Verlag Berlin Heidelberg 2004

execution of the mentioned applications be a very difficult task (if no care is taken long idle times will be obtained).

In general terms load balancing is a hot topic in heterogeneous computing [3] and it has made that some authors have studied efficient techniques to minimize idle time in the faster processors during the execution of parallel programs [4] [5] [6]. An important fact is that these works consider applications running in a wired network of workstations (homogeneous communication bandwidth) and in a fixed number of processes during all the execution time. In [7] we proposed a simple but efficient strategy for load balancing considering a combination of a LAN and a WLAN of fixed and portable computers (heterogeneous communication bandwidth) obtaining excellent results when the number of processes during all the execution time is fixed.

But in a WLAN it is very probable that portable computers change their physical location entering and going out of coverage. This fact introduces a new challenge in the design of heterogeneous parallel applications in local area networks and in particular in the design of load balancing techniques because the number of parallel processes change at run time naturally and with high probability. Clearly, if this variation of processes is not well managed idle time and poor execution times are obtained. Therefore, it is important to design a load balancing technique that can anticipate the amount of data to be distributed to new processes that enter to cooperate in the parallel application.

In this paper, we present a new method to estimate the amount of data to be sent to newly spawned processes on the WLAN computers. We consider a new metric based on some parameters related with the current performance of computers. In order to collect these parameters we have developed a non-intrusive framework based on the standard SNMP [8]. The use of this protocol is appropriated due to its low overhead both in calculations and in communication.

The rest of the paper is organized as follows. In section 2 we briefly describe background but necessary information to understand the new method objective. In section 3 we present the metric of the new method. The developed SNMP framework is explained in section 4. Next, in section 5 we show experimental results. Finally, we sum up the conclusions and we present the future work.

2 System Architecture and Background Information

Figure 1.a shows a graph of the combination of the LAN-WLAN. We consider parallel applications in which the master process distributes (in each iteration) a particular amount of data to each slave process (figure 1.b) that is directly correlated with the calculation performance of the slave processors. When each slave process finishes its calculation it will send results to the master process. The master process must receive the results of all slave processes before sending them the new data (next iteration).

We implement this kind of applications using our LAMGAC middleware [2] based on Message Passing Interface (MPI-2) to manage the dynamic expansion of slave processes on the portable computers that enter and go out of coverage at run time (figure 1.c). We control the variation of the number of portable computers in each iteration.

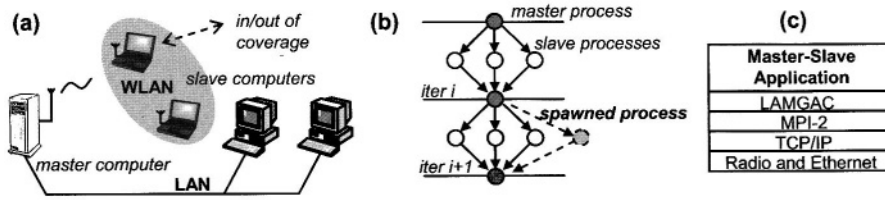


Fig. 1. a) LAN and WLAN combination network of computers, b) Master/Slave application dependences structure, c) Software architecture

Our previous load balancing method [7] minimizes the idle time in the slave processors and therefore it reduces the execution time of the overall program. Basically, first it measures in one particular iteration (for example iteration i) the calculation and communication time spent for each slave process, and then it calculates a new amount of data to be distributed in the next iteration (i+1) as a function of these values.

While this method works well when no newly slave process is spawned, it must be improved to consider this fact. That is, when a newly portable computer enters in coverage for cooperating as a slave processor, a new slave process is spawned on it. At this time the master can not know the optimum amount of data to send to it because it didn't work in the previous iteration. To anticipate this information is the objective of our novel load balance method because it can prevent certain idle time in the slave computers improving in this way the overall execution time.

3 The Metric of the Novel Load Balancing Method

In order to avoid that some computers are in an idle state during a long time, the execution time of each slave process in every iteration must be similar. The execution time of the process i during the iteration j can be expressed as:

$$t_exe_j(p_i) = t_comm_j(p_i) + t_calc_j(p_i) + t_idle_j(p_i) \quad (1)$$

Where the communication time $(t_comm_j(p_i))$ is the time elapsed to send data from the master process to the process i and send results from slave process i to the master process. Calculation time $(t_calc_j(p_i))$ is the time spent to carry out the calculations. During the idle time $(t_idle_j(p_i))$ neither calculation nor communication is done.

Communication and calculation time can be estimated in function of parameters related with the performance of networks and computers. Then, the communication time can be expressed as follows:

$$t_comm(p_i) = 2 \times t_{lat}(p_i) + \frac{n_unit \times (sizeof(data_u) + sizeof(result_u))}{B} \quad (2)$$

where:

- $t_{lat}(p_i)$ is the network latency between the master and the slave computer where the process i is running. For simplicity, we assume that this value is constant and equal for both communication directions.
- n_unit is the number of data units distributed to the slave process i. One data unit is the minimum amount of data to obtain one single result.
- $sizeof(data_u)$ is the size of one data unit.
- $sizeof(result_u)$ is the size of one single result.
- B is nominal data rate of the interface of slave computer. We assume that nominal data rate of master computer is bigger or equal to B.

On the other hand, the calculation time can be expressed as follows:

$$t_calc(p_i) = \frac{n_unit \times mips_calc}{mips_cpu}[\times \frac{load+1}{cpus}] \tag{3}$$

where:

- *mips_calc* is the number of instructions spent in the calculations in order to obtain one single result. We assume it as a constant.
- *mips_cpu* is the mips of the processor (processing power).
- *cpus* is the number of processors in the slave computer.
- *load* is the average number of jobs in the run queue of the system.

The relationship among brackets is applied if the expression *load* +1 is equal or bigger than the number of processors of the computer. This relationship is necessary because we executed the applications in a non-dedicated environment.

If the execution time in a iteration of newly spawned process i in a portable computer has to be similar to rest of the processes, then, the number of data units to send it such that the idle time is minimized is given by the metric:

$$n_unit_j(p_i) = \frac{\mu_texe_{j-1} - 2 \times t_{lat}(p_i)}{\frac{mips_calc}{mips_cpu}[\times \frac{load+1}{cpus}] + \frac{sizeof(data_u)+sizeof(result_u)}{B}} \tag{4}$$

Where μ_texe_{j-1} is the arithmetic average of the execution times of all slave processes in the previous iteration. This metric can be also applied in the first iteration of the parallel program, in this case, μ_texe_{j-1} is replaced by a fixed execution time for all processes.

In order to apply this metric, the parameters *mips_cpu, cpus, load,* t_{lat} and B must be collected from slave computers because they depend of the physical characteristics and of the current performance. To collect this information we used a framework based on SNMP which is described in the next section. The parameter *mips_calc* depend of the calculations carried out in the parallel program, and therefore, they have to be supplied to the application.

4 Using SNMP for Collecting Performance Information

SNMP is the protocol most widely used for managing network devices [8]. The core of this protocol is a simple set of operations that gives administrators the ability to change or monitor the state of SNMP based devices.

SNMP architecture is based on two kinds of entities: managers and agents. A manager handles management tasks for a network, being responsible for polling and receiving traps from agents in the network. A poll is the act of querying an agent for some information. A trap or notification message is a way for the agent to tell the manager that some event has happened. The second entity, the agent, runs on the managed network devices. The agent provides management information to the manager by keeping track of various operational aspects of the device. Also, when the agent notices that some event has occurred, it sends a trap to the manager. The agent manages the Management Information Base (MIB) which can be thought of as data-base of managed aspects that the agent tracks. The standard MIB-II defines variables to provide general information about system, TCP/IP, interface statistics, and so on.

4.1 SNMP Framework with Parallel Programs

Figure 2 shows the software architecture we implemented to collect information for being able to apply the proposed metrica in our Master/Slave parallel programs. In each slave computer runs an extended agent termed as Slave Information Collection Agent (SICA). SICA monitors some parameters and sends a notification message when some significant event related with the performance of the computer occurs.

In the master computer a manager named Traps Reception Manager (TRM) is in charge of decoding the received traps. Besides, a library that implements the metrics developed in this paper and in [7] is used for the master process to know the adequate amount of data to distribute to each slave process.

4.2 Slave Information Collection Agent

In order to implement SICA we have extended the capabilities of the standard SNMP agent. For that, we have designed an additional database named LBGAC-MIB to store the parameters to be applied in the metric proposed, and we have also built several functions to monitor the parameters and notify the traps.

Next, each one of the added parameters is described. All parameters are integer values, except the last two that are float values.

- *lbCpus.* Number of processors in the slave computer.
- *lbBogomips.* Mips of the processor. It is a Linux measurement.
- *lbLamdNet.* Type of communication link (wired or wireless).
- *lbLamdSpeed.* Nominal data rate in bps of the interface.
- *lbLatency.* Network latency between the master and the slave computer.
- *lbLoad.* Average number of jobs in the run queue in the last minute.

The *lbCpus* and *lbBogomips* parameters depend of physical characteristics of each computer and these values never change, therefore they are calculated once when the SICA starts. On the other hand, the parameters *lbLamdNet, lbLamdSpeed* and *lbLatency* are updated when the *lamd* daemon from LAM/MPI distribution is booted in the computer. Finally, the parameter *lbLoad* depend of

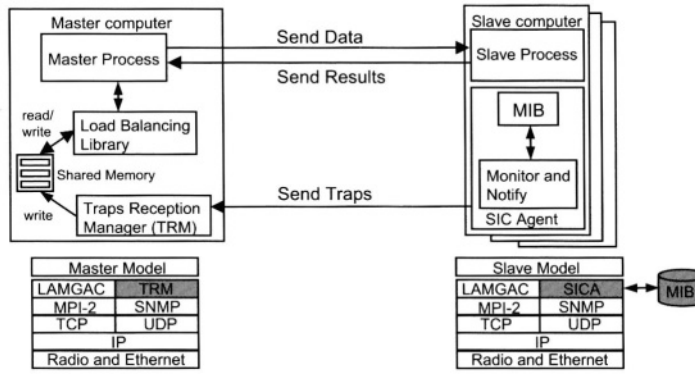


Fig. 2. SNMP framework with parallel programs

current number of processes running in the system, therefore, it needs to be monitored periodically.

Regarding the traps sent to the TRM, table 1 describes in details the two types of SNMP notifications. Parameters concerning to every event are enclosed to the notification message. In this way, parameters are communicated in a non-intrusive manner, and therefore the manager has not to carry out a poll to query its values.

4.3 Traps Reception Manager

The master process in the parallel application runs TRM when it starts. TRM is in charge of processing the received traps. The task carried out to process the notification messages is very light computationally. Simply, it extracts the enclosed parameters, and then, it stores its values in shared variables together with the trap type and the computer identifier that sent the trap, so that the master process via the library for load balancing can access to them. This module uses call-back mechanisms to collect traps from SICA. In this way, it only competes for resource usage with the parallel application when a trap message is received.

Table 1. Description of the notifications sent from SICA

Notification	Description	Enclosed Parameters
lbnAppStart	Slave process has started to run (when the parallel application starts or when a new process is spawned in a computer)	lbCpus, lbBogomips, lbLamdSpeed, lbLoad, lbLamdNet, lbLatency
lbnLoad	Average number of jobs in the run queue in the last minute has varied in a multiple of one	lbLoad

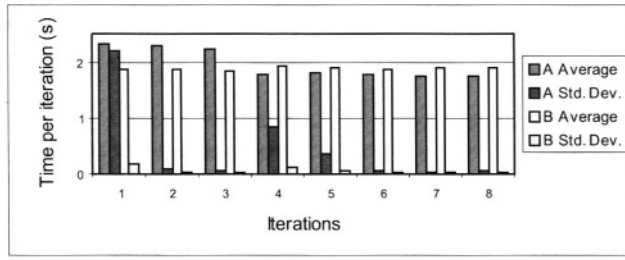


Fig. 3. Average execution time and standard deviation

5 Experimental Results

In this section we show the results obtained with the Hw/Sw Codesign Tool [9] applied to a voice recognition system. This tool has a parameter estimation phase where for a given VHDL input specification Hw/Sw parameters estimations are obtained (execution time, power consumption and area). In this problem, the data distribution consists of sending to each slave process a number of combinations of Hw/Sw resources for implementing the system. The data packet size of this information is the size of two integer values. For each combination, each slave process estimates the above parameters. The results obtained are sent to the master process. The data packet size of results is the multiplication of the number of combinations by the size of three float values. When the master process receives the results of all slaves, it calculates a new data distribution. These steps are repeated until there are no more combinations.

Table 2. Computing resources characteristics

Processor / Memory Size	Network Card (Mbps)	Network Latency (ms)
PIII 666Mhz/128 MB (master)	100 / 2	
PIII 1 Ghz/256 MB	100	0.069
Celeron 666 Mhz/128 MB	10	0.24
P 200Mhz MMX/64MB	10	0.25
PIV 2.4Ghz/512MB	2	4.48
PII 300Mhz/64MB	2	4.50

We have obtained experiments results in an IEEE 802.3 and IEEE 802.11 computers network with the computer specifications of table 2 running under Linux operating system, without and with the new metric using the SNMP framework (values labeled as A and B in fig. 3, respectively). Both experiments were run with the first four resources of the table 2, and in the iteration 4 and 5 of the simulation we have added the last two computers, respectively. A slave process was spawned in each one. It was assumed no wireless channel failures.

In B, the new method was applied to the new processes in the iterations 1, 4 and 5, because a trap from each computer was received. In the rest of iterations and for all processes (in both experiments) was applied the method presented in

[7]. Figure 3 shows per iteration the average execution time of the slave processes and the standard deviation for both experiments. As it can be observed, in the first iteration of A there is a high standard deviation due to the load unbalance. However, using the new method, the standard deviation is strongly reduced. The same situation occurs in the iterations 4 and 5, due to the spawned processes.

The accumulated execution time of the parallel application in the iteration 8 is about 23 seconds for A, and 19 seconds for B. Besides, the workload completed by the experiment A in that iteration is 2140 per 2409 combinations calculated by B. It represents a 12% more of work carries out in approximately four second less of simulation. Therefore, we demonstrate that with this new method the idle time is minimized in the newly spawned slave processes.

6 Conclusions and Future Work

In this paper, we presented a new method for load balancing that obtains excellent results in an environment where the number of slave processes can vary at run time. This method use a non-intrusive SNMP framework that is used to anticipate the current performance of the newly portable computers in coverage.

In our ongoing work we are planning to extend the LAMGAC middleware so that the programmer can manage the load balancing in a simple way. Other important facts in WLAN are to detect disconnections of wireless computers due to channel failures and low battery. These facts can produce heavy load unbalance, and in order to consider it we will extend our SNMP framework.

References

1. Cheng, L. Wanchoo, A., Marsic, I.: Hybrid Cluster Computing with Mobile Objects. 4^{th} IEEE Conference on High Perf. Comp. Beijin, China (2000) 909–914
2. Macías, E., Suárez, A.: Solving Engineering Applications with LAMGAC over MPI-2. 9^{th} EuroPVM/MPI. Linz, Austria. LNCS 2474. Springer Verlag (2002) 130–137
3. Braun, T.D., Siegel, H.J., Maciejewski, A.A.: Heterogeneous Computing: Goals, Methods, and Open Problems. Parallel and Distributed Processing, Techniques and Applications. Las Vegas, USA Vol. I (2001) 7–18
4. Zaki, M., Li, W., Parthasharathy, S.: Customized Dynamic Load Balancing for a Network of Workstations. 5^{th} High Perf. Distrib. Comp. USA (1996) 282–291
5. Eggen, M., Eggen, R.: Load Balancing on a Non-dedicated Heterogeneous Network of Workstations. Parallel and Distributed Processing, Techniques and Applications. Las Vegas, USA Vol. II (2002) 856–862
6. Maheshwari, P.: A Dynamic Load Balancing Algorithm for a Heterogeneous Computing Environment. IEEE Conference on System Sciences. USA (1996) 338–346
7. Sánchez D., Macías E., Suárez A.: Effective Load Balancing on a LAN-WLAN Cluster. Parallel and Distributed Processing, Techniques and Applications. Las Vegas, USA (2003) 473–479
8. Subramanian M.: Network Management: Principles and Practice. Addison-Wesley (2000)
9. Castellano J.P., Sánchez D., Cazorla O., Bordón J., Suárez A.: GACSYS: a VHDL based Hw/Sw Codesign Tool. Design and Diagnostics of Electronic Circuits and Systems. Szcyrk, Poland (1998) 293–299

Performance Study of Scheduling Mechanisms for Peer-to-Peer Computing Environments

Ilias Savvas and Tahar Kechadi

Parallel Computational Research Group,
Department of Computer Science
University College Dublin
Belfield, Dublin 4, Ireland.
savvas@teilar.gr, tahar.kechadi@ucd.ie

Abstract. Peer-to-peer (P2P) computing has emerged as an alternative model of communication and computation to client-server model. While, P2P computing may significantly increase the performance and the scalability of the whole system, they still are facing many challenges in achieving these goals. In this paper we study the problem of scheduling a large number of tasks on such systems. We propose two techniques based on heuristic approach: a Recursive Neighbour Search and an Augmented Tabu-Search technique. These techniques do not address directly the load-balancing problem since it is completely unrealistic in such large environments, but we will show that even a non-perfectly load-balanced system can behave reasonably well by taking into account the tasks' time demands. These algorithms are compared to a well known scheduling algorithm in order validate their performance.

Keywords: Peer-to-peer computing, task scheduling, heuristic techniques, Tabu-Search.

1 Introduction

A typical Peer-to-peer (P2P) system is composed of a large number of autonomous and distributed nodes, called peers. These peers can share data and resources, such as spare computing cycles and storage capacity, and collaborate spontaneously in a network without the necessity of central coordinator. Internet-connected PCs constitutes an example of P2P systems. This system offers at least 10 billion MHz of processing power and 10 petabytes of storage [3]. Unlike the traditional client-server model, all peers of a P2P system are equal, as they can act as clients and servers. Therefore, P2P technology may significantly increase the performance and reliability of the overall distributed system since it does not rely on dedicated servers [4].

P2P technology is not new; it has been around for a while. However, this model of communication and computation has started again to gain significant importance [5]. This is due to two main reasons: Firstly, the success of file sharing utilities, such as Nepster, Gnutella, BearShare, etc., has brought the idea back

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 954–962, 2004.
© Springer-Verlag Berlin Heidelberg 2004

in new forms. Secondly, the success of Internet and the significant improvement in the performance of the hardware resources (CPU speed and storage capacity and networking facilities) open new era of computing. There are many important applications to P2P technology. In addition to file sharing collaborative P2P spaces, e-commerce models [3,13,7], instant messaging, and Grid and distributed computing are key applications of P2P. In the area of distributed computing P2P has attracted several institutions and organisations. P2P allows them to use the computing resources of idle peers (clients) to compute their applications in order to reduce the cost and administration.

The P2P computing raises many challenges which constitute interesting research problems in distributed systems. These challenges include resource discovery, security, replication, programming paradigms, migration, caching, scheduling, heterogeneity, utilisation of untrusted resources, etc. In this paper we are concerned with the study of scheduling a large number of tasks in a P2P computing environment.

A number of scheduling tools have been developed in order to help users to develop their applications and optimise the performance of their system. Some of them provide a graphical user interface (GUI) and the ability to interactively specify the application, partition and schedule it, and finally analyse its performance [2,9,1]. Other scheduling techniques are based on the tasks assignment during their execution, from overloaded peers to under-loaded peers with the assumption that if the load among all peers is balanced, then the overall execution time of the application is minimised. Here, the system has to decide, according to some local information, whether a task has to be transferred or not and to which peer must be transferred [8,10,14,15]. Many of these algorithms have been proposed and many of system parameters have been measured but it seems that each P2P system may need different policies according to its characteristics and its users expectations [6]. In this paper we propose a technique based on two different algorithms. They are executed according to the state of the system as they both have different features.

2 Model and Problem Statement

A P2P computing system consists of N peers, each with capacity ρ which is the total load that the peer can hold. We assume that the following hold in a P2P network:

- The nodes (peers) are connected; and there is at least one path between any pair of peers.
- Each peer is autonomous and has a full information on its own resources. It manages its load ρ_i which represents the number of waiting tasks (or jobs) to be executed.
- Each peer has a processing speed τ_i. For the purpose of this paper it is defined as the number of work units that can be execute per unit of time.

- Each peer s has a neighbourhood $\mathcal{N}(s)$, defined as a set of peers that are close. For instance, one can define the neighbourhood as a set of peers connected directly to it, or as a set of peers which can be reached within certain access time. The notion of neighbourhood can be extended to any distance measure defined according to P2P applications. The number N of peers is assumed to be very large and the topology is irregular.
- The tasks are independent and can be performed by any peer regardless its initial placement. The initial placement of a task can be any peer of the system.
- A task is characterised by three parameters: 1) the number of work units (in terms of computations) in the task, β_i. 2) The number of packets required to transfer the task, μ_i. 3) The number of times that the task has been already transferred, ϵ_i.

3 Task Scheduling Using Heuristics

The response time of a task depends on the combination of three major factors: the peer processing speed and the peer load, and the communication time if the task has to migrate. The major problem when trying to minimise the response time of a task is to find a peer in the system that has the best combination of (processing power, load) while the transfer time should also be kept to minimum. However this problem is known to be NP-complete. Heuristic methods appear to be a suitable approach to solve this class of problems [11].

3.1 Recursive Local Neighbour Search (RLNS)

For each new arrival of a task t_i to a peer v_k, its set of local tasks is updated; $T' = T \cup \{t_i\}$. The peer v_k performs a local neighbourhood search in order to find a more suitable (optimises its response time) peer for it within its neighbours. The new elected peer updates its local set of task and repeats the procedure of funding a better peer within its neighbourhood. The procedure will stop when a peer accepts it. The RLNS algorithm, for an incoming task is described below.

3.2 Augmented Tabu-Search (ATS)

The first phase of this technique uses RLNS, which balances the load of the tasks locally on each neighbourhood. However, the whole system may not be balanced at all. Therefore, when the system reaches an unbalanced threshold, the Augmented Tabu-Search algorithm (ATS - phase 2) is triggered. Since the ATS method produces an overhead (both communication and computation), it is not efficient to apply it at any new arrival task. The ATS process should be triggered only when the load imbalance of the system is beyond a threshold value. The problem is how to determine efficiently this critical point in order to proceed with balancing the system using ATS. A general procedure of determining the critical point is by trying to identify under-loaded and/or over-loaded peers.

Algorithm 1 Recursive Local Neighbor Search - RLNS

```
1: Response time R_i(v_k) and set v_best = v_k.
2: repeat
3:    for all v_x ∈ N(v_k) do
4:       Response time R_i(t_i, v_best → v_x).
5:       if R_i(t_i, v_best → v_x) then
6:          Set v_best = v_x.
7:       end if
8:    end for
9:    if v_best ≠ v_k then
10:      Migrate t_i to peer v_best
11:      Set v_x = v_k and v_k = v_best
12:      Set N(v_k) = N(v_k) − {v_x}.
13:   end if
14: until v_best = v_k.
```

The problem that we face in a large P2P system is the lack of global information about the state of the system. This situation makes the global solution for the task scheduling problem nearly unachievable. A Tabu Search like technique, oriented to a more or less fuzzy local environment, will attempt to complement the first phase (RLNS) by overcoming bottlenecks and weaknesses of a local scheduling. The proposed ATS algorithm considers the following key issues:

- **Tabu active and Tabu inactive:** We proved in [12] that after the application of RLNS, the powerful peers of a neighbourhood structure have the largest load, therefore,these peers must be excluded by the ATS. Thus, the most powerful peers are the Tabu active elements and the less powerful peers constitute the Tabu inactive elements.
- **Threshold value and triggering point:** We use the local information of the neighbourhood. If the work load is fairly distributed among the most powerful peers (as each peer belongs to more than one neighbourhoods), while the weakest peers are under-loaded than these peers may operate like the bottleneck to RLNS (proposition 1). This could be the triggering point of the ATS algorithm. The ATS algorithm is summarised below.

Proposition 1. *A relatively weak peer could cause the bottleneck of RLNS algorithm to its overlapping neighbourhood structure.*

4 Complexity and Performance

The response time R_{ij}, of a task t_i executed on peer v_j, is defined as the time spent in the system. R_{ij} can be expressed as the sum of the computational time and the communication time. Let $T_{cmp}(t_i, v_j)$ be the time needed by peer v_j to execute the task t_i. In order to calculate the communication time, assume

Algorithm 2 Augmented Tabu Search - ATS

1: Determine the over-loaded peer v_{o_x}, from the last incoming task t_i.
2: Update the set of candidate peers, $V'_{cand} = \{v_{u_1}, v_{u_2}, \cdots, v_{u_\mu}\}$. The set V'_{cand}, is sorted by the processing power of the peers, in increasing order.
3: **if** $V'_{cand} = \emptyset$ **then**
4: Update the history record H, using the aspiration criteria.
5: **end if**
6: **if** $V'_{cand} = \emptyset$ **then**
7: The algorithm ends.
8: **end if**
9: **repeat**
10: Set the candidate peer to be $v_{cand} \leftarrow v_{u_1}$.
11: Set $V'_{cand} = V'_{cand} - \{v_{u_1}\}$, so $V'_{cand} = \{v_{u_1}, v_{u_2}, \cdots, v_{u_\nu}\}$
12: Update the history record H.
13: Migrate t_i to v_{cand}
14: Perform RLNS for t_i, on v_{cand}.
15: **until** $(V'_{cand} = \emptyset)$ **or** (t_i does not return from RLNS to v_{o_x}).

that the task t_i has been transferred from the peer v_k to v_j through a path $v_k = v_{x_1}, v_{x_2}, ..., v_{x_l} = v_k$, then the response time is given by the following:

$$R_i(t_i, v_k \rightarrow v_j) = T_{cmp}(t_i, v_j) + \sum_{q=x_1}^{x_l} T_{com}^{t_i}(v_q, v_{q+l}) \tag{1}$$

where the peers v_q and v_{q+1} are neighbours. A peer can reject a task by sending it to one of its neighbours. The decision of accepting or rejecting a task is made by the ATS scheduler.

Due to the complexity of large P2P systems and all the problems we may encounter in finding such large systems, we adopt the technique of simulating the system. The goal is twofold: 1) validate the theoretical model developed, and 2) examine the behaviour and the efficiency of the proposed technique. At the application level, two different probability distributions have been considered at different levels: uniform and Poisson were both used for the task arrival time, while uniform distribution was used for the tasks size and the communication overhead produced by the tasks' migration. The system parameters have been chosen in order to simulate as close as possible the attributes of both small P2P and large P2P networks like the Internet.

4.1 Results Analysis

The performance measures used in this study are the response time of tasks (expressed as a cost function P) and the cost of the links. P_i of a task t_i is defined as the mean time that a task takes to be executed. This is given by:

$$P_i = \frac{1}{\lambda} \sum_{k=1}^{n} \frac{R_{ik}}{R_{ik} - R_{il}} \tag{2}$$

Where i is the task index, k the originating peer, l the destination peer, and λ is the total number of tasks submitted to the system. P_i has to be compared to a maximum delay value P_{max} and has to be less than this maximum value in order to ensure the non-congestion of the network. The goal is to determine the average number of packets carried out by each link. It is uniquely determined by the traffic matrix and by the routing policy. In this study, the routing policy is fixed and it is based on the Euclidean distance between peers. Basically, one of the targets is to find a topology which satisfies delay an connectivity constraints at the lowest possible cost.

The number of peers varied from 16 to 900. Since similar results were obtained from all size networks while comparing the two techniques, a 16-peer model is presented in detail, mainly for clarity reasons. The task redistribution on different topologies; 4x4 grid, 4-dimensional hypercube, 16-peers ring, and 16-peers irregular P2P are presented in table 1. The first two columns present the initial placement and the perfect load balanced system according to the effective processing power of each peer, respectively. The remaining columns display the results after applying the RLNS algorithm. The table 4.2 presents the task redistribution using the ATS algorithm. One can notice that the ATS technique is smoothing even more the load, reducing the overall response time of the tasks. The improvement gained by ATS over RLNS is between 1.58% to 4.63%, depending mainly on the networks structure. The irregular topology presents the highest improvement.

Table 1. Scheduling of tasks using the RLNS algorithm.

Peer	Initial	Per.Bal.	Irregular	Ring	Hypercube	Grid
1	3687148	**4179544**	4626826	4455279	4245273	4436089
2	3192923	**1240802**	1184912	1186877	1258163	1196109
3	3453950	**5942790**	5947464	5079445	5786902	5933807
4	3391951	**3918323**	3847943	4127419	3988420	3819765
5	3744361	**4832598**	5063223	4929289	4860514	4953505
6	3615381	**3657101**	3809592	3138713	3610755	4685076
7	3371301	**3722407**	3909655	3993438	3809924	3683650
8	3331615	**1110191**	1154869	1151449	1105445	1111850
9	3547875	**3918323**	4125551	4241390	3985895	4039574
10	3346761	**979581**	1029294	1026941	951637	985989
11	3747913	**6204011**	5342208	5843833	6091713	6017991
12	3270446	**5485652**	5148146	5307459	5062443	5038539
13	3392626	**3983628**	3792335	4150214	4091269	4221514
14	3700194	**718359**	714690	770302	742242	706770
15	3676311	**1959161**	2169642	2176587	2072321	2110311
16	3626565	**4244850**	4230971	4518686	4434405	4156782

Finally, one wants to explore how the network size affects the efficiency of the algorithms. We vary the size of network while keeping the number of tasks

Table 2. Scheduling of tasks using the ATS algorithm.

Peer	Initial	Per.Bal.	Irregular	Ring	Hypercube	Grid
1	3687148	**4179544**	4206848	4367679	4131738	4315121
2	3192923	**1240802**	1186348	1146311	1228573	1192515
3	3453950	**5942790**	5894189	5635952	5786928	5759569
4	3391951	**3918323**	3908069	4016773	4013349	3853658
5	3744361	**4832598**	4956825	4883420	4902507	4843740
6	3615381	**3657101**	3694750	3237216	3671722	3703323
7	3371301	**3722407**	3710038	3919071	3804512	3609194
8	3331615	**1110191**	992667	1093301	1112418	1106290
9	3547875	**3918323**	3936692	3899849	4014817	4093043
10	3346761	**979581**	1034540	1019964	942667	997117
11	3747913	**6204011**	6144200	6167678	6219407	6044817
12	3270446	**5485652**	5423125	5317454	5211415	5626694
13	3392626	**3983628**	4058958	4151902	3925748	4088928
14	3700194	**718359**	727618	765458	741166	714598
15	3676311	**1959161**	2054437	2027412	1976223	2077047
16	3626565	**4244850**	4168017	4447881	4414131	4071667

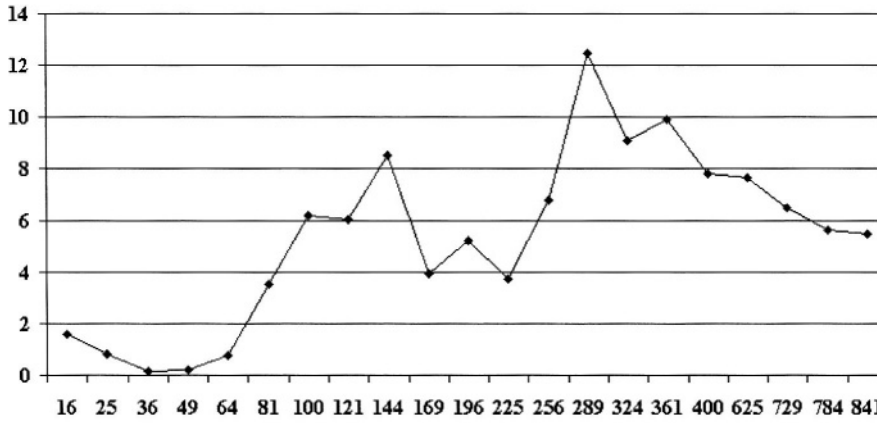


Fig. 1. Efficiency of ATS over RLNS

the same. In figure 1, the efficiency of ATS algorithm over RLNS. ATS presents an improvement between 1% to 12%, and an average of 7%.

4.2 Algorithms Analysis and Overhead

The computations needed for the RLNS algorithm are highly depended on the neighbourhood structure. The overhead has to perform for one move is given by the maximum neighbourhood size. The communication overhead produced, is function of the bandwidth and the effective rate between the neighbour peers.

5 Conclusions and Future Work

In this study, we proposed a Tabu Search algorithm that operates on two phases for the task scheduling problem on large P2P systems. As it has been shown, the algorithm performs very well and speedups the system. The first phase of the algorithm is the one, which affects the more and offers a very good improvement. In order to void possible bottlenecks, the second phase of the Tabu Search meta-heuristic algorithm is triggered to schedule tasks that are not dealt with properly during the first phase. The results were promising. We will extend these results to study other features of a P2P system, where the peers can leave or join the system during the schedule.

Further extension of this work consists in the application of the algorithm to tasks with time dependencies among them, as well as with tasks which are peer oriented, i.e., tasks which can be executed only on specific peers or set of peers, and finally, the combination of these both sub-cases is the final step of this research.

References

1. K. Kavi J. Marquis B. Shirazi, H.B. Chen and A.R. Hurson. Parsa: A parallel program software tool. In IEEE CS Press, editor, *1994 Symp. Assessment of Quality Software Development Tools,* Los Alamitos, Calif, 1994.
2. T. Bemmerl. The topsys architecture. In Springer Verlag, editor, *CONPAR '90,* volume 457, pages 732–743, New York, N.Y., 1990.
3. D. Clark. Face-to-face with peer-to-peer networking. *Computer,* 34(1):18–21, Jan. 2001.
4. L. Gong. Peer-to-peer networks in action. *IEEE Internet Computing,* pages 37–39, Jan./Feb. 2002.
5. D. Karger R. Morris H. Balakrishnan, M.F. Kaashoek and I. Stoica. Looking up data in p2p systems. *CACM,* 46(2):43–48, Feb. 2003.
6. P. Krueger and M. Livny. The diverse objectives of distributed scheduling policies. In IEEE CS Press, editor, *Proc. IEEE 7th Int'l Conf. On Distributed Computing Systems,* pages 242–249, Los Alamitos, Calif, 1987.
7. J. Lee. End-user perspective on file-sharing systems. *CACM,* 46(2):49–53, Feb. 2003.
8. S.Y. Lee and C.H. Cho. Load balancing for minimizing execution time of a target job on a network of heterogeneous workstations. In D.G. Feitelson and L. Rudolph, editors, *JSSPP'00,* pages 174–186, 2000.
9. V. Lo, S. Rajopadhy, S. Gupta, D. Keldsen, M. Mohamed, B. Nitzberg, J. Telle, and X. Zhong. Oregami: Tools for mapping parallel computations architectures. In *Int'l Parallel Programming,* volume 20, pages 237–270, 1991.
10. S. Pulidas, D. Towsley, and J. Stankovic. Imbedding gradient estimators in load balancing algorithms. In IEEE CS Press, editor, *Proc. IEEE 8th Int'l Conf. On Distributed Computing Systems,* pages 482–490, Los Alamitos, Calif., 1988.
11. C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* McGraw-Hill, 1995.
12. I. Savvas and M-T. Kechadi. Some results on the load distribution of heterogeneous distributed systems. In *SCI-02,* Orlando, Florida, USA., June 2002.

13. V.A.F. Almeida W. Meira, D.A. Menascé and R. Fonseca. E-representative: A scalable scheme for e-commerce. In *2nd Int'l. Workshop on Advanced Issues of E-commerce and Web-Based Infroamtion Systems (WECWIS-00),* Milpitas, CA, June 8-9 2000.
14. Y.T. Wang and R.J.T. Morris. Load sharing in distributed systems. *IEEE Trans. Computers,* C-34(3):204–217, March 1985.
15. S. Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Trans. Software Eng.,* 14(11):1327–1341, November 1988.

Analyzing the Efficiency of Block-Cyclic Checkerboard Partitioning in Neville Elimination

Policarpo Abascal[1], Pedro Alonso[1], Raquel Cortina[2], Irene Díaz[2], and José Ranilla[2]

[1] Departamento de Matemáticas, Universidad de Oviedo, E-33271 Gijón, Spain
[2] Departamento de Informática, Universidad de Oviedo, E-33271 Gijón, Spain

Abstract. In this paper we analyze the performance of the Neville method when a block-cyclic checkerboard partitioning is used. This partitioning can exploit more concurrency than the striped method because the matrix computation can be divided out among more processors than in the case of striping. Concretely, it divides the matrix into blocks and maps them in a cyclic way among the processors. The performance of this parallel system is measured in terms of efficiency, which in this case is close to one when the optimum block size is used and it is run on a *Parallel PC Cluster.*

1 Introduction

The recent availability of advanced-architecture computers has had a significant impact on all spheres of scientific computation, including algorithm research and software development in numerical linear algebra (for both serial and parallel computers). In particular, the solution of linear systems of equations lies at the heart of most calculations in scientific computing.

Neville elimination appears naturally when Neville's strategy of interpolation is used for solving linear systems; this also occurs with the Gaussian method and Aitken's strategy. This process is an alternative to Gaussian elimination that has been proved to be very useful with totally positive matrices, sign-regular matrices or other related types of matrices (see [8] and [10]), without increasing error bounds (see [2] and [9]).

A real matrix is called totally positive if all its minors are non-negative. Totally positive matrices arise naturally in many areas of Mathematics, Statistics, Economics, etc. (see [1]).

To process a matrix in parallel, we must partition it so that the partitions can be assigned to different processors. Data partitioning significantly affects the performance of a parallel system. Hence, it is important to determine which data-mapping scheme is the most appropriate one for each algorithm. In order to distribute the matrices among the processors, two different methods are usually taken into consideration: striped partitioning and checkerboard partitioning. When striped partitioning is performed, the matrix is divided into groups

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 963–968, 2004.
© Springer-Verlag Berlin Heidelberg 2004

of complete rows or columns. In this case, each processor is assigned to one such group.

Recent papers of ours (see [3,4,5,6]) have focused on studying the performance of Neville method, considering different strategies in which the coefficient matrix is striped into blocks of rows (or columns) and these blocks are distributed among the processors.

In the present paper, we analyze the performance of Neville's method when checkerboard partitioning is performed. In this partitioning, the matrix is divided into smaller square or rectangular blocks (or submatrices) that are distributed among processors. Checkerboard partitioning splits both the rows and the columns of the matrix, so a non-complete row or column is assigned to each processor. We focus the study around a particular case, namely block-cyclic checkerboard partitioning. In this partitioning, the blocks are mapped in a cyclic way among the processors. The performance of this parallel algorithm is analyzed using some common metrics (see [12]).

The theoretical and numerical results are compared in the last part of this paper. Theoretical expectancies are successful in the sense that they coincide with the results obtained when implementing the algorithm on a *Parallel PC Cluster*. In our algorithm, efficiency is close to one when the optimum block size is used.

2 Neville Elimination

Neville elimination is a method for upper-triangularization which is the first step for solving a system of linear equations $Ax = b$ (see [11]). Neville elimination is a procedure to make zeros in a column of a matrix by adding to each row an adequate multiple of the previous one (instead of using just a row with a fixed pivot, as in Gaussian elimination). For a detailed introduction to this process, we refer the reader to [9].

Let us consider the important case in which Neville elimination can be performed without changing rows; this happens, for example, when A is a non-singular totally positive matrix (see [9]).

If A is a non-singular (which is the most common case) square matrix of order n, this elimination procedure consists of at most $n-1$ successive major steps, resulting in a sequence of matrices $A = A^{(1)}, A^{(2)}, \ldots, A^{(n)} = U$, where U is an upper triangular matrix. We have that for each $1 \leq k \leq n$, the matrix $A^{(k)} = (a_{ij}^{(k)})_{1 \leq i,j \leq n}$ has zeros below its main diagonal in the first $k-1$ columns. To get $A^{(k+1)}$ from $A^{(k)}$, we produce zeros in the column k below the main diagonal by subtracting a multiple of the i^{th} row from the $(i+1)^{th}$ for $i = n-1, n-2, \ldots, k$.

Therefore, if back-substitution is considered the sequential run time of the procedure is

$$T(n;1) = \frac{4n^3 + 3n^2 - 7n}{6} t_c \approx \frac{2n^3}{3} t_c, \tag{1}$$

where t_c is the time spent to carry out one operation in float point.

This cost coincides with the cost of sequential Gaussian elimination (see [12]). However, the cost of Neville's elimination method applied to certain matrices is lower than Gaussian cost (see [10]).

3 Block-Cyclic Checkerboard Partitioning

In this section we consider a parallel implementation of Neville's algorithm in which the matrix A of a linear equation system $Ax = b$ is divided into smaller square blocks. This generic distribution is called checkerboard partitioning.

Consider a two-dimensional wraparound mesh of k processors, such that $k \leq n^2$. The processor in row i and column j is denoted by P_{ij}, with $i = 1, ..., \sqrt{k}$ and $j = 1, ..., \sqrt{k}$. A two-way communication line links P_{ij} to its neighbors $P_{i+1,j}$, $P_{i-1,j}$, $P_{i,j+1}$ and $P_{i,j-1}$.

Let $A = (A_{ij})_{1 \leq i,j \leq p}$ be an $n \times n$ matrix and A_{ij} a $s \times s$ submatrix of A for $1 \leq i, j \leq p$, then each one of the A_{ij} blocks are split among the processors in a cyclic way.

This partitioning, called block-cyclic checkerboard, assigns $h \times h$ blocks of $m \times m$ consecutive elements to each processor, being $h = n/(m\sqrt{k})$ with $m \in [1, n/\sqrt{k}]$ (see [12]). If $m = n/\sqrt{k}$ this algorithm is known as block checkerboard partitioning and if $m = 1$ the method is named cyclic checkerboard partitioning.

If we need to implement a matrix algorithm by using checkerboarding on an architecture other than mesh, the logical mesh is embedded in the physical interconnection network.

3.1 Algorithm

Let us study the parallel algorithm performance in an iteration j. In this iteration the variable x_j must be removed. Hence, it is necessary to reduce the elements $a_{nj}^{(j)}, a_{n-1,j}^{(j)}, \ldots, a_{j+1,j}^{(j)}$ to zero. In order to get this result, it is necessary to follow four steps:

- *Step 1.* A processor P_{il} sends the last row of each block to processor $P_{i+1,l}$ when the index of this row is larger than j. If $i = \sqrt{k}$, the elements are transferred to processor P_{1l}. These rows are necessary to obtain $A^{(j+1)}$ from $A^{(j)}$.
- *Step 2.* Calculate the multipliers to get zeros in entries (n, j), $(n-1, j)$, ..., $(j+1, j)$.
- *Step 3.* Transfer the multipliers calculated in P_{il} (in *Step 2*) to each active processor situated in row i of the mesh.
- *Step 4.* Calculate the elements $a_{it}^{(j+1)}$ with $j \leq t \leq n$.

Let $T(n; k)$ be the total parallel cost when k processors are used. This cost is obtained by computing communication and computation time. Hence

$$T(n; k) = t_{\text{computation}} + t_{\text{communication}}. \tag{2}$$

3.2 Computation and Communication Time

In this algorithm the computations are made simultaneously over all processors. Hence, computation cost is studied for processor $P_{\sqrt{k}\sqrt{k}}$ because it is the processor that makes the largest number of computations.

Let the computation time be $t_{\text{computation}} = (n_{sr} + n_p + n_d)t_c$, where t_c is the time needed to make a float operation, and n_{sr}, n_p and n_d are the number of sums/subtractions, products or divisions respectively.

Thus, the computation time is

$$t_{\text{computation}} \approx \left(4n^2 + 6mn\sqrt{k} - 6mn + 2m^2k - 2m^2\sqrt{k}\right)\frac{n}{6k}t_c. \tag{3}$$

As regards communication time, the communication model used in this work is the model defined by Kumar in [12].

This approximation is based on an MIMD distributed memory architecture, where the processors are connected using an interconnection network. In this case, a message containing α words (float numbers) can be sent between directly-connected processors in time $t_s + \alpha t_w$, where t_s is the time required to handle a message at a sending processor and t_w is the transfer time of a number in float point.

Taking into account all of the above considerations, if the size of the message is α, then the communication time of a one-to-all broadcast with a cut-through routing is $\log_2(k)t_s + \alpha\log_2(k)t_w + 2(\sqrt{k}-1)t_h$ for a mesh, where k is the number of processors. As in Kumar and Dongarra (see [12] and [13]), the term $2(\sqrt{k}-1)t_h$ is not considered, due to t_h (per-hop time) being quite small on most parallel systems.

The communication time is now calculated for parallel Neville elimination. There exist two kinds of communications.

In *Step 1* a processor P_{il} sends the last row of each block to processor $P_{i+1,l}$, so a communication between neighboring processors is made. Consequently, the processor that makes the most number of communications is $P_{\sqrt{k}-1,\sqrt{k}}$.

In *Step 3,* on the other hand, a processor transfers the multipliers to the processors that need them (one-to-all broadcast). The processors of the row $\sqrt{k}$ make the most number of communications.

Hence the communication time $(t_{communication})$ is approximately

$$n\log_2(\sqrt{k})t_s + \left(\frac{n^3}{3mk} + \frac{n^2}{2\sqrt{k}} + \frac{nm}{6} + \frac{n^2\log_2(\sqrt{k})}{2\sqrt{k}} + \frac{nm\log_2(\sqrt{k})}{2}\right)t_w. \tag{4}$$

3.3 Efficiency

The performance of Neville elimination is studied by measuring its efficiency.

Therefore, once communication and computation time are calculated, we are able to estimate the efficiency of block-cyclic checkerboard partitioning. The theoretical efficiency reached for large values of n

$$E \approx \frac{2t_c}{2t_c + \frac{1}{m}t_w}, \text{ with } m \in \left[1, \frac{n}{\sqrt{k}}\right]. \tag{5}$$

Note that efficiency depends on the block size, the time spent to carry out one operation in float point and the transfer time of a number in float point.

In practice, the efficiency is between zero and one, depending on the degree of effectiveness with which the processors are used. In our algorithm, efficiency is almost one.

4 Numerical Results

Empirical and theoretical performances are compared at this point, as it is well known that the empirical behavior of both may be different due to a deficient model, collateral effects or overheads. If the empirical and the theoretical efficiencies present the same behavior, then the theoretical values will allow us to extrapolate what will happen when resources are insufficient (typically memory size).

The algorithms were thus implemented and their empirical efficiency was obtained when executing them on a *Parallel PC cluster* using a public domain Message Passing Interface (MPICH) and a Linux Terminal Server Project.

The theoretic model constants used in this work were likewise estimated in the same conditions. These constants, t_c, t_s and t_w are:

$$t_c = 3.8 * 10 - 8\ s,\ t_s = 8.8 * 10 - 5\ s,\ t_w = 1.3 * 10 - 6\ s. \tag{6}$$

All of the values were obtained using a least squares approximation, as in [7].

We now analyze what happens with the studied partitioning using the constants estimated above. We use all the terms in expressions (3) and (4) and the values computed in (6) in order to obtain an estimation of its behavior.

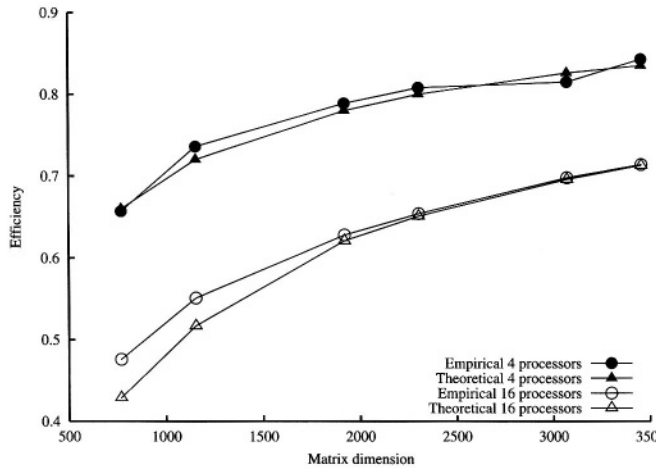


Fig. 1. Theoretical and empirical efficiency for block-cyclic checkerboard partitioning.

Figure 1 shows the efficiency (both theoretical and empirical) obtained for block-cyclic checkerboard partitioning using the size of block that provides maximum efficiency (optimum block size). To carry out experiments, we look for the integer closest to the size of such an ideal block that allows all the processors to have the same number of blocks. Due to the memory size of each node, it is not possible to run the sequential algorithm for matrix dimensions larger than 3500. The performance of both efficiencies is quite similar.

On the other hand, let us consider the special cases where the block size is $m = 1$ and $m = n/\sqrt{k}$. Thus, for example, when $m = n/\sqrt{k}$, $n = 3600$ and $k = 4$, theoretical efficiency is 0.499 and empirical efficiency is 0.493. If $m = 1$, $n = 3720$ and $k = 4$, theoretical efficiency is 0.055 and empirical efficiency is 0.045. Note that in these last cases the efficiencies (both theoretical and empirical) are far from one, especially when $m = 1$.

Therefore, we can conclude that the performance model used is adequate and there are no collateral effects when working with the studied partitionings. What is more, if the optimum block size is used, the obtained efficiency is very close to one when the problem size and the number of processors are appropriate.

References

1. T. Ando, Totally positive matrices, *Linear Algebra Appl.* 90: 165-219 (1987).
2. P. Alonso, M. Gasca and J.M. Peña, Backward error analysis of Neville elimination, *Appl. Numer. Math.* 23: 193-204 (1997).
3. P. Alonso, R. Cortina and J. Ranilla, Block-Striped partitioning and Neville elimination, *Lect. Notes in Comput. Sc.* 1685: 1073-1077 (1999).
4. P. Alonso, R. Cortina, V. Hernández and J. Ranilla, Study the performance of Neville elimination using two kinds of partitioning techniques, *Linear Algebra Appl.* 332-334: 111-117 (2001).
5. P. Alonso, R. Cortina, I. Díaz, V. Hernández and J. Ranilla, A Columnwise Block Striping in Neville Elimination, *Lecture Notes in Comput. Sci.* 2328: 379-386 (2002).
6. P. Alonso, R. Cortina, I. Díaz, V. Hernández and J. Ranilla, A Simple Cost-Optimal parallel algorithm to solve linear equation systems. *International Journal of Information* 6(3): 297-304 (2003).
7. J.J. Dongarra, Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report), University of Tennessee Computer Science Technical Report, CS-89-85, 2001.
8. M. Gasca and G. Mühlbach, Elimination techniques: from extrapolation to totally positive matrices and CAGD, *J. Comput. Appl Math.* 122: 37-50 (2000).
9. M. Gasca and J.M. Peña, Total positivity and Neville elimination, *Linear Algebra Appl.* 165: 25-44 (1992).
10. M. Gasca and J.M. Peña, A matricial description of Neville elimination with applications to total positivity, *Linear Algebra Appl.* 202: 33-45 (1994).
11. G.H. Golub and C.F. Van Loan, *Matrix computations,* Johns Hopkins, 1989.
12. V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing. Design and Analysis of Algorithms,* The Benjamin/Cummings, 1994.
13. A.P. Petitet and J.J. Dongarra, Algorithmic Redistribution Methods for Block Cyclic Decompositions, *IEEE T. Parall. Distr.* 10: 201-220 (1999).

Parallel Algorithms for the Solution of Toeplitz Systems of Linear Equations*

Pedro Alonso[1], José M. Badía[2], and Antonio M. Vidal[1]

[1] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, cno. Vera, s/n, 46022 Valencia, Spain
{palonso,avidal}@dsic.upv.es

[2] Departamento de Ingeniería y Ciencia de los Computadores, Universidad Jaume I, Campus de Riu Sec, 12071 Castellón de la Plana, Spain
badia@inf.uji.es

Abstract. In this paper we present two parallel algorithms to solve non-symmetric Toeplitz systems of linear equations. The first algorithm performs a modified QR factorization of the matrix by using the generalized Schur algorithm. The second one is based on the transformation of the Toeplitz matrix into a Cauchy-like matrix in order to reduce the communication cost. Both sequential methods have small computational cost. This fact makes it difficult to implement efficient parallel algorithms. We have tested the efficiency and stability of the algorithms on a cluster of personal computers. The results show the speed-up reaches the number of processors in many cases and both algorithms offer an accurate solution of the linear system. Besides, we have used public domain computation and communication libraries in order to get portable codes.

1 Introduction

In this paper we present two new parallel algorithms based on "fast" sequential methods for solving Toeplitz linear systems:

$$Tx = b\,, \tag{1}$$

where $T \in \mathrm{R}^{n\times n}$ is a Toeplitz matrix of the form $T = (t_{ij}) = (t_{i-j})$ for $i, j = 0, \ldots, n-1$, $b \in \mathrm{R}^n$, and $x \in \mathrm{R}^n$ is the solution vector.

Fast algorithms for solving Toeplitz linear systems are based on the displacement rank property of this kind of matrix. There is a group of algorithms called Schur-type that offers more possibilities to implement parallel versions.

Regarding the accuracy of the results, almost all the algorithms that solve Toeplitz systems produce poor results except with strongly regular matrices, that is, matrices with all their leading submatrices well conditioned. Several methods are proposed to improve the solution, including *look-ahead* or refinement techniques [6,3].

* This work was supported by the Spanish project CICYT TIC 2000-1683-C03-03.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 969–976, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Our aim in this work is to offer stable and efficient algorithms for general purpose architectures. Our codes are portable because we extensively use standard libraries like LAPACK [4], ScaLAPACK [5] and BIHAR [10].

The first parallel algorithm presented in this paper solves (1) by means of a modified QR decomposition of T proposed in [3] that improves the accuracy of the solution.

Our second parallel algorithm makes a LU factorization of a Cauchy-like matrix resulting from applying fast trigonometric transformations to the Toeplitz matrix. We exploit Cauchy-like matrices in order to reduce the communication cost avoiding a lot of communications present in the classical fast algorithms.

In the next two sections both parallel algorithms are described. Section 4 includes the experimental analysis and comparison of the algorithms in a cluster of personal computers. Finally, some conclusions are presented.

2 QR Factorization of T

The concept of *displacement structure* was first introduced in [8] to describe the special structure of Toeplitz and Toeplitz-like matrices. Given a symmetric matrix $M \in \mathrm{R}^{n\times n}$, and a lower triangular matrix $F \in \mathrm{R}^{n\times n}$, we call *displacement* of M with respect to F to the matrix $\nabla_F M$ defined as:

$$\nabla_F M = M - FMF^T = GJG^T \ . \tag{2}$$

We say that matrix M has *displacement structure* with respect to F, if the rank r of $\nabla_F M$, is considerably lower than n [9]. Matrix $G \in \mathrm{R}^{n\times r}$ is called *generator* and $J = (I_p \oplus -I_q)$, $r = p + q$, is the *signature* matrix, where p the number of positive eigenvalues of $\nabla_F M$ and q the number of negative eigenvalues.

The Generalized Schur Algorithm (GSA) uses the generator pair (G, J) to factorize matrices with the previous structure in $O(rn)$ operations. Applying GSA to the appropriate matrix we can obtain different factorizations (QR, LU, etc.) of a Toeplitz matrix T. In this paper we use the following matrices

$$M = \begin{pmatrix} T^T T & T^T \\ T & 0 \end{pmatrix} \ , \ F = \begin{pmatrix} Z & 0 \\ 0 & Z \end{pmatrix} \ , \tag{3}$$

where $Z = (z_{ij})_{i,j=1,\ldots,n}$ is the down shift matrix, being $z_{ij} = 1$ if $i + 1 = j$ and 0 otherwise. The QR factorization of a Toeplitz matrix can be obtained by applying n steps of the algorithm GSA to the generator pair (G, J),

$$M = \begin{pmatrix} T^T T & T^T \\ T & 0 \end{pmatrix} = \begin{pmatrix} \mathrm{R}^T \\ Q \end{pmatrix} (I) \begin{pmatrix} R & Q^T \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & -I \end{pmatrix} \ , \tag{4}$$

such as $T^T T = \mathrm{R}^T R$ where R is upper triangular, $T = QR$ and $QQ^T - I = 0$. The displacement rank of M (4) is 5, and the generator can be found in [3].

However, it is not possible to guarantee the stability of the GSA algorithm and the perfect orthogonality of factor Q. To solve this problem, we have incorporated two basic modifications suggested in [3] to our algorithm. First, $2n$ steps of

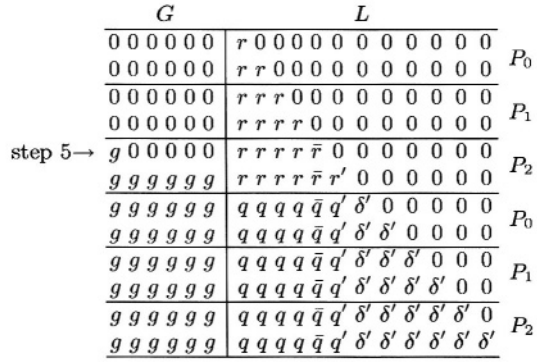


Fig. 1. Example of row block cyclic distribution with $\nu = 2$ of a generator matrix $G \in \mathrm{R}^{18\times 6}$ (case ill-conditioned matrix T), and the triangular matrix $L \in \mathrm{R}^{12\times 12}$ in a mesh of 3×1 processors. The figure shows the PAQR at the middle of the step 5. Entries g are generator entries while r, q and δ denote entries of R^T, Q and Δ respectively. Entries $\bar{r}$ and $\bar{q}$ denote the columnn of L computed at step 5 while entries with $'$ denote values that will be computed in the following steps from 6 to 12.

the GSA algorithm are applied to produce the following triangular factorization:

$$L\hat{L}^T = \begin{pmatrix} R^T & 0 \\ Q & \Delta \end{pmatrix} \begin{pmatrix} R & Q^T \\ 0 & -\Delta^T \end{pmatrix}, \tag{5}$$

so that $(\Delta^{-1}Q)$ is orthogonal and the Toeplitz system (1) can then be solved using, $x = R^{-1}(Q^T\Delta^{-T})\Delta^{-1}b$.

Secondly, if matrix T is ill-conditioned, that is, if $\kappa(T) > 1/\sqrt{\epsilon}$, being ϵ the machine precision, then the algorithm GSA can fail. To avoid this problem a modified matrix M with a displacement rank of 6 is factorized in order to guarantee the backward stability. For a deeper discussion see [3].

The computation of generator G involves matrix-vector products that can be performed in parallel without communications. The generator is distributed cyclically by blocks of ν ($n \bmod \nu = 0$) rows in a one-dimensional mesh of $p \times 1$ processors denoted as P_k, $k = 0, \ldots, p-1$ (see Fig. 1). We use a one-dimensional topology because the generator has only 5 or 6 columns and the operations are applied in parallel on different groups of rows.

The parallel algorithm that we call PAQR (Parallel Algorithm for the QR decomposition of T) proceeds as follows.

1. Compute generator G.
2. Compute QR factorization, for $i = 1, \ldots, n$,
   a) Processor P_k ownin grow i of G (g_i) chooses a J-unitary transformation Θ_i ($\Theta_i J \Theta_i = J$) such as $g_i\Theta_i = \begin{pmatrix} x & 0 & \ldots & 0 \end{pmatrix}$ iff $g_i^T J g_i > 0$ or $g_i\Theta_i = \begin{pmatrix} 0 & \ldots & 0 & x \end{pmatrix}$ otherwise, and broadcasts Θ_i.
   b) The rest of processors update their rows $j = i+1, \ldots, 2n$ of G, $G \leftarrow G\Theta_i$. Column i of L is the first (or the last) column of G.

c) Update g, $g \leftarrow Fg$, being g the first (or the last) column of G.
3. Compute $x = R^{-1}(Q^T \Delta^{-T})\Delta^{-1} b$ by calling some routines of PBLAS.

Step 2c implies a shift one position down the i-th to n entries of g on the one hand, and the $n+1$ to $2n$ entries of g on the other hand. This last operation of the iteration i requires a point-to-point communication between adjacent processors, and involves a great amount of the total time of the parallel algorithm. In order to reduce the communication cost, all elements to be sent from a given processor P_k to processor $P_{\mathrm{mod}(k+1,p)}$ are packed in a unique message.

3 LU Factorization of *T*

We can avoid the shifting operation on each iteration (step 2c) on the PAQR if we transform the Toeplitz matrix into a Cauchy-like matrix. This greatly reduces the communication cost of the parallel algorithm.

We say that C is a Cauchy-like matrix if it is the unique solution of the displacement equation

$$\Omega\, C - C\, \Lambda = G\, H^T\,, \tag{6}$$

where $\Omega = \mathrm{diag}(\omega_1, \ldots, \omega_n)$, $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, and we call matrices G and H of size $n \times r$ *generators.*

Given a Toeplitz matrix T, its displacement equation can be expressed as

$$Z_{00}\, T - T\, Z_{11}^T = \hat{G}\, \hat{H}^T\,. \tag{7}$$

In the displacement equation (7), $Z_{\varepsilon\psi} = Z + Z^T + \varepsilon e_1 e_1^T + \psi e_n e_n^T$ where e_1 and e_n are the first and last columns of the identity matrix respectively and $\varepsilon, \psi \in \{0, 1\}$. An explicit form for $\hat{G}$ and $\hat{H}^T$ can be found in [1].

There exist real discrete trigonometric transformations associated with the FFT that diagonalize the matrix $Z_{\varepsilon\psi}$ in $O(n \log(n))$ operations. The discrete sine transformation S_{00} (DST-I) diagonalizes Z_{00}, while the discrete cosine transformation S_{11} (DCT-II) diagonalizes Z_{11} [1,7]. Using the mentioned transformations, we can convert the displacement equation (7) into the displacement equation (6).

Given a Cauchy-like matrix C (6), its LU factorization can be obtained where L is a unit lower triangular matrix and U is an upper triangular matrix. The algorithm proceeds as follows. In the first step, the first column of C is computed by solving

$$\Omega C_{:,j} - C_{:,j} \Lambda = G H_{j,:}^T\,.$$

Let us partition C and to define matrices X and Y as

$$C = \begin{pmatrix} d & u \\ l & C_1 \end{pmatrix}\,, \quad X = \begin{pmatrix} 1 & 0 \\ l d^{-1} & I \end{pmatrix}\,, \quad Y = \begin{pmatrix} 1 & d^{-1} \\ 0 & I \end{pmatrix}\,,$$

then C can be factorized as

$$C = X \begin{pmatrix} d & 0 \\ 0 & C_s \end{pmatrix} Y\,,$$

where C_s is the Schur complement of C regarding its first element d. Further, let Ω and A be conformally partitioned $\Omega = \left(\Omega_1 \oplus \Omega_2\right)$ and $\Lambda = \left(\Lambda_1 \oplus \Lambda_2\right)$. Applying the transformation $X^{-1}(.)Y^{-1}$ to (6) we obtain the following equation

$$\Omega_2\, C_s - C_s\, \Lambda_2 = G_1\, H_1^T\ , \tag{8}$$

where G_1 is the portion of $X^{-1}G$ from the second row down and H_1 is the portion of $Y^{-1}H$ from the second row down. The first column of L in the LU factorization is $\left(1 \;\; d^{-1}l^T\right)^T$ while the first row of U is $\left(d \;\; u\right)$. The process can now be repeated on the displacement equation (8) of the Schur complement of C with respect to d, C_s, to get the second column of L and row of U.

In the algorithm PALU we have used the same unidimensional mesh topology as in the first one. The generators G and H have been distributed cyclically by blocks of ν rows as it was made with generator G in the previous algorithm PAQR (see Fig. 1). The computed lower triangular factors L and U^T are distributed as the factor L in PAQR.

The parallel algorithm performs a block version of the algorithm described above. Let the following partition for generators G and H and matrix C be,

$$\begin{pmatrix} G_1 \\ G_2 \end{pmatrix}\ ,\quad \begin{pmatrix} H_1 \\ H_2 \end{pmatrix}\ ,\quad C = \begin{pmatrix} C_{11}\, C_{12} \\ C_{21}\, C_{22} \end{pmatrix} = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} \left(U_1\, U_2\right) + \begin{pmatrix} 0 & 0 \\ 0 & C_s \end{pmatrix}\ ,$$

where $G_1, H_1 \in \mathrm{R}^{\nu\times 4}$ and $G_2, H_2 \in \mathrm{R}^{(n-\nu)\times 4}$, $L_1, U_1 \in \mathrm{R}^{\nu\times\nu}$ are lower and upper triangular factors respectively, $L_2, U_2^T \in \mathrm{R}^{(n-\nu)\times\nu}$ and C_s is the Schur complement of C regarding its principal submatrix C_{11}. At the first step of the parallel algorithm, processor P_0 (processor having the firsts blocks G_1 and H_1 of the generators) computes the LU factorization of $C_{11} = L_1U_1$. Next, processor P_0 broadcasts blocks G_1 and H_1 properly updated. The rest of processors receive blocks G_1 and H_1, compute their blocks of factors L_2 and U_2 and update their blocks of G_2 and H_2. G_2 and H_2 are the generators for a displacement equation of C_s of the form (6). The following steps of the algorithm proceed in the same way.

4 Experimental Results

We have performed the experimental analysis of both algorithms on a cluster of personal computers. Each node of this cluster is a Intel Pentium II-300MHz with 128 Mbytes of memory. The nodes are connected through a Myrinet network [2]. The time required for one of these nodes to perform a flop is approximately $1.55 \times 10^{-3}\ \mu s$. On the other hand, we model by $\beta + n\tau$ the time required to send a message of size n between two nodes. The latency time of the network is $\beta = 62\ \mu s$, while the time to transfer each double precision real value is $0.021\ \mu s$.

Table 1 shows the time spent by both parallel algorithms to solve Toeplitz systems of different matrix sizes using only one processor. We can see how the main part of PAQR is devoted to the modified QR decomposition while the largest time spent in PALU is devoted to the LU factorization of a Cauchy–like

Table 1. Time in seconds (percentage of the total time) of each of the three main steps of both algorithms executed in one processor

		PAQR			PALU		
n	$n+1$	Calc. gen.	Mod. QR	System	Calc. gen.	LU	System
1000	$(7 \cdot 11 \cdot 13)$	0.01 (1%)	0.82 (86%)	0.12 (13%)	0.01 (2%)	0.39 (93%)	0.02 (5%)
1200	(1201)	0.03 (2%)	1.36 (88%)	0.16 (10%)	0.27 (29%)	0.61 (66%)	0.04 (4%)
1400	$(3 \cdot 467)$	0.04 (2%)	1.84 (88%)	0.22 (10%)	0.08 (8%)	0.84 (87%)	0.05 (5%)
1600	(1601)	0.05 (2%)	2.38 (87%)	0.30 (11%)	0.47 (29%)	1.10 (67%)	0.06 (4%)
1800	(1801)	0.07 (2%)	3.04 (87%)	0.38 (11%)	0.60 (29%)	1.41 (67%)	0.08 (4%)
2000	$(3 \cdot 23 \cdot 29)$	0.08 (2%)	3.68 (87%)	0.47 (11%)	0.03 (2%)	1.73 (94%)	0.09 (5%)

matrix. The time required to compute the generator in PAQR is almost negligible. However, the study of the generator computation in PALU, that involves the Toeplitz to Cauchy–like translation, shows an interesting behavior. The speed of this process depends on the decomposition in prime factors of $n+1$ (second column of Table 1). The time spent in this step is little as far as the prime factors are little. The final step of the algorithms, that involves several triangular systems solution and matrix–vector products, are carried out by subroutines of the BLAS library optimized for the target machine and it takes a small percentage of the total time.

One important factor that affects the performance of both parallel algorithms is the block size denoted by ν. The value of ν fixes the number of messages and their sizes, therefore, determines the load-balance between computations and communications. In our experiments with PAQR we have seen that with matrices of a size smaller than $n = 1536$, the best block size is n/p, but, with larger matrices, the best block size depends on the size of the matrix. In the case of the algorithm PALU the best value of ν in our cluster is 31.

Table 2 shows time and speed-up of both parallel algorithms up to 8 processors. Algorithm PAQR always improves its sequential version as the number of processors increases. The speed-up obtained with this parallel algorithm are not very good but always grows with the problem size. The limited performance is due to the influence of the communications forced by the down shift of a column of the generator in each iteration. Another constraint of PAQR is the great amount of memory required but the use of several processors allows us to deal with bigger problems.

On the other hand, it can be seen that PALU obtains better time and speed-up than PAQR. However, the first step is a sequential process. This fact limits the maximum speed-up when the prime factors of $n+1$ are large and the number of processors increases, as it can be seen if we compare the speed-ups with matrices of sizes 1800 and 2000 in Table 2. For a matrix of size $n = 4095$ we have obtained an efficiency about 65% in 16 processors. Another advantage of the second parallel algorithm is that it can deal with larger matrices than PAQR.

To analyze the stability of the algorithms we have used the matrix $T = T_0 + \xi T_1$ where T_0 is a symmetric Toeplitz matrix called KMS (Kac-Murdock-

Table 2. Time in seconds (speed-up) of the parallel algorithm PAQR

	PAQR				PALU			
$n/p =$	1	2	4	8	1	2	4	8
1000	0.95	0.87 (1.09)	0.73 (1.30)	0.60 (1.58)	0.45	0.25 (1.80)	0.14 (3.21)	0.10 (4.50)
1200	1.54	1.20 (1.28)	0.97 (1.59)	0.82 (1.88)	0.94	0.66 (1.42)	0.45 (2.09)	0.38 (2.47)
1400	2.10	1.59 (1.32)	1.26 (1.67)	0.99 (2.12)	1.01	0.60 (1.68)	0.34 (2.97)	0.23 (4.39)
1600	2.74	2.02 (1.36)	1.56 (1.76)	1.21 (2.26)	1.70	1.14 (1.49)	0.85 (2.00)	0.65 (2.62)
1800	3.40	2.46 (1.38)	1.85 (1.84)	1.46 (2.33)	2.16	1.45 (1.49)	1.08 (2.00)	0.82 (2.63)
2000	4.23	2.96 (1.43)	2.21 (1.91)	1.68 (2.52)	2.01	1.07 (1.88)	0.63 (3.19)	0.30 (6.70)

Table 3. Forward and backward errors of both parallel algorithms

	PAQR		PALU	
n(cols)	Back. error	For. error	Back. error	For. error
1000 (6)	8.47×10^{-14}	6.40×10^{-3}	4.16×10^{-12}	1.07×10^{-5}
1200 (5)	1.20×10^{-15}	1.20×10^{-13}	4.79×10^{-12}	1.83×10^{-11}
1400 (5)	1.10×10^{-15}	8.17×10^{-13}	6.70×10^{-12}	1.68×10^{-11}
1600 (6)	2.79×10^{-13}	5.01×10^{-2}	6.69×10^{-13}	1.05×10^{-11}
1800 (5)	3.90×10^{-15}	1.15×10^{-12}	1.16×10^{-11}	6.72×10^{-11}
2000 (5)	8.73×10^{-16}	5.75×10^{-13}	1.40×10^{-11}	3.12×10^{-11}

Szegö) whose elements are $t_0 = \epsilon$ and $t_i = t_{-i} = \left(\frac{1}{2}\right)^i$ for $i = 1, 2, \ldots, n-1$, and matrix T_1 is randomly generated. We have chosen $\epsilon = \xi = 10^{-14}$. In this case the leading submatrices of T with sizes $3m+1$, $m = 0, 1, \ldots$ are ill conditioned. Classical Levinson and Schur-type algorithms break down or produce bad results with that matrix because it is not strongly regular. The right-hand side vector b has been chosen in such a way so that the exact solution x is a vector where all elements have a value of one. Now, we have been able to obtain the backward and forward errors,

$$\frac{\|T\tilde{x} - b\|}{\|T\| \cdot \|\tilde{x}\| + \|b\|} \quad \text{and} \quad \frac{\|\tilde{x} - x\|}{\|x\|},$$

where $\tilde{x}$ is the computed solution.

Table 3 shows both errors with the two parallel algorithms. The first column shows the matrix sizes and also shows the number of columns of the generator in the algorithm PAQR. When $n = 3m + 1$ for a given m, $\kappa(T) \approx 10^{14}$. Backward errors of PAQR are good because it produces a corrected QR factorization over the product $T^T T$ which is strongly regular. However, if T is ill conditioned ($n = 1000, 1600$), then $\kappa(T^T T) \gg \kappa(T)$. The Cauchy-like matrix preserves the conditioning of the original Toeplitz matrix. As PAQR works with $T^T T$ while PALU deals with the transformed matrix T, PALU produces smaller forward errors than PAQR with ill-conditioned matrices.

5 Conclusions

Both algorithms presented in this paper parallelize *fast* sequential methods that exploit the displacement structure of Toeplitz matrices. Despite the small computational cost, both parallel algorithms improve their sequential versions. The implemented algorithms are portable because they are based on standard sequential and parallel libraries. They have been tested on a cluster of personal computers, but they can be used on any distributed memory architecture.

Algorithm PAQR involves many communications and has a fine–grain parallelism. This produces small speed–ups in our cluster of personal computers, but the time is reduced with the number of processors. Algorithm PAQR is more backward–stable than PALU in all cases and more accurate for well–conditioned matrices. On the contrary, algorithm PALU avoids a great number of communications and increases the overlapping between computation and communications. Although its efficiency can be affected by the cost of the initial Toeplitz–to–Cauchy transformation, for large matrices with a decomposition of $n + 1$ in small primes we can expect a good efficiency with several processors. Algorithm PALU is more forward–stable with ill–conditioned matrices.

References

1. Pedro Alonso, José M. Badía, and Antonio M. Vidal. Resolución de sistemas lineales de ecuaciones Toeplitz en paralelo por el método de Cauchy. TR DSIC-II/26/2002, DSIC, Universidad Politécnica de Valencia, 2002.
2. N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.K. Su. Myrinet. a gigabit-per-second local-area network. *IEEE Micro,* 15:29–36, 1995.
3. S. Chandrasekaran and Ali H. Sayed. A fast stable solver for nonsymmetric Toeplitz and quasi-Toeplitz systems of linear equations. *SIAM Journal on Matrix Analysis and Applications,* 19(1):107–139, January 1998.
4. E. Anderson et al. *LAPACK Users' Guide.* SIAM, Philadelphia, 1995.
5. L.S. Blackford et al. *ScaLAPACK Users' Guide.* SIAM, Philadelphia, 1997.
6. R. W. Freund. A look-ahead Schur-type algorithm for solving general Toeplitz systems. *Zeitschrift für Angewandte Mathe. und Mechanik,* 74:T538–T541, 1994.
7. Georg Heinig and Adam Bojanczyk. Transformation techniques for Toeplitz and Toeplitz-plus-Hankel matrices. I. transformations. *Linear Algebra and its Applications,* 254(1–3):193–226, March 1997.
8. T. Kailath, S.-Y. Kung, and M. Morf. Displacement ranks of a matrix. *Bulletin of the American Mathematical Society,* 1:769–773, 1979.
9. Thomas Kailath and Ali H. Sayed. Displacement structure: Theory and applications. *SIAM Review,* 37(3):297–386, September 1995.
10. P. N. Swarztrauber. FFT algorithms for vector computers. *Parallel Computing,* 1(1):45–63, August 1984.

An Embedded Iterative Scheme in Electromagnetism

Bruno Carpentieri[1], Iain S. Duff[1,2], Luc Giraud[1], and Guillaume Sylvand[3]

[1] CERFACS, 42 Avenue G. Coriolis, 31057 Toulouse Cedex, France,
{carpenti,duff,giraud}@cerfacs.fr
[2] Also at RAL, Oxfordshire, England
[3] CERMICS-INRIA, Sophia Antipolis,
gsylvand@sophia.inria.fr

Abstract. In this paper we describe an embedded iterative scheme based on the GMRES method for the solution of large electromagnetic scattering problems expressed in an integral formulation. In our scheme, we use a Frobenius-norm minimization preconditioner to speed-up the inner iterations, and the M-V products are carried out using the Fast Multipole Method at different accuracy within the inner and the outer solvers. We show that the proposed technique enables to significantly reduce the computational cost of the simulation on large industrial and academic models. In particular, we are able to solve a linear system with more than one million unknowns arising from a simulation on a real aircraft.

1 Introduction

In recent years a significant amount of work has been devoted to the simulation of wave propagation phenomena, to address the physical issue of detecting the diffraction pattern of the electromagnetic radiation scattered from a large and complex body when illuminated by an incident ingoing wave. For the solution of scattering problems, Maxwell's equations are often formulated in the frequency domain rather than in the time domain. Using the equivalence principle, the initial differential system can be recast in the form of four integral equations which relate the electric and magnetic fields to the equivalent electric and magnetic currents on the surface of the object. Amongst integral formulation, in this paper we consider the electric-field integral equation which expresses the electric field $\mathbf{E}$ outside the object in terms of the unknown induced current J, and reads

$$\mathbf{E}(x) = -\int_{\Gamma} \nabla G(x,x')\rho(x')d^3x' - \frac{ik}{c}\int_{\Gamma} G(x,x')\mathbf{J}(x')d^3x' + \mathbf{E}^e(x)$$

where $\mathbf{E}^e$ is the electric field due to external sources, and G is the Green's function for scattering problems:

$$G(x,x') = \frac{e^{-ik|x-x'|}}{|x-x'|} \quad . \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 977–984, 2004.
© Springer-Verlag Berlin Heidelberg 2004

For homogeneous or layered homogeneous dielectric bodies, the Method of Moments can be used to discretize integral equations on the surface of the object or at the discontinuous interfaces between two different materials, and gives rise to a linear system of the form

$$Ax = b \tag{2}$$

where the coefficient matrix A is dense, complex, symmetric but non-Hermitian, and the unknowns are the coefficients of the expansion of the current $\mathbf{J}$ into a set of basis functions (the Rao-Wilton-Glisson basis functions). Direct methods are not feasible for solving systems (2) because they require storage of n^2 single or double precision complex entries of A and $\mathcal{O}(n^3)$ floating-point operations to compute the factorization. The use of preconditioned Krylov solvers can be a viable alternative, provided we have fast matrix-vector products and robust preconditioners. Active research efforts have been recently devoted to *hierarchical methods* [4,6] to perform fast matrix-vector products with $\mathcal{O}(n \log(n))$ computational complexity, including strategies for parallel distributed memory implementations. In this paper we focus on the other main component of Krylov methods in this context, that is the design of robust and parallel preconditioners. In Sect. 2, we describe a Frobenius-norm minimization preconditioner with a pattern prescribed in advance and we introduce a two-level iterative scheme based on the GMRES method and combined with multipole techniques. In Sect. 3, we illustrate the effectiveness of this scheme for the solution of very large problems up to one million unknowns. Finally, in Sect. 4 we present some conclusions from the work.

2 A Two-Level Iterative Scheme Combined with Multipole Techniques

The design of robust preconditioners for boundary integral equations can be challenging. Simple preconditioners like the diagonal, block diagonal or a band are effective only when the discretization matrix of system (2) has some degree of diagonal dominance, and incomplete factorization methods may suffer of ill-conditioning of the triangular factors because of the indefiniteness of A [3]. Approximate inverse techniques are less prone to instabilities. Owing to the exponential decay of the Green's function(1), the location of the large entries in A^{-1} exhibit some structure, and a very small number of entries have large magnitude compared to the others that are much smaller. Thus a very sparse matrix is likely to retain the most relevant contributions to the exact inverse. When fast methods are used for the matrix-vector products, the coefficient matrix is not completely stored in memory and only some of the entries, corresponding to the near-field interactions, are explicitely computed and available for the construction of the preconditioner. These methods are often implemented in parallel, partitioning the domain among different processors. Thus, parallelism is a relevant factor to consider in the design of the preconditioner.

In this section we describe an embedded iterative scheme based on Krylov solvers and implemented in the context of the Fast Multipole Method (FMM) [4,

6]. In our scheme, we carry out a few steps of an inner Krylov method for the preconditioning operation. The overall algorithm results in an inner-outer solution scheme and is sketched in Fig. 1. The efficiency of our approach relies on two factors, that is: the matrix-vector products within the inner and the outer solvers are carried out at different accuracy, and the inner solver itself is preconditioned so that the residual in the inner iterations can be significantly reduced in a few number of steps. The desirable feature of using different accuracy for the matrix-vector products is enabled by the use of the FMM. In our scheme, highly accurate FMM is used within the outer solver that is used to actually solve the linear system, and a lower accurate FMM within the inner solver that is used as preconditioner for the outer scheme. More precisely, the relative error in the matrix-vector computation is around $5 \cdot 10^{-4}$ compared to the exact computation for the outer iteration and around 10^{-3} for the inner iteration. In fact, we solve a nearby system for the preconditioning operation.

Outer solver $\longrightarrow$ FGMRES, FQMR

Do k=1,2, ...
- M-V product: FMM with high accuracy
- Preconditioning : **Inner solver** (GMRES, TFQMR, ...)

  Do i=1,2, ...
  - M-V product: FMM with low accuracy
  - Preconditioning

  End Do

End Do

Fig. 1. Inner-outer solution schemes in the FMM context

The preconditioner M for the inner solver is a sparse approximate inverse based on Frobenius-norm minimization. It is computed by minimizing $\|I-MA\|_{\mathrm{F}}$ (or $\|I-AM\|_{\mathrm{F}}$ for right preconditioning) subject to certain sparsity constraints, and can be obtained by solving n independent linear least-squares problems, one for each row (or column) of the preconditioner. The independence of these least-squares problems follows immediately from the identity:

$$\|I - MA\|_{\mathrm{F}}^2 = \|I - AM^T\|_{\mathrm{F}}^2 = \sum_{j=1}^{n} \|e_j - Am_{j\bullet}\|_2^2 \tag{3}$$

where e_j is the jth canonical unit vector and $m_{j\bullet}$ is the column vector representing the jth row of M; an analogous relation holds for right preconditioning. Clearly, there is considerable scope for parallelism in this approach. The cost for constructing M is significantly reduced when M is computed using as input a sparse approximation $\tilde{A}$ of the dense coefficient matrix A. If the sparsity pattern

of M is known in advance, the nonzero structure for the jth column of M is automatically determined, and defined as $J = \{i \in [1,n] \, s.t. \, (i,j) \in S\}$. The least-squares solution involves only the columns of $\tilde{A}$ indexed by J; we indicate this subset by $\tilde{A}(:, J)$. When $\tilde{A}$ is sparse, many rows in $\tilde{A}(:, J)$ are usually null, not affecting the solution of the least-squares problems (3). Thus if I is the set of indices corresponding to the nonzero rows in $\tilde{A}(:, J)$, and if we define by $\hat{A} = \tilde{A}(I, J)$, by $\hat{m}_j = m_j(J)$, and by $\hat{e}_j = e_j(J)$, the actual "reduced" least-squares problems to solve are

$$min \|\hat{e}_j - \hat{A}\hat{m}_j\|_2, j = 1, ..., n. \tag{4}$$

Usually problems (4) have much smaller size than problems (3) and can be effectively computed by dense QR factorization.

The main issue in the construction of the preconditioner is the selection of the sparsity pattern for $\tilde{A}$ and M. Here we propose to use *geometric* information from the mesh, that is the coordinates of the nodes describing geometric neighbourhoods amongst the edges. For each edge in the mesh, we consider all those edges within a sufficiently large geometric neighbourhood.

In Table 1 we show some numerical results to assess the performance of the proposed Frobenius-norm minimization approach, referred to as FROB, compared with some classical preconditioners on a typical model problem, a sphere of size 1080. These are: diagonal scaling, SSOR and ILU(0) applied to a sparse approximation $\tilde{A}$ of A constructed using the geometric approach. All these preconditioners have roughly the same number of nonzeros; the relative density of M is around 1.50%. We do not report on the AINV [1] and on the SPAI [7] preconditioners because the results are disappointing. As shown in this table, especially when the preconditioner is very sparse the Frobenius-norm minimization method combined with geometric information from the mesh can be very robust.

Table 1. Number of iterations required by different preconditioned Krylov solvers to reduce the residual by 10^{-5}

Precond.	GMRES(50)	Bi-CGSTAB	UQMR	TFQMR
Unprec	61	75	69	40
M_j	67	80	71	46
SSOR	77	82	80	55
$ILU(0)$	+500	+500	284	217
FROB	14	10	19	10

In the fast multipole algorithm, the entire object is enclosed in a cube, the cube is divided into eight subcubes and each subcube is recursively divided until the size of the smallest boxes (referred to as *leaf-boxes*) is generally half of a wavelength. In the FMM, we adopt the following criterion for the computation

of the patterns: the nonzero structure of each column of the preconditioner is defined by retaining all the edges within a given leaf-box and those in one level of neighbouring boxes, and the structure for the sparse approximation $\tilde{A}$ of the dense coefficient matrix is defined by retaining the entries associated with edges included in the given leaf-box as well as those belonging to two levels of neighbours. The preconditioner has a sparse block structure; each block is dense and is associated with one leaf-box. The least-squares problems corresponding to edges within the same box are identical because they are defined using the same nonzero structure and the same entries of A. It means that we only have to compute one QR factorization per leaf-box; on parallel computer architectures, disjoint subsets of leaf-boxes are assigned to different processors and the least-squares problems are solved independently on each processor.

3 Numerical Results

In this section we show results on the numerical scalability of the proposed scheme on two test examples, that is a sphere of radius 1 metre and an Airbus aircraft, that represents a real life model problem in an industrial context. The surface of the object is always discretized using ten points per wavelength, and larger discretizations are obtained by increasing the value of the frequency. We consider FGMRES [8] as the outer solver with an inner GMRES iteration [9] preconditioned with the Frobenius-norm minimization method described in the previous section. For the FGMRES method we consider the implementations described in [5]. The preconditioner has on the average 250 nonzeros per column on all the test examples.

In Tables 2-3 we show the number of inner/outer matrix-vector products needed to achieve convergence using a tolerance of 10^{-2} on the norm-wise backward error $\frac{||r||_2}{||b||_2}$, where r denotes the residual and b is the right-hand side of the linear system. This tolerance is accurate for engineering purposes, as it enables us to detect correctly the radar cross section of the object. We checked at the end of the computation that the radar cross section of the obstacle was accurately determined. Experiments using a lower tolerance and a larger set of geometries can be found in [2]. In order to assess the robustness of the proposed scheme, for comparison we also report on results using restarted GMRES, preconditioned with the same preconditioner used for the inner GMRES iterations in the two-level scheme. The comparison is fair because GMRES(30) has exactly the same storage requirements as the combination FGMRES(5)/GMRES(20). In fact, for the same restart value, the storage requirement for the FGMRES algorithm is twice that for the standard GMRES algorithm, as it stores the preconditioned vectors of the Krylov basis. We also give timings for runs in single precision arithmetic on eight processors of a Compaq Alpha server. The Compaq Alpha server is a cluster of Symmetric Multi-Processors. Each node consists of four Alpha processors that share 512 Mb of memory and around 189Gb of temporary disk space.

On the sphere, the number of iterations required by GMRES(30) is nearly constant for small problems, but increases for larger problems. The combination FGMRES/GMRES remarkably enhances the robustness of the preconditioner. On the sphere with 367500 points, it enables convergence in 16 outer and 252 total inner iterations whereas GMRES(30) does not converge in 1500 iterations due to interior cavity resonances. On the sphere with one million unknowns the elapsed time for the iterative solution is reduced from one day to slightly more than 3 hours on 8 processors. The enhancement of the robustness of the preconditioner is even more significant on the Airbus aircraft as GMRES(30) does not converge in 1500 iterations on problem sizes larger than around 250000 unknowns.

Table 2. Total number of matrix-vector products and timing required to converge on 8 processors on a Compaq machine on a sphere

Size of the linear system	FGMRES(5)	GMRES(20)	Solution time (sec)	GMRES(30)	Solution time (sec)
71148	7	105	4 mins	83	3 mins
161472	9	126	13 mins	96	8 mins
288300	13	210	37 mins	348	1 hour
367500	16	252	1 h 10 mins	+1500	+8 hours
549552	17	260	1 h 50 mins	532	4 hours
1023168	17	260	3 h 20 mins	1196	1 day

Table 3. Total number of matrix-vector products and timing required to converge on 8 processors on a Compaq machine on an aircraft

Size of the linear system	FGMRES(5)	GMRES(20)	Solution time (sec)	GMRES(30)	Solution time (sec)
23676	15	220	7 mins	61	3 mins
94704	7	100	9 mins	101	13 mins
213084	11	160	36 mins	225	1 hour
591900	17	260	3 h 25 mins	+1500	4 hours
1160124	19	300	8 h 42 mins	+1500	1 day

The convergence history of GMRES depicted in Fig. 2 for different values of the restart gives us some clues to the numerical behaviour of the proposed scheme. The residual of GMRES tends to decrease very rapidly in the first few iterations independently of the restarts, then decreases much more slowly, and finally stagnates to a value that depends on the restart; the larger the restart, the lower the stagnation value. It suggests that a few steps (up to 20) in the inner solver can be very effective for obtaining a significant reduction of the initial residual. Figure 2 also shows that large restarts of GMRES do not enable a further reduction of the normwise backward error in the beginning of convergence. Thus small restarts should be preferred in the inner GMRES iterations.

Table 4. Experiments on the parallel scalability of the code on the Airbus aircraft

Problem size	Nb procs	Construction time (sec)	Elapsed time precond (sec)	Elapsed time mat-vec (sec)
112908	8	513	0.39	1.77
221952	16	497	0.43	2.15
451632	32	509	0.48	2.80
900912	64	514	0.60	3.80

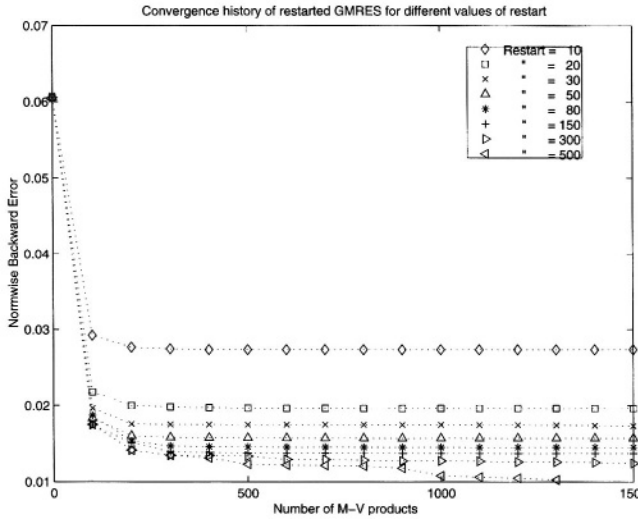


Fig. 2. Convergence history of restarted GMRES for different values of restart on an aircraft with 94704 unknowns

Finally, in Table 4, we show the parallel scalability of the implementation of the preconditioner in the FMM code. We solve problems of increasing size on a larger number of processors, keeping the number of unknowns per processor constant. It can be seen that the construction of the preconditioner scales perfectly; its application requires some communication but still scales reasonably well. The scalability of the mat-vec is also satisfactory as the increase of the elapsed time is not only due to the amount of data exchanges but also to the $log(n)$ effect of its complexity. We refer to [10] for a complete description of the parallel code that we used.

4 Conclusions

In this paper, we have proposed a robust preconditioned iterative scheme based on the GMRES solver and combined with multipole techniques for the solution of very large dense systems in electromagnetism. We have shown that the proposed

scheme can afford the solution of very large problems at reduced computational cost and storage requirement. Most of the experiments shown in this paper require a huge amount of computation and storage, and they often reach the limits of our target machine in terms of Mbytes. For the solution of systems with one million unknowns direct methods would require eight Tbytes of storage and more than thirty years of computation on one processor of the target computer, assuming that the computation runs at peak performance.

References

1. M. Benzi, C.D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Scientific Computing,* 17:1135–1149, 1996.
2. B. Carpentieri, I. S. Duff, L. Giraud and G. Sylvand. Combining fast multipole techniques and an approximate inverse preconditioner for large parallel electromagnetism calculations. Technical Report TR/PA/03/77, CERFACS, Toulouse, France, 2003.
3. B. Carpentieri. *Sparse preconditioned for dense linear systems from electromagnetic applications.* PhD thesis, CERFACS, Toulouse, France, 2002.
4. E. Darve. The fast multipole method: Numerical implementation. *J. Comp. Phys.,* 160(1):195–240, 2000.
5. V. Frayssé, L. Giraud, and S. Gratton. A set of Flexible-GMRES routines for real and complex arithmetics. Technical Report TR/PA/98/20, CERFACS, Toulouse, France, 1998.
6. L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics,* 73:325–348, 1987.
7. M. Grote and T. Huckle. Parallel preconditionings with sparse approximate inverses. *SIAM J. Scientific Computing,* 18:838–853, 1997.
8. Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Scientific and Statistical Computing,* 14:461–469, 1993.
9. Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific and Statistical Computing,* 7:856–869, 1986.
10. G. Sylvand. *La Méthode Multipôle Rapide en Electromagnétisme : Performances, Parallélisation, Applications.* PhD thesis, Ecole Nationale des Ponts et Chaussées, 2002.

Cholesky Factorization of Matrices in Parallel and Ranking of Graphs*

Dariusz Dereniowski and Marek Kubale

Department of Algorithms and Modeling of Systems,
Gdańsk University of Technology, Poland,
kubale@eti.pg.gda.pl

Abstract. The vertex ranking problem is closely related to the problem of finding the elimination tree of minimum height for a given graph. This implies that the problem has applications in the parallel Cholesky factorization of matrices. We describe the connection between this model of graph coloring and the matrix factorization. We also present a polynomial time algorithm for finding edge ranking of complete bipartite graphs. We use it to design an $O(m^{2+d})$ algorithm for edge ranking of graphs obtained by removing $O(\log m)$ edges from a complete bipartite graph, where d is a fixed number. Then we extend our results to complete k-partite graphs for any fixed $k > 2$. In this way we give a new class of matrix factorization instances that can be optimally solved in polynomial time.

1 Introduction

A k-ranking of the vertices of $G = (V, E)$ where $|V| = n, |E| = m$ is a labeling of its vertices with integers $1, \ldots, k$ such that every path between vertices with the same color contains a vertex with a greater color. If k is the smallest integer such that G has k-ranking then this number is denoted by $\chi_r(G)$. Edge ranking of graph G is a labeling of edges of G such that every path between two edges with color i contains an edge with color $j > i$. $\chi'_r(G)$ is used to denote the minimum k such that G has an edge ranking with k colors.

Vertex ranking and edge ranking problems are interesting because of their potential applications. Rankings are important in computing Cholesky factorizations of matrices in parallel [8] and in VLSI-layout [7]. Edge ranking of trees has applications in modeling the parallel assembly of a product from its components [3]. Also, vertex ranking problem is equivalent to the problem of finding minimum height elimination tree of a graph.

Pothen [10] proved that vertex ranking problem is NP-hard. It remains NP-hard even if restricted to bipartite and cobipartite graphs [1]. On the other hand, efficient vertex ranking algorithms for some classes of graphs are known. An $O(n^3\chi_r(G)^{3d-3})$ algorithm for vertex ranking of d-trapezoid graphs has been presented in [2]. It has running time $O(n^3)$ for interval graphs and $O(n^3\chi_r(G)^3)$

* Supported in part by KBN grant 4T11C 04725

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 985–992, 2004.
© Springer-Verlag Berlin Heidelberg 2004

for permutation graphs. In the same paper an $O(n^3)$ algorithm for vertex ranking of circular-arc graphs has been given. There exists a linear time algorithm for trees [11] and the problem is solvable in polynomial time for graphs with treewidth at most k, where k is a fixed integer [1].

Lam and Yue showed in [5] that the general problem of finding an optimal edge ranking of graphs is NP-hard. In [1] formula for computing $\chi'_r(K_n)$ for a complete graph K_n has been presented. Lam and Yue have described a linear time algorithm for edge ranking of trees [6].

In the next section we describe the connection between the vertex ranking problem and the problem of finding the minimum height elimination tree of a graph. We list some facts to show that the vertex ranking problem can be used in the parallel Cholesky factorization of matrices. Section 3 gives polynomial time algorithms for the edge ranking problem if restricted to some special classes of bipartite graphs. In particular, an $O(m^2)$ algorithm for complete bipartite graphs is described and an $O(m^{2+d})$ algorithm for graphs obtained by removing $O(\log m)$ edges from a complete bipartite graph, where d is a nonnegative constant number. We prove that these results can be generalized to complete k-partite graphs, where $k > 2$ is a fixed integer. In this way we give a new class of dense graphs, namely line graphs of complete and nearly complete k-partite graphs, that can be labeled in polynomial time.

2 Cholesky Factorization of Matrices in Parallel

Let A be an $n \times n$ symmetric positive definite matrix. Consider a linear system of the form

$$Ax = b. \tag{1}$$

There exists a lower triangular matrix L such that

$$A = LL^T.$$

We obtain vector x by solving the following two linear systems:

$$Ly = b,$$

$$L^T x = y.$$

Note that if P is a permutation matrix then we can rewrite the linear system (1) to the form

$$PAP^T(Px) = Pb,$$

thus the matrix A can be reordered before the factorization is performed. Let us define graph $G(A)$ as follows:

$$V(G(A)) = \{1, \ldots, n\},$$

$$E(G(A)) = \{ij | A_{ij} \neq 0\}.$$

Let $p_1, \ldots, p_n$ be a permutation of the vertices of $G(A)$. To create the graph $F(A)$ repeat the following step for each $i = 1, \ldots, n$: add edges to G, such that neighbors of p_i in the set $\{p_{i+1}, \ldots, p_n\}$ form a complete subgraph.

Definition 1. *Vertex p_i is the parent of a vertex p_j in the elimination tree if and only if $i = \min\{k | k > j$ and $p_k p_j \in E(F(A))\}$. The height of an elimination tree is the length of the longest path from the root to a leaf and is denoted by $h(T)$. Symbol $h(G)$ is used to denote the height of the elimination tree whose height is minimum.*

An elimination tree describes the dependencies between the columns of the lower triangular matrix during the Cholesky factorization, i.e if a vertex a is a descendant of a vertex b then numeric values in column b depend on values in column a, thus column a has to be computed before b [8].

The vertex ranking problem is closely related to the the problem of finding the elimination tree of minimum height. If T is an elimination tree of G then the vertex ranking of G can be obtained by assigning color i to the vertices in the i-th level in T, where level $h(T)+1$ is the root. Let z be the root of a subtree of T. If $c(x) = c(y)$ where $x, y \in T[z]$ and no z is the closest common ancestor of x and y then x and y are not adjacent in graph F because otherwise (assume that x is ordered before y) y is an ancestor of x in T and cannot have the same color as x - contradiction. So if $S = \{z_0 = z, \ldots, z_p = \text{root}(T)\}$ is the path from z to the root of T then S is a vertex separator, such that x and y belong to different components of the graph $G - S$. This means that each connected component of induced subgraph $G[\{v | c(v) \leq i\}]$ contains at most one vertex v such that $c(v) = i$, thus coloring c is a valid vertex ranking of G. Now let us assume that c is k-ranking of G. We can obtain an elimination tree whose height is at most $k-1$ by choosing the following ordering of the vertices of G:

$$c^{-1}(1), \ldots, c^{-1}(k), \tag{2}$$

where $c^{-1}(i)$ is the set of vertices of G with color i under c. In this tree a node with color i is not a descendant of a node with color $j < i$. Suppose that $c(v) = c(u)$ and v is the parent of u. In this case u and v are adjacent in F which means that there exist a path between u and v in G containing vertices labeled with colors $\{1, \ldots, c(u)\}$, which is impossible because c is a proper vertex ranking of G. Thus the vertices with the same color are unrelated in T. This means that the elimination tree obtained by the permutation (2) has height at most $k-1$.

If c is a k-ranking of G then we can compute the lower triangular matrix in k steps using $\max\{|c^{-1}(1)|, \ldots, |c^{-1}(k)|\}$ processors. As an example consider the following matrix.

$$A = \begin{bmatrix} a & * & * & & & * \\ * & b & * & & & \\ * & * & c & * & & \\ & & * & d & * & * \\ & & & * & e & * \\ * & & & * & * & f \end{bmatrix}$$

The symbol * was used to denote its nonzero elements. The adjacency graph G for matrix A is shown in Fig. 1(a). Fig. 1(b) gives an optimal vertex ranking of

G with four colors, which means that the factorization can be performed in four steps when two processors are used for the computation.

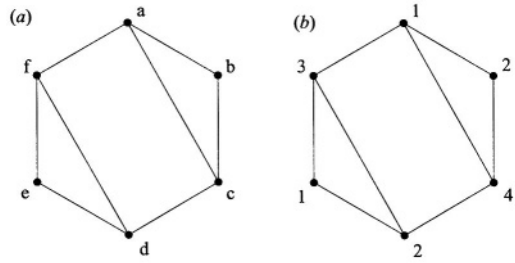


Fig. 1. (a) the adjacency graph G for matrix A; *(b)* optimal ranking of G.

Fig. 2 presents the elimination tree corresponding to the vertex ranking in Fig. 1*(b)*, i.e. the elimination ordering was created on the basis of (2).

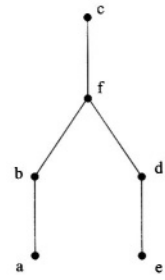


Fig. 2. An elimination tree of the graph in Fig. 1(a).

3 Complete Bipartite Graphs

In this chapter we consider some classes of graphs for which the vertex ranking problem is polynomially solvable, namely line graphs of complete k-partite graphs and line graphs of almost complete k-partite graphs. For convenience, we will consider the edge ranking problem of graphs instead of the vertex ranking problem of the line graphs.

Let $K_{a,b}$ be a complete bipartite graph. We denote the vertices in $V_1(K_{a,b})$ by $v_1^a, \ldots, v_a^a$ and in $V_2(K_{a,b})$ by $v_1^b, \ldots, v_b^b$. $S_{a_1,b_1}^{a,b}$ where $a_1 \leq a, b_1 \leq b$ denotes the set of edges joining vertices $v_1^a, \ldots, v_{a_1}^a$ to $v_{b_1+1}^b, \ldots, v_b^b$ and $v_{a_1+1}^a, \ldots, v_a^a$ to $v_1^b, \ldots, v_{b_1}^b$.

Lemma 1. *Let c be an edge ranking of $K_{a,b}$ using k colors. Then there exists an edge ranking c' using the same k colors, and positive integers $a_1 \leq a$ and $b_1 \leq \lfloor b/2 \rfloor$ such that colors $k - |S^{a,b}_{a_1,b_1}| + 1, \ldots, k$ are assigned to the edges in $S^{a,b}_{a_1,b_1}$.*

Proof. The edges with unique labels under c form an edge-separator S in $K_{a,b}$. Consider the graph $G = (V(K_{a,b}), E(K_{a,b}) - S^{a,b}_{a_1,b_1})$, which is disconnected. Let G' denote a connected subgraph of G containing a_1 vertices from set $\{v^a_1, \ldots, v^a_a\}$ and b_1 vertices from set $\{v^b_1, \ldots, v^b_b\}$. Since G is disconnected, $G' \neq G$. Edges between vertices in $V(G')$ and $V(G) - V(G')$ get unique colors under c, so we can shuffle the labels on the edges of S so that the edges in $S^{a,b}_{a_1,b_1}$ receive the biggest labels. This results in an edge ranking c'. The set of edges between $V(G')$ and $V(G) - V(G')$ is equal to $S^{a,b}_{a_1,b_1}$ and $|c'(K_{a,b})| \leq |c(K_{a,b})|$. Note that if separator $S^{a,b}_{a_1,b_1}$, where $b_1 > \lfloor b/2 \rfloor$, leads to the optimal solution then we can also find optimal solution using separator $S^{a,b}_{a-a_1,b-b_1}$. We have $b - b_1 \leq \lfloor b/2 \rfloor$ so we can assume that $b_1 \leq \lfloor b/2 \rfloor$. This completes the proof. □

Theorem 1. *There exists an $O(m^2)$ algorithm for finding an edge ranking of complete bipartite graphs with m edges.*

Proof. The procedure described below computes the edge ranking number for a complete bipartite graph $K_{a,b}$, where $a \leq b$.

```
procedure Complete-Bi-Ranking(K_{a,b})
begin
  for j := 0 to b do begin
    Opt[0, j] := j;
    Opt[1, j] := j;
  end;
  for i := 2 to a do
    for j := i to b do begin
      Opt[i, j] := +∞;
      for k := 0 to i do
        for l := 0 to ⌊j/2⌋ do begin
          r1 := Opt[min{k, l}, max{k, l}];
          r2 := Opt[min{i − k, j − l}, max{i − k, j − l}];
          r := k(j − l) + (i − k)l + max{r1, r2};
          if r < Opt[i, j] then
            Opt[i, j] := r;
        end
    end
  return Opt[a, b];
end
```

Array *Opt* contains edge ranking numbers of subgraphs $K_{i,j}$, $i \leq a$, $j \leq b$, so we have to compute the edge ranking number only once for each graph $K_{i,j}$.

Lemma 1 implies that to compute $\chi'_r(K_{i,j})$, $i \leq j$ we have to check the following separators: $S^{i,j}_{k,l}$, $k = 0, \ldots, i$, $l = 0, \ldots, \lfloor j/2 \rfloor$. In order to compute edge ranking of $K_{a,b}$ we have to use another array to store the values of k, l for each i, j such that separator $S^{i,j}_{k,l}$ is optimal for finding edge ranking of $K_{i,j}$. This completes the proof of the correctness of the algorithm. Clearly, the algorithm has running time $O((ab)^2) = O(m^2)$. □

Theorem 2. *Let $K_{a,b}$ be a complete bipartite graph, let c be a nonnegative integer and let $E^{a,b}_c$ denote any set of edges such that $E^{a,b}_c \subset E(K_{a,b})$, $|E^{a,b}_c| \leq c$. If c is a constant number then there exists an $O(t^c m^2)$ algorithm to compute edge ranking of graph $K_{a,b} - E^{a,b}_c$, where $t > 1$ is a constant number.*

Proof. Let us consider a set $E^{a,b}_c$. Let

$$\{v_1^{c_a}, \ldots, v_{c_a}^{c_a}\} = \{v \in V_1(K_{a,b}) \mid \deg(v) < b\},$$

$$\{v_1^{c_b}, \ldots, v_{c_b}^{c_b}\} = \{v \in V_2(K_{a,b}) \mid \deg(v) < a\}.$$

Let us consider fixed values of variables i, j, k, l in the procedure given in the proof of Theorem 1. Let G' be a graph induced by i vertices in set $V_1(K_{a,b} - E^{a,b}_c)$ and j vertices in $V_2(K_{a,b} - E^{a,b}_c)$. If $v_1, v_2 \in V_1(G'), \deg(v_1) = \deg(v_2) = j$ and $U = V(G') \setminus \{v_1, v_2\}$, then graphs induced by vertices $U \cup \{v_1\}$ and $U \cup \{v_2\}$ are isomorphic. Thus to find edge ranking of all subgraphs containing i vertices from $V_1(K_{a,b} - E^{a,b}_c)$ and j vertices from $V_2(K_{a,b} - E^{a,b}_c)$ we have to consider all subsets of type $\{v_1^{c_a}, \ldots, v_{c_a}^{c_a}, v_1^{c_b}, \ldots, v_{c_b}^{c_b}\}$ in $K_{i,j}$. Therefore there are at most $2^{c_a} 2^{c_b}$ cases to consider. Since the cardinality of the separator $S^{i,j}_{k,l}$ can be computed in linear time so for fixed values of variables i, j, k, l the running time of the algorithm is $O(c 2^{2c_a} 2^{2c_b}) = O(t^c)$. Thus, $O(t^c m^2)$ is the complexity of the procedure similar to the procedure described in the proof of Theorem 1. The dimension of array *Opt* is $a \times b \times 2^{c_a} \times 2^{c_b}$, which is a polynomial if c is constant. □

From Theorem 2 we obtain the following

Corollary 1. *There exists an $O(m^{2+d})$ algorithm for the edge ranking problem of almost complete bipartite graphs $K_{a,b} - E^{a,b}_{d \log(m)}$, where $d \geq 0$ is a constant number.*

The above results can be extended to complete k-partite graphs for $k > 2$. Let $K_{s_1,\ldots,s_k}$ denote the complete k-partite graph. Define a set of edges

$$S^{s_1,\ldots,s_k}_{p_1,\ldots,p_k} = \{v_q^i v_r^j | i, j = 1, \ldots, k, i \neq j, q = 1, \ldots, p_i, r = p_j + 1, \ldots, s_j\}.$$

Note that

$$|S^{s_1,\ldots,s_k}_{p_1,\ldots,p_k}| = \sum_{i=1}^{k} \sum_{j=1, j\neq i}^{k} p_i(s_j - p_j).$$

Lemma 2. *If c is an edge ranking of $K_{s_1,\ldots,s_k}$ using l colors then there exists an edge l-ranking c' and positive integers $0 \leq p_i \leq s_i$, $i = 1,\ldots,k$ such that unique colors $l - |S^{s_1,\ldots,s_k}_{p_1,\ldots,p_k}| + 1,\ldots,l$ are assigned to the edges in $S^{s_1,\ldots,s_k}_{p_1,\ldots,p_k}$.*

We omit the proof because it is analogous to the proof of Lemma 1. As before, we store the values of $\chi'_r(K_{s'_1,\ldots,s'_k})$, $s'_i \leq s_i$ in k-dimensional matrix *Opt,* i.e.

$$Opt[s'_1,\ldots,s'_k] = \chi'_r(K_{s'_1,\ldots,s'_k}).$$

Consider the graph $K_{s'_1,\ldots,s'_k}$. Assume that edge rankings of all graphs $K_{s''_1,\ldots,s''_k} \neq K_{s'_1,\ldots,s'_k}$ have been already computed, $0 \leq s''_i \leq s'_i$. From Lemma 2 we have

$$\chi'_r(K_{s'_1,\ldots,s'_k}) = \min\{|S^{s'_1,\ldots,s'_k}_{p_1,\ldots,p_k}| + \max\{\chi'_r(K_{p_1,\ldots,p_k}), \chi'_r(K_{s'_1-p_1,\ldots,s'_k-p_k})\}\}, \quad (3)$$

where $p_i = 0,\ldots,s'_i, i = 1,\ldots,k$. Thus, in order to compute $\chi_r(K_{s'_1,\ldots,s'_k})$ using equation (3), we have to consider $(s'_1+1)\cdot\ldots\cdot(s'_k+1)$ separators. Note that computing an address in the *Opt* array requires time proportional to k and we can find the value of $|S^{s'_1,\ldots,s'_k}_{p_1,\ldots,p_k}|$ also in $O(k)$ time. This means that we obtain values of $|S^{s'_1\ldots,s'_k}_{p_1,\ldots,p_k}|$, $\chi'_r(K_{p_1,\ldots,p_k})$ and $\chi'_r(K_{s'_1-p_1,\ldots,s'_k-p_k})$ performing $O(k)$ operations in the main loop. So the running time of the algorithm is

$$O(k(s_1\cdot\ldots\cdot s_k)^2) = O(km^{2\lceil k/2\rceil}).$$

In this way we obtained a polynomial time algorithm for computing the edge ranking of complete k-partite graph, where $k \geq 2$ is fixed. This algorithm can be extended to the algorithm for computing the edge ranking of graphs obtained by removing $O(\log m)$ edges from a complete k-partite graph. On the basis of the discussion above we can write the following theorem.

Theorem 3. *Let k be a fixed number. There exists a polynomial time algorithm with running time $O(km^{2\lceil k/2\rceil})$ for computing edge ranking of complete k-partite graph. If G is a graph obtained from the complete k-partite graph by removing $O(\log m)$ edges then $\chi'_r(G)$ can be computed in polynomial time.*

4 Conclusions

We discussed the vertex ranking problem and its application in the parallel Cholesky factorization of matrices. In the case of the line graph of a complete bipartite graph we can find all minimal separators such that one of them contains vertices with unique colors in an optimal ranking. This property and dynamic programming leads to a polynomial time algorithm for edge ranking of complete bipartite graphs. We used this procedure to design algorithm for "almost" complete bipartite graphs. The results have been extended to complete k-partite graphs, where $k > 2$ is fixed. It is interesting if there exist faster algorithms for these problems and which other classes of graphs have polynomial time algorithms for the edge ranking problem.

References

1. H. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, Z. Tuza, Rankings of graphs, *SIAM J.Discrete Math.* **11** (1998) 168-181.
2. J.S. Deogun, T. Kloks, D. Kratsch, H. Müller, On the vertex ranking problem for trapezoid, circular-arc and other graphs, *Discrete Appl. Math.* **98** (1999) 39-63.
3. A.V. Iyer, H.D. Ratliff, G. Vijayan, Parallel assembly of modular products - an analysis, *Tech. Report* 88-06, Georgia Institute of Technology, 1988
4. M. Katchalski, W. McCaugh, S. Seager, Ordered colourings, *Discrete Math.* **142** (1995) 141-154.
5. T.W. Lam, F.L. Yue, Edge ranking of graphs is hard, *Discrete Appl. Math.* **85** (1998) 71-86
6. T.W. Lam, F.L. Yue, Optimal edge ranking of trees in linear time, *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms,* (1998) 436-445.
7. C.E. Leiserson, Area-efficient graph layout (for VLSI) *Proc. 21st Ann. IEEE Symp. on Foundations of Computer Science* (1980) 270-281.
8. J.W.H. Liu, The role of elimination trees in sparse factorization, *SIAM J.Matrix Analysis and Appl.* **11** (1990) 134-172.
9. F. Manne, Reducing the height of an elimination tree through local reorderings. *Tech. Report* 51, University of Bergen, Norway, 1991.
10. A. Pothen, The complexity of optimal elimination trees, *Tech. Report* CS-88-13, The Pennsylvania State University, 1988
11. A.A. Schäffer, Optimal node ranking of trees in linear time, *Inform. Process. Lett.* **33** (1989/90) 91-96.
12. P. de la Torre, R. Greenlaw, A.A. Schäffer, Optimal edge ranking of trees in polynomial time, *Algorithmica* **13** (1995) 529-618

Parallel Square Modular Computer Algebra

Sergey A. Inutin

Surgut State Pedagogic University, 50 Let VLKSM, 10/2,
628413 Surgut, Russia inyutin@surguttel.ru

Abstract. The computer algebra of parallel modular operations with a square diapason for a variable is described. The base set of the algebra is a finite dimension metric space of modular integer vectors. Two metrics are introduced. An orthogonal normal basis is employed to reconstruct the value of the integer number corresponding to the vector. An analog of the inner product is used to advance beyond the additive range, and the vector product is defined in two ways. The algebra could serve as the basis for parallel computer arithmetic of unbounded digit integers, a theoretical foundation of parallel computing.

1 Introduction

Parallel computer algorithms are a good source of speed increase for electronic computational devices [1], [2]. The paper describes the computer algebra of parallel modular operations with a square diapason for a variable.

2 Algebra Description

To define the parallel computer algebra we first describe the base set. The finite dimensional metric space **V** of vectors of modular components is defined in the following way. The elements of the space are the vectors of modular components $\bar{a} = (\alpha_1, ..., \alpha_n) = ((\bar{a} \cdot \bar{e_1}), ..., (\bar{a} \cdot \bar{e_n}))$, where

$$\bar{a} \in p_1^{'} \times ... \times p_n^{'} = \prod_{i=1}^{n} p_i^{'}, \tag{1}$$

$\alpha_i = |A|_{p_i^2} \subset p_i^{'} = \{0, ..., p_i, p_i^2 - 1\}$, and $(\bar{a} \cdot \bar{e}_i)$ is the inner product of the vectors $\bar{a}$ and $\bar{e}_i$, i = 1, . . . , n. The components of the vector $\bar{a}$ are the residues of mod p_i^2 of a number $A \in N$, and $\{\bar{a}\} \leftrightarrow \{A\}$. The vector space is linear, i.e.

$$\bar{c} = \mu \cdot \bar{a} + \nu \cdot \bar{b} = \left(..., |\mu \cdot \alpha_i + \nu \cdot \beta_i|_{p_i^2}, ...\right) \in \prod_{i=1}^{n} p_i^{'}. \tag{2}$$

The first inner product is defined as $\left(\bar{a} \cdot \bar{b}\right) = \sum_{i=1}^{n} \alpha_i \beta_i$. It has the following properties: $(\bar{a} \cdot \bar{a}) \geq 0$, $(\bar{a} \cdot \bar{a}) = 0 \Rightarrow \bar{a} = (0, ..., 0)$, $\left(\bar{a} \cdot \bar{b}\right) = \left(\bar{b} \cdot \bar{a}\right)$, and $\left((\bar{a} + \bar{c}) \cdot \bar{b}\right) = \left(\bar{a} \cdot \bar{b}\right) + (\bar{a} \cdot \bar{c})$.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 993–997, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The characteristic of a vector is defined as its modulus:

$$(\bar{a}\cdot\bar{a})=\sum_{i=1}^{n}\alpha_i^2=|\bar{a}|^2. \tag{3}$$

Note that the first inner product defines a metric but not a norm because $(\lambda\bar{a}\cdot\bar{a})\leq\lambda(\bar{a}\cdot\bar{a})$.

The second inner product

$$\left(\bar{a}\cdot\bar{b}\right)_2\equiv\sum_{i=1}^{n}|\alpha_i|_{p_i^2}|\beta_i|_{p_i^2} \tag{4}$$

has the following properties: $\left(\bar{a}\cdot\bar{b}\right)_2\leq n\left(p_i^2-1\right)^2$ and $\left(\bar{a}\cdot\bar{b}\right)_2=\sum\limits_{i=1}^{n}|\alpha_i|_{p_i^2}^2$ for $A=B\left(\mathrm{mod}\,P^2\right)$. Also, from $\alpha_i,\beta_i\in p_i^{'}$ follows $\left(\bar{a}\cdot\bar{b}\right)=\left(\bar{a}\cdot\bar{b}\right)_2$.

The first (Euclidean) distance is defined as

$$\varepsilon(A,B)=\sqrt{\sum_{i=1}^{n}(\alpha_i-\beta_i)^2}=\sqrt{(A-B,A-B)}. \tag{5}$$

The second modular distance is defined as

$$l(A,B)=\sqrt{\sum_{i=1}^{n}|\alpha_i-\beta_i|_{p_i^2}^2}=\sqrt{\left(|A-B|_{p_i^2},|A-B|_{p_i^2}\right)}. \tag{6}$$

The second modular distance is a metric, because $l(A,B)\geq 0$; from $l(A,B)=0$ follows $A=B\left(\mathrm{mod}\,P^2\right)$; $l(A,B)=l(B,A)$, and the triangle inequality is satisfied.

Residue distance can be very useful in construction of modular error control codes. Analogously to the Hemming distance, the residue distance between two vectors is defined as $\omega\left(\bar{a},\bar{b}\right)=\sum\limits_{i=1}^{n}\delta\left(|\alpha_i-\beta_i|_{p_i^2}\right)$, where

$$\delta\left(|\alpha_i-\beta_i|_{p_i^2}\right)=\begin{cases}1, & \text{if } \alpha_i\neq\beta_i\\ 0, & \text{if } \alpha_i=\beta_i\end{cases},\ i=1,...,n. \tag{7}$$

The residue distance is a metric because $\omega\left(\bar{a},\bar{b}\right)\geq 0$, $\omega\left(\bar{a},\bar{b}\right)=\omega\left(\bar{b},\bar{a}\right)$, and $\omega\left(\bar{a},\bar{b}\right)\geq\omega\left(\bar{a},\bar{c}\right)+\omega\left(\bar{c},\bar{b}\right)$ for any $\bar{a}$, $\bar{b}$, and $\bar{c}$.

The modular coding between the sets $\left\{0,P^2-1\right\}\subset N$ and $\{(\alpha_1,...\alpha_n)\}$, where $P^2=\prod\limits_{i=1}^{n}p_i^2$, establishes a bijection mapping.

Equivalence classes on the set of modular vectors are introduced by fixing the modulus of a modular vector. Note that the minimum increment of a vector by a unit in the sense of the Lee distance causes the change of the vector

modulus squared by an odd integer. Indeed, $|\bar{a}|^2 = \sum_{i=1}^{n} (\alpha_i^2 + 2\alpha_i + 1)$, where $\alpha_i \in \{0, ..., p_i^2 - 1\}$.

The normal vectors in the modular vector space are the only basis vectors since $|\bar{e}_i| = 1$ for them only. The modulus of a vector is equal zero for the null vector only, and the distance from the null vector to a vector is equal 1 for the basis vectors only [4], [5].

Proposition 1. *The orthogonal normal basis of the n-dimensional space with* $P^2 = \prod_{i=1}^{n} p_i^2$ *consists of the vectors* $\left\{\left(\overline{m_i P^2/p_i^2}\right)\right\}$, *where* $m_i = \left|P^2/p_i^2\right|_{p_i^2}^{-1}$. *That is, the following statements are true:*

1. $\left(\left(\overline{m_i P^2/p_i^2}\right) \cdot \left(\overline{m_j P^2/p_j^2}\right)\right) = 0$ *for all* $i \neq j$,
2. $\left(\left(\overline{m_i P^2/p_i^2}\right) \cdot \left(\overline{m_i P^2/p_i^2}\right)\right) = 0$ *for all* i, *and*
3. $p_i^2 \left(\overline{m_i P^2/p_i^2}\right) + \left(\overline{p_i^2}\right) = p_i^2 \left(\overline{1}\right)$ *for all* i.

A number $A \in N$ and a vector $\bar{a}$ are interrelated through the following:

$$A = \sum_{i=1}^{n} \left(\bar{a} \cdot \left(\overline{m_i P^2/p_i^2}\right)\right) \cdot m_i P^2/p_i^2 - rP^2, \quad A = \left(\bar{a} \cdot \bar{M}\right) - rP^2, \tag{8}$$

where r is the characteristic of A depending on the choice of the basis, $\bar{M} = \left(..., m_i P^2/p_i^2, ...\right)$.

The modular algebra is defined by the signature $(V, +, -, \cdot, \times, /, \div)$, where $V = p_1' \times ... \times p_n'$ is the basis set with the properties above described, + and – are the additive operations of component-by-component addition and subtraction by the system module, $\cdot$ is the component-by-component multiplication by the system module of the first type, $\times$ is the second type multiplication, / is the component-by-component analog of the exact division or the first type multiplication by the inverse element [3], and $\div$ is the analog of the exact division or the second type multiplication by an inverse element. To define the above-mentioned operations we need a set of n numbers (in general case, mutually prime ones), which constitute the module of the modular system.

The additive operations are defined by $\bar{a} \pm \bar{b} = \left(..., |\alpha_i \pm \beta_i|_{p_i^2}, ...\right)$.

Vector multiplication in the modular algebra could be defined in two ways. The modular multiplication of the first type is a component-by-component operation. The result of this operation is the vector $\bar{a} \cdot \bar{b} = \left(..., |\alpha_i \cdot \beta_i|_{p_i^2}, ...\right)$.

Particularly, $\left(\left(\overline{m_i P^2/p_i^2}\right) \cdot \left(\overline{m_j P^2/p_j^2}\right)\right) = 0$ for all $i \neq j$.

Vector multiplication of the second type $\bar{a} \times \bar{b}$ is defined in the standard way. The product is the vector orthogonal to the vectors-multipliers; its modulus is the product of the multipliers' module times the sin of the angle between them.

Remark 1. If $n = 3$ then for the orthogonal vectors $\left(p_i^2 - 1, 0, 0\right)$ and $\left(0, p_j^2 - 1, 0\right)$ the product is orthogonal to the vectors-multipliers, and its modulus is equal to

$$\begin{vmatrix} p_i^2 - 1 & 0 \\ 0 & p_j^2 - 1 \end{vmatrix} = \left|\left(p_i^2 - 1\right) \cdot \left(p_j^2 - 1\right)\right|_{p_i^2 p_j^2} = -p_i^2 - p_j^2 + 1. \tag{9}$$

This result is consistent with the way the nonprime bases of modular system are introduced.

Remark 2. The modules of the product vectors computed using the different multiplication definitions are not equal:

$$\sum_{i=1}^{n} |\alpha_i \cdot \beta_i|_{p_i^2}^2 \neq \sum_{i=1}^{n} \alpha_i^2 \cdot \sum_{i=1}^{n} \beta_i^2 - \left(\sum_{i=1}^{n} \alpha_i \cdot \beta_i\right)^2. \tag{10}$$

and, particularly, for $n = 2$,

$$|\alpha_1 \cdot \beta_1|_{p_i^2}^2 + |\alpha_2 \cdot \beta_2|_{p_i^2}^2 \neq \left(\alpha_2 \cdot \beta_1 - \alpha_1 \cdot \beta_2\right)^2. \tag{11}$$

The operation of the formal division of the first type is defined accordingly:

$$\bar{a}/\bar{b} = \bar{a} \cdot |\bar{b}|_{P^2}^{-1} = \left(..., \left|\alpha_i \, |\beta_i|_{p_i^2}^{-1}\right|_{p_i^2}, ...\right), \tag{12}$$

where vector $\left|\bar{b}\right|_{P^2}^{-1}$ is the one for which $\left|\left|\bar{b}\right|_{P^2}^{-1} \cdot \bar{b}\right|_{P^2} = 1$. The inverse vector is defined for every vector in V [3].

The operation of the formal division of the second type is defined as

$$\frac{\bar{a}}{\bar{b}} = \bar{a} \times \bar{b}^{-1} = \left(..., \left|\frac{\alpha_i}{\sqrt{\sum_{i=1}^{n} \beta_i^2}}\right|_{p_i^2}, ...\right). \tag{13}$$

Here vector $\bar{b}^{-1}$ is the vector orthogonal to the vector $\bar{b}$ for which the following holds:

$$\bar{b}^{-1} = \left(..., \left|\frac{1}{\sqrt{\sum_{i=1}^{n} \beta_i^2}}\right|_{p_i^2}, ...\right), \text{and } \bar{b} \times \bar{b}^{-1} = \bar{1}. \tag{14}$$

Since there exist quadratic non-residues taken by a non-prime p_i^2, the inverse vector in the sense of the second type multiplication exists not for every vector in V. Consequently, not for every vector of the space the division of the second type is defined.

References

1. Koliada, A.A.: The Modular Structures of the Conveyer Handling of Digital Information. Universitetskoie, Minsk (1992)
2. Munro, I.: The Computational Complexity of Algebraic and Numerical Problems. American Elseviewer, **7** (1980)
3. Inutin, S.A.: A Method of an Inverse Element Computation in a Finite Field. Scientific Works of the Surgut State University, **1** Regional Publishing House, Surgut (1997) 102-107
4. Inutin, S.A.: A Modular Calculation in the High Computer Diapason. The Electronics, **6** Moscow (2001) 12-17
5. Inutin, S.A.: Base Algorithms for Multiple Digits Calculations. Regional Publishing House, Surgut (2002)

Semi-systolic Architecture for AB^2 Operation over $GF(2^m)$

Hyun-Sung Kim, Il-Soo Jeon, and Jin-Ho Lee

Kyungil University, Computer Engineering,
712-701, Kyungsansi, Kyungpook Province, Korea
kim@kiu.ac.kr

Abstract. This paper proposes a new algorithm and two parallel semi-systolic array architectures to compute AB^2 over $GF(2^m)$. They are based on the standard basis representation and use the property of irreducible all one polynomial as a modulus. The first architecture, named PSA(Parallel-in parallel-out Semi-systolic array Architecture) has the critical path with $1D_{AND} + 1D_{XOR}$ per cell and the latency of $m+1$. The second architecture, named MPSA(Modified Parallel-in parallel-out Semi-systolic array Architecture) has the critical path with $1D_{XOR}$ per cell and has the same latency with PSA. They have lower latency and smaller hardware complexity than previous architectures. Since the proposed architectures have regularity, modularity and concurrency, they are suitable for VLSI implementation.

1 Introduction

Finite field or Galois fields play an important role in error-control coding, digital signal processing and cryptography [1]-[5]. Information processing in such areas usu-ally requires performing multiplication, power-sum(AB^2+C), inverse, division, and exponentiation. Among these operations, the power-sum is known as an efficient basic operation for public key cryptosystems [3] [15] over $GF(2^m)$. For example, division which using multiplication and multiplicative inverse(A/B $= AB^{-1}$) is computed efficiently by power-sum operations to design high-speed circuits. And inverse can be regarded as a special case of exponentiation because $B^{-1} = B^{2^m-2} = (B(B(B...B(B(B)^2)^2...)^2)^2)^2$. The equation can be computed as [15]:

Step1: $R = B$
Step2: for $i = m - 2$ downto 1
Step3: $R = BR^2$
Step4: $R = R^2$

Here, power-sum or AB^2 operation can be used to compute step 3 and 4 operations. But, computing these operations is much more complex than addition. It shows that designing high-speed circuits is required for them.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 998–1005, 2004.
© Springer-Verlag Berlin Heidelberg 2004

In recent years, three types of multipliers for $GF(2^m)$ have been proposed that are easily realized using VLSI techniques. These are normal, dual, and standard basis multipliers, which have their own distinct features. The normal basis architecture is very effective in performing inverse, squaring, and exponentiation operations. The dual basis architecture requires less chip area than the other two types. The standard basis architecture has lower design complexity, and its size is easier to extend to meet various applications due to its simplicity, regularity, and modularity in architecture. However, the former two types of finite field multipliers need basis conversion, while the latter type does not. From now on, we are focused on the standard basis multiplier.

Numerous architectures in $GF(2^m)$ have been proposed in [8]-[11]. In 1984, Yeh et al. [8] developed a parallel systolic architecture for performing the operation $AB + C$ in a general $GF(2^m)$. Jain et al. proposed semi-systolic array architectures for modular multiplication and squaring based on the standard basis representation [9]. Architectures to compute multiplication and inverse were represented with the normal basis [10]. Wei in [11] proposed a systolic power-sum architecture along with a modified basic cell that can perform eight different types of computations by adding one MUX and one DEMUX. Yet these circuits are still have some shortages for cryptographic applications due to their high circuit complexity and long latency. Thus, further research for efficient circuit for cryptographic applications is needed. To reduced the system complexity, Itoh and Tsujii designed two low-complexity multipliers for the class of $GF(2^m)$, based on the irreducible AOP (All One Polynomial) of degree m and the irreducible ESP (Equally Spaced Polynomial) of degree m [12]. Later, Hasan, et al. in [13] proposed an ESP-based multiplier using small-scale AOP-based multipliers as the processing unit. However, previous architectures have still some shortage.

This paper proposes a new algorithm and two parallel-in parallel-out semi-systolic array architectures to compute AB^2 over finite field $GF(2^m)$. They are based on the standard basis representation and use the property of irreducible AOP(All One Polynomial) as a modulus. Let D_{AND} and D_{XOR} be the latency of AND and XOR gate, respectively. The first architecture, named PSA(Parallel-in parallel-out Semi-systolic array Architecture) has the critical path with $1D_{AND} + 1D_{XOR}$ per cell and the latency of $m + 1$. The second architecture, named MPSA(Modified Parallel-in parallel-out Semi-systolic array Architecture) has the critical path with $1D_{XOR}$ per cell and has the same latency with PSA. The proposed two architectures have lower latency and smaller hardware complexity than previous architectures. Since the proposed architectures have regularity, modularity and concurrency, they are suitable for VLSI implementation. Moreover, they could be used to secure cryptosystem application.

2 Finite Fields and Operation

$GF(2^m)$ is finite extension field of finite field $GF(2)$ [6] [7] [15]. First of all, all the non-zero elements in finite field $GF(2^m)$ may be represented by three bases.

These are the standard, normal and dual basis. In the standard basis, an arbitrary element A over $GF(2^m)$ can be represented with $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$, i.e., $A = A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + ... + A_1\alpha + A_0$. For normal basis, an arbitrary element A over $GF(2^m)$ can be represented with $\{\alpha, \alpha^2, ..., \alpha^{2m-2}\}$, i.e., $A = A_{m-1}\alpha^{2m-2} + A_{m-2}\alpha^{2m-4} + ... + A_1\alpha + A_0$. The dual basis represents an arbitrary element A over $GF(2^m)$ with $\{u_0, u_1, ..., u_{m-1}\}$, i.e., $A = A_{m-1}u_{m-1} + A_{m-2}u_{m-2} + ... + A_1u_1 + A_0u_0$. The A_i's $(i = 0, 1, ..., m-1)$ are elements of $GF(2)$, respectively. From now on, we will only consider the standard basis representation.

A polynomial of the form $f(x) = f_m x^m + f_{m-1}x^{m-1} + ... + f_1 x + f_0$ is called an irreducible polynomial if and only if a divisor of $f(x)$ is 1 or $f(x)$. Also, a polynomial of the form $f(x) = f_m x^m + f_{m-1}x^{m-1} + ... + f_1 x + f_0$ is called a primitive polynomial if and only if a greatest common divisor of $f_i (0 \le i \le m)$ is '1'. Assume that a polynomial of the form $f(x) = f_m x^m + f_{m-1}x^{m-1} + ... + f_1 x + f_0$ over GF(2) is called an AOP (All One Polynomial) degree m if $f_i = 1$ for $i = 0, 1, ..., m$. It has been shown that an AOP is irreducible if and only if $m+1$ is prime and 2 is primitive modulo $m+1$. Let a set $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$ be generated by α which is a root of AOP f(x) and be standard basis. In standard basis, an elements A over $GF(2^m)$ are presented by $A = A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + ... + A_1\alpha^1 + A_0$. A set with $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}, \alpha^m\}$ is called extended basis of $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$. In extended basis, an elements a over $GF(2^m)$ are represented by $a = a_m\alpha^m + a_{m-1}\alpha^{m-1} + ... + a_1\alpha + a_0$ with $A_i = a_m + a_i (0 \le i \le m-1)$. Thus, an element over $GF(2^m)$ has two different representations. Let $F(\alpha) = x^m + x^{m-1} + x^{m-2} + ... + x^1 + 1$ be an irreducible AOP of degree m: and let α be a root of $F(x)$. i.e., $F(\alpha) = \alpha^m + \alpha^{m-1} + \alpha^{m-2} + ... + \alpha^1 + 1$. Then, we have $\alpha^m = \alpha^{m-1} + \alpha^{m-2} + ... + \alpha^1 + 1$,

$$\alpha^{m+1} = 1 \tag{1}$$

The multiplication operation of elements a and b^2 over $GF(2^4)$ in the extended basis can be performed by ab^2 mod p with $p = \alpha^{m+1} + 1$ which applied the property of AOP as a modulus. Let the result of this multiplication, ab^2 mod p, be $r = r_m\alpha^m + r_{m-1}\alpha^{m-1} + ... + r_1\alpha + r_0$. The recurrence equation for the proposed MSB first algorithm with the property of AOP is as follows: $r = ab^2 \bmod p = \{...[[ab_m]\alpha^2 \bmod p + ab_{m-1}]\alpha^2 \bmod p + ... + ab_1\}\alpha^2 \bmod p + ab_0$. From the recurrence equation, a new algorithm to compute ab^2 mod p can be derived as following Algorithm 1.

[Algorithm 1] MSB AB^2 algorithm
Input : $a = (a_m, a_{m-1}, ..., a_1, a_0), b = (b_m, b_{m-1}, ..., b_1, b_0)$
Output : $r = ab^2 \bmod p$
Initial value : $r^{m+1} = (r_m, r_{m-1}, ..., r_1, r_0) = (0, 0, ..., 0, 0)$
Step 1 for $i = m$ to 0
Step 2 $r^i = Circular_Left(r^{i+1}, 2) + ab_i$

where $Circular_Left(x, y)$ is the y-bits-left-circular shift of x and r_i is

used to represent i-th intermediate result for the final result r. In the above algorithm, the modular reduction is performed just by using 2-bits-left-circular-shift operation. Specially, all the operations in step 2 can be performed bit by bit in parallel.

Next section presents two parallel-in parallel-out semi-systolic array architectures based on Algorithm 1.

3 Semi-systolic Architecture

This section proposes two semi-systolic array architectures, PSA and MPSA, following the procedure in [9] and [14]. They are based on the property of all one polynomial as a modulus and are derived from the Algorithm 1.

3.1 Parallel Architecture (PSA)

Let $a, b,$ and b^2 be an elements in $GF(2^4)$. Then a, b and b^2 with an extended basis $\{1, \alpha, \alpha^2, \alpha^3, \alpha^4\}$ can be represented as follows:

$a = a_4\alpha^4 + a_3\alpha^3 + a_2\alpha^2 + a_1\alpha^1 + a_0$

$b = b_4\alpha^4 + b_3\alpha^3 + b_2\alpha^2 + b_1\alpha^1 + b_0$

$b^2 = b_4\alpha^8 + b_3\alpha^6 + b_2\alpha^4 + b_1\alpha^2 + b_0$

When $p = \alpha^5 + 1$ is used as a modular in extended basis, we have

$$\begin{aligned} r &= ab^2 \bmod p \\ &= a(b_4\alpha^4 + b_3\alpha^3 + b_2\alpha^2 + b_1\alpha + b_0)^2 \bmod p \\ &= \{...[[ab_4]\alpha^2 \bmod p + ab_3]\alpha^2 \bmod p + ... + ab_1\}\alpha^2 \bmod p + ab_0 \\ &= r_4\alpha^4 + r_3\alpha^3 + r_2\alpha^2 + r_1\alpha + r_0 \end{aligned}$$

Base on the above equation, the bit level operation is to be

$$\begin{aligned} t^4 &= a_1b_4\alpha^4 + a_0b_4\alpha^3 + a_4b_4\alpha^2 + a_3b_4\alpha + a_2b_4 \\ t^3 &= a_3b_3\alpha^4 + a_2b_3\alpha^3 + a_1b_3\alpha^2 + a_0b_3\alpha + a_4b_3 \\ t^2 &= a_0b_2\alpha^4 + a_4b_2\alpha^3 + a_3b_2\alpha^2 + a_2b_2\alpha + a_1b_2 \\ t^1 &= a_2b_1\alpha^4 + a_1b_1\alpha^3 + a_0b_1\alpha^2 + a_4b_1\alpha + a_3b_1 \\ t^0 &= a_4b_0\alpha^4 + a_3b_0\alpha^3 + a_2b_0\alpha^2 + a_1b_0\alpha + a_0b_0\,, \end{aligned}$$

where $t^i (0 \le i \le 4)$ represents intermediate values.

Fig.1 shows an architecture named PSA based on Algorithm 1 over $GF(2^4)$. PSA is composed of $(m+1)(m+1)$ basic cells. It is parallel architecture which a_i and $b_i (0 \le i \le m)$ are input at the same time. The m-bits of data a_i are input from the top of column cells and transmitted to the adjacent cells following each row. But the data in each column, which is b_i, is broadcasted to all cells in a same row at the same time step. Let D_{AND} and D_{XOR} be the latency of AND and XOR gate, respectively. PSA has a critical path with $1D_{AND} + 1D_{XOR}$ per cell. PSA in Fig. 1 can be generalized for every m as well as $m = 4$. Fig.2 shows the basic cell for PSA.

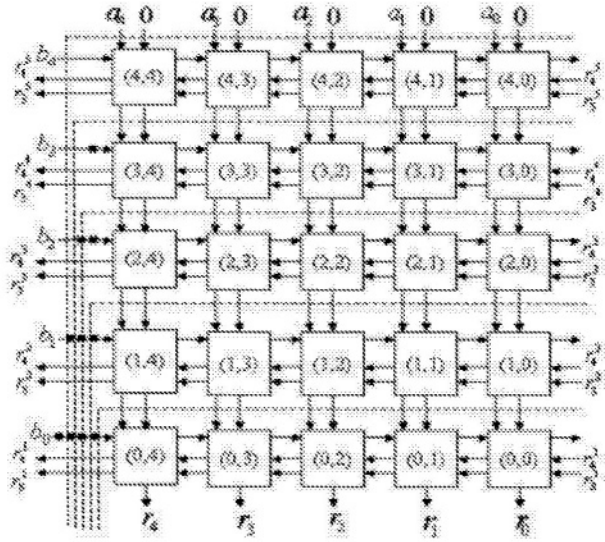


Fig. 1. PSA over $\mathrm{GF}(2^4)$

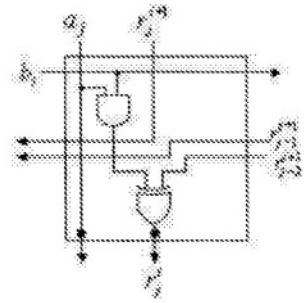


Fig. 2. Basic cell of PSA

3.2 Modified Architecture (MPSA)

PSA has the latency with $1D_{AND} + 1D_{XOR}$. If we postpone the modular reduction of each cell in PSA, we can get a better architecture with a low critical path. Fig. 3 shows an architecture for MPSA over $GF(2^4)$.

MPSA has a basic architecture with Fig. 4 (a) except the last row. The last row has the architecture of Fig. 4 (b). The Latency of MPSA is the same with PSA. But MPSA has more efficient latency than PSA. MPSA in Fig. 3 also can be generalized for every m as well as $m = 4$. Note that PSA and MPSA both are based on an extended basis, which have the result with $m+1$ bits, not with m bits. Accordingly, these get the result with an extended basis. Therefore, additional modular reduction is necessary after the process of PSA and MPSA.

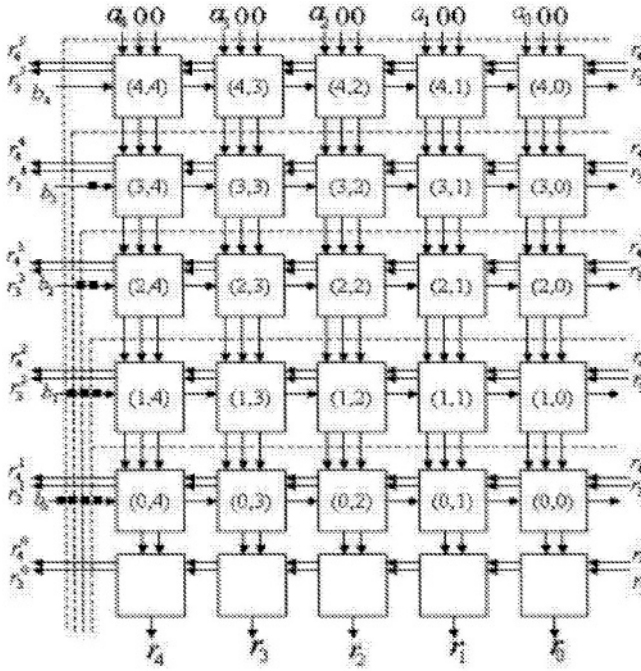


Fig. 3. Architecture for MPSA over $\mathrm{GF}(2^4)$

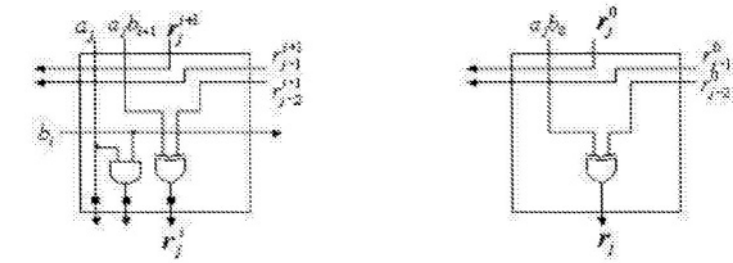


Fig. 4. Basic cells for MPSA

Fig. 5 shows the architecture for the final modular reduction, which is for the result to be an ordinary field element from an extended field element.

4 Comparison and Analysis

Table 1 shows the comparison between proposed architectures and pervious architectures. It is assumed that AND and XOR represent 2-input AND and XOR

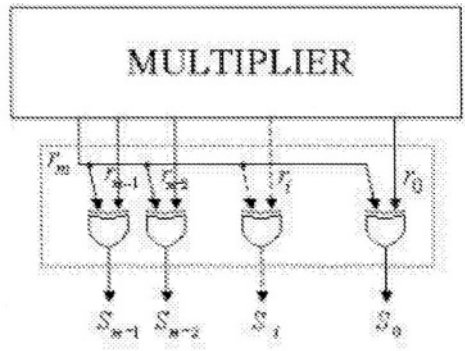


Fig. 5. Architecture for the final modular reduction

gates, respectively, and latch for 1-bit latch. Let D_{AND} and D_{XOR} be the latency of AND and XOR gate, respectively. The architecture in [9] has a latency of $m(m-1)+\lfloor m/2 \rfloor + 1$ and the critical path of $1D_{AND} + 2D_{XOR}$. In [11], Wei proposed the architecture with a latency of $3m$ and a critical path of $1D_{AND} + 2D_{XOR}$. PSA has a latency of $m+1$ and a critical path of $1D_{AND} + 1D_{XOR}$ but MPSA has critical path with $1D_{XOR}$. As a result, the proposed two architectures, PSA and MPSA, have lower latency and smaller complexity than previous architectures in [9] and [11].

Table 1. Comparisons

Architecture / Item	Jain [9]	Wei [11]	PSA	MPSA
Function	AB	AB^2+C	AB^2	AB^2
Number of cell	m^2	m^2	$(m+1)^2$	$(m+1)(m+2)$
Cell complexity	2 AND 2 XOR 3 latches	3 AND 3 XOR 10 latches	1 AND 1 XOR 2 latches	1 AND 1 XOR 3 latches
Latency	$m(m-1)+\lfloor m/2 \rfloor+1$	$3m$	$m+1$	$m+1$
Critical path	1-D_{AND} +2-D_{XOR}	1-D_{AND} +2-D_{XOR}	1-D_{AND} +1-D_{XOR}	1-D_{XOR}

5 Conclusions

This paper proposed a new algorithm and two parallel-in parallel-out semi-systolic array architectures to compute AB^2 over finite field $GF(2^m)$. The property of irreducible AOP was used as an irreducible polynomial to get a better

hardware and time complexity. Proposed two architectures have lower latency and smaller hardware complexity than previous architectures as shown in Table 1. Since PSA and MPSA have regularity, modularity and concurrency, they are suitable for VLSI implementation. Moreover, they could be used to secure cryptosystem application.

References

1. W. W. Peterson, E. J. Weldon, Error-Correcting Codes, Cambridge, MA: MIT Press, 1972.
2. I. S. Reed, T. K. Truong, The use of finite fields to compute convolutions, IEEE Trans.Inform. Theory, vol. IT-21, pp.208-213, Mar. 1975.
3. D. E. R. Denning, Cryptography and data security Reading, MA: Addison-Wesley, 1983.
4. A. M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, Adv. Cryptol., Proc. Eurocrypt '84, pp.224-314, Apr. 1984.
5. W. Diffe, M. Hellman, New Directions in Cryptography, IEEE Trans. Information Theory, vol. 22, pp. 644-654, 1976.
6. E. R. Berlekamp, Algebraic Coding Theory, New York: McGraw-Hill, 1986.
7. B. Benjauthrit,, I. S. Reed, Galois switching function and their applications, IEEE Trans. Comp., vol. C-25, pp. 78-86, Jan. 1976.
8. C. S. Yeh, S. Reed,T. K. Truong, Systolic multipliers for finite fields $GF(2^m)$, IEEE Trans. on Computers, vol. C-33, pp.357-360, Apr. 1984.
9. S. K. Jain, L. Song, Efficient Semisystolic Architectures for finite field Arithmetic, IEEE Trans, on VLSI Systems, vol. 6, no. 1, pp. 101-113, Mar. 1998.
10. J. L. Massey, J. K. Omura, Computational method and apparatus for finite field arithmetic, U. S. Patent application, submitted 1981.
11. S. W. Wei, A systolic power-sum circuit for $GF(2^m)$, IEEE Trans. on Computers, vol. 43, pp. 226-229, Feb. 1994.
12. T. Itoh, S. Tsujii, Structure of parallel multipliers for a class of finite fields $GF(2^m)$, Info. Comp., vol. 83, pp. 21-40, 1989.
13. M. A. Hasan, M. Z. Wang, V. K. Bhargava, Modular Construction of low complexity parallel multipliers for a class of finite fields $GF(2^m)$, IEEE Trans. on Computers, vol.8. pp. 962-971, Aug. 1992.
14. S. Y. Kung, VLSI Array Processors, Prentice Hall, 1988.
15. H. S. Kim, Serial AOP Arithmetic Architecture for Modular Exponentiation, Ph. D. Thesis, Kyungpook National Univ., 2002.

A Class of Block Smoothers for Multigrid Solution of Saddle Point Problems with Application to Fluid Flow*

Piotr Krzyżanowski

Institute of Applied Mathematics,
Warsaw University,
Banacha 2, 02-097 Warszawa, Poland,
przykry@mimuw.edu.pl

Abstract. We design and analyse an iterative method, which uses a specific block smoother for the multigrid cycle. Among many possibilities we choose a few multigrid iterations as the smoother's blocks. The result is a multilevel procedure that works for regular saddle point problems and features all good properties of the classical multigrid for elliptic problems, such as the optimal complexity and convergence rate independent of the number of levels.

1 Introduction

In many applications, one needs to solve an ill-conditioned, large discrete saddle point problem with a block matrix

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix}, \tag{1}$$

which is non-symmetric, indefinite and ill-conditioned. For example, after a linearization of the Navier–Stokes system one ends up with a huge linear system with such a nonsymmetric block matrix, which ill-conditioned with respect to the mesh size h. We propose and analyse a new multilevel method for solving the linear system, based on inner and outer multigrid iteration.

Multigrid schemes for saddle point problems have been considered by many authors before, see for example [2], [14], [13], [10], [15]. These methods have usually been designed with a very specific equation in mind. Some of these works, e.g. [2], stressed the necessity of using sufficiently strong smoothers in order to achieve satisfactory performance of the multigrid. On the other hand, the block nature of (1) promotes the development of preconditioners exploiting this structure of the problem. Block preconditioning has also attained a lot of attention from many authors, see e.g. [6], [3], [11], [7], [12].

* This work has partially been supported by State Committee for Scientific Research (KBN) research grant 2 P03A 005 24

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1006–1013, 2004.
© Springer-Verlag Berlin Heidelberg 2004

We propose a method that combines these two approaches: the multigrid and the block preconditioning. Similar combination has been used in e.g. [2], however our approach looks a bit more flexible and makes the choice of concrete preconditioner broader. We design and analyse an iterative method, which may choose from a variety of blocked approximate solvers. An interesting option is to use an inner multigrid cycle as a smoother inside the outer multigrid cycle, which results in a multilevel procedure that resembles to some extent the W-cycle method but has better properties. We treat each variable separately, following the block approach used for preconditioning. Our method works for saddle point problems such as the Stokes problem, and it features all good properties of the classical multigrid for elliptic problems, such as the optimal complexity and convergence in "natural" norms.

In this paper, we describe the blocked smoother and derive from it a specific inner-outer multigrid iteration which uses two inner multigrid iterations to apply the smoother. We discuss the performance of these methods, including convergence theorems and numerical results. The details of the theoretical analysis, which is based on a combination of the multigrid and block preconditioners theory, will be given elsewhere.

2 Blocked Multigrid Framework for Saddle Point Problems

Let $\bar{V}, \bar{W}$ be real Hilbert spaces with scalar products denoted by $((\cdot, \cdot))$ and $(\cdot, \cdot)$, respectively. The corresponding induced norms are $\|\cdot\|$ and $|\cdot|$. Let us consider two continuous bilinear forms, $a(\cdot, \cdot) : \bar{V} \times \bar{V} \to R$ and $b(\cdot, \cdot) : \bar{V} \times \bar{W} \to R$ and assume that $a(\cdot, \cdot)$ satisfies

$$\exists \alpha > 0 \quad a(u, u) \geq \alpha \|u\|^2 \quad \forall u \in V^0 = \{v \in \bar{V} : b(v, q) = 0 \quad \forall q \in \bar{W}\}, \tag{2}$$

and there holds the *inf-sup* condition:

$$\exists \beta > 0 \quad \sup_{v \in \bar{V}, v \neq 0} \frac{b(v, q)}{\|v\|} \geq \beta |q| \quad \forall q \in \bar{W}. \tag{3}$$

We consider a family of nested finite element spaces $V_0 \times W_0 \subset V_1 \times W_1 \subset \ldots \subset V_K \times W_K \subset \bar{V} \times \bar{W}$, where every $V_{k+1} \times W_{k+1}$ is obtained from $V_k \times W_k$ through mesh refinement procedure, $h_{k+1} = \frac{1}{2} h_k$. These spaces inherit their norms from $\bar{V} \times \bar{W}$, but in practice, one additionally uses another (mesh-dependent) inner products and norms, denoted by $((\cdot, \cdot))_k$ and $\|\cdot\|_k$ in V_k and analogously in W_k. Later on, we shall use these auxiliary inner products to define certain linear operators in V_k and W_k. We shall also denote for short $X_k = V_k \times W_k$; for $x = (u, p)^T \in X_k$ and $y = (v, q)^T \in X_k$, we define the natural inner product in X_k by $\langle x, y \rangle = ((u, v)) + (p, q)$, and the discrete one $\langle (u, p)^T, (v, q)^T \rangle_k = ((u, v))_k + (p, q)_k$, with corresponding norms denoted by $|||\cdot|||$ and $|||\cdot|||_k$.

Additionally, we assume that a uniform discrete *inf-sup* condition holds for all levels k:

$$\exists \beta > 0 \quad \forall k = 1, \ldots, K \qquad \sup_{v \in V_k, v \neq 0} \frac{b(v, q)}{\|v\|} \geq \beta |q| \quad \forall q \in W_k. \tag{4}$$

In what follows, for nonnegative scalars x, y, we shall write $x \lesssim y$ if there exits a positive constant C, independent of x, y and the level k, such that $x \leq Cy$. Similarly, $y \gtrsim x$ is equivalent to $x \lesssim y$. Finally, $x \simeq y$ means $x \lesssim y$ and $y \lesssim x$ simultaneously.

On the kth level, we consider the following saddle point problem:

Problem 1. Find $(u_k, p_k) \in V_k \times W_k$ such that

$$\mathcal{M}_k \begin{pmatrix} u_k \\ p_k \end{pmatrix} \equiv \begin{pmatrix} A_k & B_k^* \\ B_k & 0 \end{pmatrix} \begin{pmatrix} u_k \\ p_k \end{pmatrix} = \begin{pmatrix} F_k \\ G_k \end{pmatrix}. \tag{5}$$

The finite dimensional space operators in (5) are discretizations on the kth level mesh of the corresponding differential operators, that is,

$$\begin{aligned} &A_k : V_k \to V_k, \quad ((A_k u, v))_k = a(u, v) \quad \forall u, v \in V_k, \\ &B_k : V_k \to W_k, \quad (B_k u, p)_k = b(u, p) \quad \forall u \in V_k, p \in W_k, \end{aligned}$$

B_k^* denotes the formal adjoint operator to B_k, i.e. $(B_k u, p)_k = ((u, B_k^* p))_k$ for all $u \in V_k$, $p \in W_k$.

We introduce four more operators. $L_k : V_k \to V_k$ and $M_k : W_k \to W_k$ define the correspondence between the original and auxiliary inner products in V_k and W_k,

$$\begin{aligned} ((L_k u, v))_k &= ((u, v)) \qquad \forall u, v \in V_k, \\ (M_k p, q)_k &= (p, q) \qquad \forall p, q \in W_k. \end{aligned} \tag{6}$$

Usually, systems with L_k and M_k are not easy to solve. Therefore, we will need two more operators, spectrally equivalent to L_k and M_k: $L_{0k} : V_k \to V_k$ and $M_{0k} : W_k \to W_k$. We assume that they are self-adjoint, their inverses are easier to apply than those of L_k and M_k, and that

$$((L_{0k} u, u))_k \simeq ((L_k u, u))_k \qquad \forall u \in V_k, \tag{7}$$

$$(M_{0k} p, p)_k \simeq (M_k p, p)_k \qquad \forall p \in W_k. \tag{8}$$

In other words, we shall always assume that L_{0k} and M_{0k} define good preconditioners for L_k and M_k. Later, it will be important to choose these preconditioners as multigrid cycles.

3 Examples

We consider two problems in the CFD that lead to saddle point problem formulation as in Problem 1. Let Ω be a bounded, open polygon in R^2.

Example 1. Linearized Navier-Stokes equation

A reasonable model for a linearization of the Navier-Stokes equations is the Oseen equation,

$$\begin{cases} -\nu\Delta u + (\omega \cdot \nabla)u + \nabla p = f & \text{in } \Omega, \\ \operatorname{div} u = 0 & \text{in } \Omega. \end{cases}$$

This problem may be expressed as a saddle point problem for $(u, p) \in H_0^1(\Omega) \times L_0^2(\Omega)$, [8]. Choosing inf-sup stable finite element functions, it follows that L_k is the discrete Laplacian matrix, M_k is the discrete mass matrix. A_k corresponds to a finite element approximation of a convection-diffusion operator, while B_k approximates the divergence operator. Note that A_k is nonsymmetric and the condition number of the saddle point problem grows proportionally to h^{-2}, making the finite element Oseen equations ill-conditioned.

Example 2. Biharmonic equation

The Ciarlet-Raviart method for a first Dirichlet biharmonic problem [5] reads:

$$\begin{aligned} (\sigma, v)_{L^2(\Omega)} - (\nabla v, \nabla u)_{L^2(\Omega)} &= 0 \qquad \forall v \in H^1(\Omega), \\ -(\nabla\sigma, \nabla w)_{L^2(\Omega)} &= -(f, v)_{L^2(\Omega)} \qquad \forall w \in H_0^1(\Omega). \end{aligned} \tag{9}$$

Then, in our notation, A_k corresponds to the usual mass matrix, while L_k is a matrix corresponding to Helmholtz operator $-\Delta + I$ discretization, and M_k is the Laplacian $-\Delta$ (with boundary constraints) representation. Note that A_k is uniformly elliptic only on $\ker B_k$, while its global ellipticity constant decays proportionally to h. The condition of the saddle point problem matrix is proportional to h^{-4}.

4 Block Smoothed Multigrid Method for Problem 1

In order to solve the k-th level problem,

$$\mathcal{M}_k x_k = g_k$$

where $x_k, g_k \in X_k = V_k \times W_k$, we use classical W-cycle multigrid scheme $\mathcal{MG}_k(x_k^0, g_k)$ with m pre- and post-smoothing iterations using smoother S_k, see e.g. [9] or [1] for details. Here, x_k^0 denotes the initial approximation to x_k. We recall the MG scheme briefly mainly for the notational purposes.

On the zeroth level, $k = 0$, we define $\mathcal{MG}_0(x_0^0, g_0) = \mathcal{M}_0^{-1} g_0$ (direct solve). For $k > 0$ we define $\mathcal{MG}_k$ recursively. First we apply m smoother iterations in the pre-smoothing step

for $i = 1, \ldots, m$
$x_k^j = x_k^{j-1} - S_k(\mathcal{M}_k x_k^{j-1} - g_k)$,

and then follow with the coarse grid correction: for $g_{k-1} \in X_{k-1}$ defined by the identity

$$\langle g_{k-1}, y\rangle k-1 = \langle g_k - \mathcal{M}_k x_k^{j-1}, y\rangle_k \quad \forall y \in X_{k-1},$$

and we compute $\tilde{x}_{k-1}$ by applying two iterations of $(k-1)$ level method (with zero initial guess) to problem

$$\mathcal{M}_{k-1}\tilde{x}_{k-1} = g_{k-1},$$

so that $\tilde{x}_{k-1} = \mathcal{MG}_{k-1}(\mathcal{MG}_{k-1}(0, g_{k-1}), g_{k-1})$. Finally, we set

$$\mathcal{MG}_k(x_k^0, g_k) = x_k^m + \tilde{x}_{k-1}.$$

The key ingredient of the above procedure is of course the smoother. It should be easy to apply to a vector, and it should remove effectively high frequency components of the error. The simplest choice used in practice is the Richardson iteration; however, it turns out that sometimes more efficient smoothers are necessary, [2]. Therefore, in what follows we shall consider smoothers based on block preconditioned Richardson iteration.

We focus here on the W-cycle iteration, note however, that it is also possible to use other variants of the multigrid (see e.g. [1]), making use of such concepts as the V-cycle, the post-smoothing, or using smoothers other than the Richardson method.

We shall consider a block preconditioned Richardson smoother, that is,

$$S_k = \frac{1}{\omega_k}\mathcal{M}_{0k}^{-1}\mathcal{M}_k^*\mathcal{M}_{0k}^{-1}, \tag{10}$$

where $\omega_k > 0$ is a prescribed parameter and

$$\mathcal{M}_{0k} = \begin{pmatrix} L_{0k} & \\ & M_{0k} \end{pmatrix}. \tag{11}$$

Theorem 1. *Let x_k be the accurate solution of Problem 1 and let $\tilde{x}_k = \mathcal{MG}(x_k^0, g_k)$ be its approximation after one iteration of the kth level W-cycle method with m inner smoother iterations defined by* (10) *and with initial guess x_k^0. Then, for any $0 < \delta < 1$, there exists m large enough such that the multigrid iteration is convergent linearly with rate δ. The convergence rate is independent of k.*

5 Smoother Based on Inner Multigrid

In order to solve Problem 1 on the kth level in $\mathcal{O}(N_k)$ floating point operations, where $N_k = \dim X_k$, we use the multigrid procedure $\mathcal{MG}_k$ described above, but with specific choice of the smoothing preconditioners L_{0k}, M_{0k}.

For $F_k \in V_k$ we define

$$L_{0k}^{-1} F_k = U_k, \tag{12}$$

where U_k is a result of one classical V-cycle multigrid applied to solve $L_k U_k = F_k$. Similarly, for $G_k \in W_k$,

$$M_{0k}^{-1} G_k = P_k, \tag{13}$$

where P_k is derived from one V-cycle multigrid for $M_k P_k = G_k$. In other words, the smoother S_k defined by (10) amounts to applying two kth level multigrid cycles to each variable separately, interlaced with multiplication by the transpose of $\mathcal{M}$.

The resulting procedure uses an inner multigrid cycle in an outer multigrid iteration. It also applies the outer multigrid to a squared preconditioned system, so that in one outer iteration, two inner multigrid cycles are performed, see Figure 1.

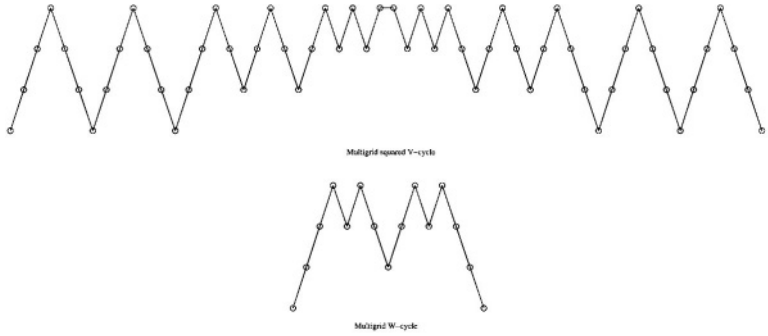


Fig. 1. The new scheme using a V-cycle inner and outer iteration (top), versus the usual W-cycle multigrid.

Theorem 2. *Under the above additional assumptions, and for sufficiently large number m of smoother iterations, the kth level MG iteration, consisting of the W-cycle multigrid for Example 2 with a smoother defined by* (10) *and with block solvers as above, is convergent. The convergence rate is independent of the level k and the arithmetic complexity of one iteration is $\mathcal{O}(N_k)$.*

6 Numerical Experiments

Let us consider a saddle point problem which is an *ad hoc* modification of the Ciarlet-Raviart saddle point formulation of the first biharmonic equation. Since in its original form, the A matrix (the mass matrix in this case) is not uniformly

H^1-elliptic with respect to the mesh size h [4], we replace this matrix with a matrix that corresponds to the discretization of the H^1 inner product. It is clear that after such a modification the uniform *inf-sup* condition remains to hold.

We report on the convergence factors of our block smoothed multigrid method in the following configuration: the outer iteration uses a 2-level multigrid V-cycle, with $m = 1, \ldots, 4$ pre- and postsmoothings. The inner (that is, the smoothing iteration) is either a direct solve or again a two-grid V-cycle, with k smoothings. The convergence factor is calculated as the mean value of $||r_{i+1}||/||r_i||$ in three consecutive iterations. The $||\cdot||$ norm is the usual Euclidean norm.

Table 1. Left: A block smoothed multigrid with a directly solved preconditioner; Right: A block smoothed multigrid with inner multigrid which is a 2-grid V-cycle.

	m			
N	1	2	3	4
9	0.76	0.58	0.44	0.34
17	0.76	0.58	0.44	0.33
33	0.76	0.58	0.44	0.34

	m			
N	1	2	3	4
9	0.91	0.85	0.79	0.73
17	0.89	0.79	0.71	0.65
33	0.87	0.77	0.68	0.60

An interesting observation, see Table 2, is that the method still works very well for the *original* Ciarlet-Raviart method with a compatible right hand side, despite the global ellipticity constant is proportional to h. Here we report on a two grid outer iteration, F = [0,rand(f)], with exactly solved block preconditioner. This suggests that it is only the V^0–ellipticity which controls the behaviour of the method under consideration.

Table 2. A block smoothed inner-outer V-cycle multigrid for the original Ciarlet-Raviart problem with random right hand side f. Exactly solved block preconditioner. Instead of the average, we report on the convergence factor on the 4-th iteration (as we obtained extremely good convergence factors in 3 previous iterations).

	m			
N	1	2	3	4
9	0.47	0.29	0.36	0.34
17	0.46	0.32	0.35	0.29
33	0.45	0.36	0.34	0.26

7 Conclusions

The new flexible multilevel scheme for saddle point problems makes efficient use of a block smoother. The method has optimal complexity $\mathcal{O}(N_k)$, where N_k is the kth level problem size, and the smoother error reduction is proportional to $\frac{1}{m}$.

The method can be applied to several saddle point problems encountered in the CFD, including Stokes, Oseen equations or the Ciarlet-Raviart method for the biharmonic problem and reuses simple multigrid schemes for *elliptic* problems in the saddle point problem context.

A potential drawback of the proposed scheme is its sensitivity to the ellipticity constant in the nonsymmetric case.

References

1. D. BRAESS, *Finite elements,* Cambridge University Press, Cambridge, 1997. Theory, fast solvers, and applications in solid mechanics, Translated from the 1992 German original by Larry L. Schumaker.
2. D. BRAESS AND R. SARAZIN, *An efficient smoother for the Stokes problem,* Appl. Numer. Math., 23 (1997), pp. 3–19. Multilevel methods (Oberwolfach, 1995).
3. J. BRAMBLE AND J. PASCIAK, *A preconditioning technique for indefinite problems resulting from mixed approximation of elliptic problems,* Math. Comp., 50 (1988), pp. 1–17.
4. F. BREZZI AND M. FORTIN, *Mixed and Hybrid Finite Element Methods,* Springer-Verlag, New York, Berlin, Heidelberg, London, Paris, Tokyo, Hong Kong, Barcelona, 1991.
5. P. G. CIARLET AND P.-A. RAVIART, *A mixed finite element method for the biharmonic equation,* in Mathematical aspects of finite elements in partial differential equations (Proc. Sympos., Math. Res. Center, Univ. Wisconsin, Madison, Wis., 1974), Math. Res. Center, Univ. of Wisconsin-Madison, Academic Press, New York, 1974, pp. 125–145. Publication No. 33.
6. E. D'YAKONOV, *On iterative methods with saddle operators,* Soviet Math. Dokl., 35 (1987), pp. 166–170.
7. H. C. ELMAN, D. J. SILVESTER, AND A. J. WATHEN, *Iterative methods for problems in computational fluid dynamics,* in Iterative methods in scientific computing (Hong Kong, 1995), Springer, Singapore, 1997, pp. 271–327. (also available as a University of Maryland report).
8. V. GIRAULT AND P. RAVIART, *Finite Element Method for Navier–Stokes Equations. Theory and Algorithms,* Springer-Verlag, Berlin, Heidelberg, New York, 1986.
9. W. HACKBUSCH, *Multigrid methods and applications,* Springer-Verlag, Berlin, 1985.
10. M. R. HANISCH, *Multigrid preconditioning for the biharmonic Dirichlet problem,* SIAM J. Numer. Anal., 30 (1993), pp. 184–214.
11. A. KLAWONN, *Preconditioners for Indefinite Problems,* PhD thesis, Universität Münster, Germany, 1996.
12. P. KRZYŻANOWSKI, *On block preconditioners for nonsymmetric saddle point problems,* SIAM J. Sci. Comp., 23 (2001), pp. 157–169.
13. P. PEISKER, *A multilevel algorithm for the biharmonic problem,* Numer. Math., 46 (1985), pp. 623–634.
14. R. VERFÜRTH, *A multilevel algorithm for mixed problems,* SIAM J. Numer. Anal., 21 (1984), pp. 264–271.
15. G. WITTUM, *Multi-grid methods for Stokes and Navier-Stokes equations. Transforming smoothers: algorithms and numerical results,* Numer. Math., 54 (1989), pp. 543–563.

Parallelizable Password-Authenticated Key Exchange Protocol

Sung-Woon Lee and Kee-Young Yoo

Kyungpook National University, Computer Engineering,
Daegu, 702-701, Rep. of Korea
staroun@hanafos.com, yook@knu.ac.kr

Abstract. Boyko et al. presented two protocols called PAK and PAK-X which are the password-authenticated key exchange protocol based on the Diffie-Hellman algorithm to provide a formal proof of security in the random oracle model. These protocols have very good property and were submitted to IEEE P1363.2 for public key cryptography: password-based techniques. However, they are not the most efficient methods in the perspective of the total execution time because they do not provide parallelizability to each party. Therefore, the purpose of this paper is to speed up these schemes by supporting parallel computability to both parties. This paper proposes two protocols, PPAK and PPAK-X, which are parallelizable password-authenticated key exchange protocols while maintaining provable security in PAK and PAK-X by simply modifying their structures.

1 Introduction

It is necessary for verifying the identities of the communicating parties when they initiate a connection. This authentication is usually provided in combination with a key exchange protocol between the parties. Techniques for user authentication are broadly based on one or more of the following categories: (1) what a user knows, (2) what a user is, or (3) what a user has. Among them, the first category is the most widely used method due to the advantages of simplicity, convenience, adaptability, mobility, and less hardware requirement. It requires users only to remember their knowledge such as a password. However, traditional password-based protocols are susceptible to off-line password guessing attacks (called dictionary attacks) since many users tend to choose memorable passwords of relatively low entropy. Since Lomas et al. [1] presented the first protocol which was resistant to these types of off-line dictionary attacks, many password authenticated key exchange protocols have been proposed [2-11]. Of these, only [3, 4, 11] were formally proven secure in the random oracle model [12], with [4] also requiring ideal ciphers.

The following classes of password-authenticated key exchange protocols are defined by IEEE Std1363a-2002 [13].

• Balanced Password-authenticated Key Exchange Schemes [3, 4, 5, 7, 11], in which two parties use a shared password to negotiate one or more shared

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1014–1019, 2004.
© Springer-Verlag Berlin Heidelberg 2004

ephemeral keys such that the shared keys are established if and only if they use the same password. The shared keys may then be used for password-based entity authentication or symmetric cryptography.

• Augmented Password-authenticated Key Exchange Schemes (usually called verifier-based protocol) [2, 3, 6, 7, 8, 9, 10, 11], in which two parties (denoted Client and Server) use related password-based values to negotiate one or more shared ephemeral keys. Server uses password verification data (usually called verifier) that is derived from client's password data. The scheme forces an attacker who steals the password verification data to further perform a successful brute-force attack in order to masquerade as client.

Boyko et al. in [3] presented a protocol called PAK which is the password-authenticated key exchange protocol based on a password pre-shared between two parties that provides mutual explicit authentication. They then extended PAK to a protocol called PAK-X that an attacker cannot be able to pose as a client after compromising the server, in which the client stores a plaintext version of the password, while the server only stores a verifier for the password. Finally, they formally proved security of their protocols in the random oracle model to show strength against both passive and active adversaries. These protocols have very good property and were submitted to IEEE P1363.2 for public key cryptography: password-based techniques [13]. However, considering the total execution time, they are not the most efficient methods due to not providing parties with parallel computation.

In this paper, we present two protocols called PPAK and PPAK-X which are parallelizable password-authenticated key exchange protocols for improving the efficiency of PAK and PAK-X while maintaining provable security by simply modifying their structures.

2 The PAK and PAK-X Protocols

We now describe the PAK and PAK-X protocols presented by Boyko et al. in [3].

Let κ and l denote security parameters, where κ is the main security parameter and can be thought of as a general security parameter for hash functions and secret keys (say 128 or 160 bits), and $l > \kappa$ can be thought of as a security parameter for discrete-log-based public keys (say 1024 or 2048 bits). Let {0,1}* denote the set of finite binary strings and $\{0,1\}^n$ the set of binary strings of length n. Two communication parties, called *Alice* and *Bob,* share a common weak secret (password) in a secure way. A and B denote the identities of *Alice* and *Bob,* respectively. Let q and p be primes such that $p = rq + 1$ for some value r co-prime to q. Let g be a generator of a subgroup of Z_p^* of size q. Define hash functions (called random oracles [12]) $H_{2a}, H_{2b}, H_3 : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ and $H_1 : \{0,1\}^* \rightarrow \{0,1\}^\eta$ (where $\eta >= l + \kappa$). Assume that H_1, H_{2a}, H_{2b}, and H_3 are independent random functions. And H_1 operates on its output by 'modulo

p'. We will omit 'mod p' from expressions for simplicity. The PAK protocol is given in Figure 1.

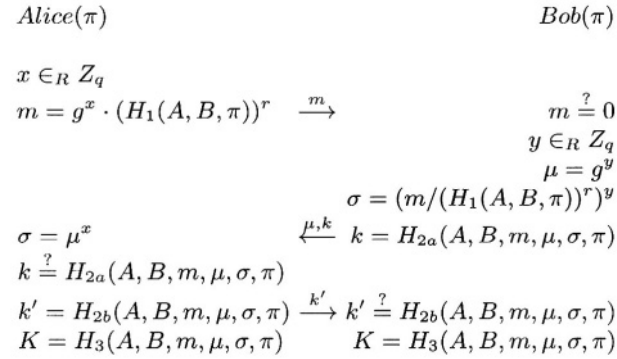


Fig. 1. PAK protocol.

In addition to the random oracles to be used in PAK, PAK-X uses additional random functions $H_0, H_0' : \{0,1\}^* \rightarrow \{0,1\}^{|q|+\kappa}$. To be resistant to server compromise, the sever must not store the plaintext password. In stead, the server stores a verifier to verify a user's password. The verifier is computed by $V(A, B, \pi) = g^{v[A,B]}$, where $v[A, B] = H_0(min(A, B), max(A, B), \pi)$. The PAK-X protocol is given in Figure 2.

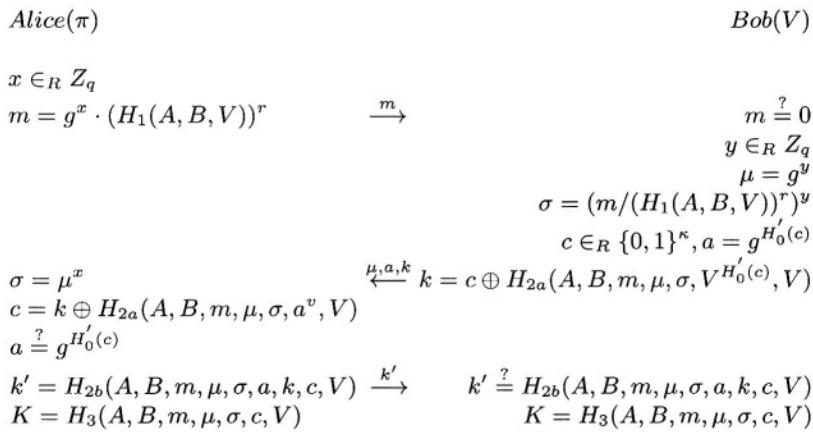


Fig. 2. PAK-X protocol.

PAK and PAK-X require both *Alice* and *Bob* to compute a shared Diffie-Hellman key. This might take a long time due to time-consuming operations such as exponentiations, i.e., perhaps some seconds on slow device. However, they cannot but be executed in serial order between two parties due to their construction nature. They can speed up the total execution time for a key exchange and authentication if *Alice* and *Bob* can compute time consuming operations at the same time.

3 The PPAK and PPAK-X Protocol

We present parallel computable PPAK and PPAK-X that can greatly reduce the total execution time for PAK and PAK-X. The idea to make them parallelizable protocols is simple. PPAK and PPAK-X take up massage exchange of 4 steps rather than 3 steps in terms of communication steps to quickly compute the message having influence on execution of the opposite party and send it to him. Note that the definitions of all parameters are identical to those of PAK and PAK-X. The PPAK protocol is given in Figure 3.

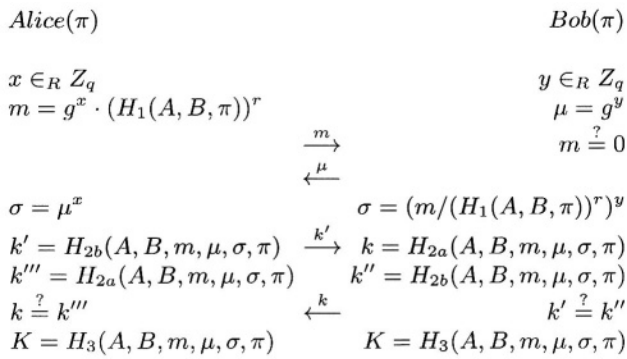


Fig. 3. PPAK protocol.

The PPAK-X protocol is given in Figure 4.

As shown in Figure 3 and 4, we preserve all operations of PAK and PAK-X without being modified, merely need four steps in PAK and PAK-X instead of 3 steps for message exchange, and then properly rearrange the operations to minimize the idle time of each party. That means that our protocols don't damage the security formally proved secure in PAK and PAK-X at all.

4 Efficiency and Comparison

Performance of key exchange protocols can be approximated in terms of communication and computation loads. The number of steps is a factor to the communication load, while the numbers of exponentiations, random numbers, and hash

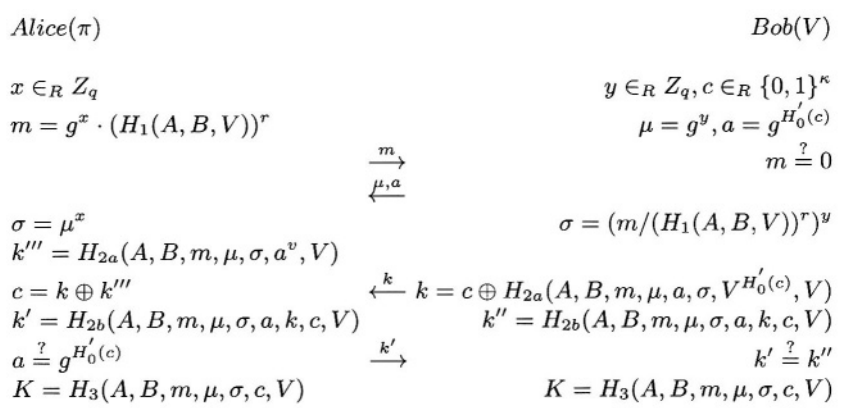


Fig. 4. PPAK-X protocol.

functions are factors to the computation load. Table 1 compares PAK related protocols regarding with several efficiency factors such as the number of protocol steps, random numbers, exponentiations, and hash functions, respectively.

Table 1. Comparison of protocols.

		PAK	PPAK	PAK-X	PPAK-X
# of steps		3	4	3	4
# of random numbers		2	2	3	3
# of exponentiations	Alice	3	3	5	5
	Bob	3	3	5	5
	Parallel	5	3	9	6
# of hash functions	Alice	4	4	5	5
	Bob	4	4	5	5
	Parallel	6	4	9	6

For the measure of a total execution time, let us only consider modular exponentiations, which are most time consuming operations in protocol. E(*Alice : Bob*) means parallel execution for modular exponentiation between both parties. That is, one party is able to compute something while he or she is waiting for the other party's reply. PAK has 4E, i.e., $\mathrm{E}((H_1(A,B,\pi))^r, (H_1(A,B,\pi))^r)$, $\mathrm{E}(g^x, g^y)$, $\mathrm{E}(-, (m/(H_1(A,B,\pi))^r)^y)$, and $\mathrm{E}(\mu^x, -)$, while PPAK has only 3E, i.e., $\mathrm{E}((H_1(A,B,\pi))^r, (H_1(A,B,\pi))^r)$, $\mathrm{E}(g^x, g^y)$, and $\mathrm{E}(\mu^x, (m/(H_1(A,B,\pi))^r)^y)$. Here '-' means no exponentiations. Similarly, PPAK-X has only 6E, while PAK-X has 8E. Note that, among them, 2E in PAK and PPAK and 3E in PAK-X and PPAK-X can be pre-computed. As we can see in table 1, PPAK and PPAK-X greatly reduce the total execution time of PAK and PAK-X due to parallel execution.

Acknowledgement. This work was supported by the Brain Korea 21 Project in 2003.

References

1. M. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham, Reducing risks from poorly chosen keys, ACM Operating Systems Review, vol. 23, no. 5, pp. 14-18, December 1989.
2. S. Bellovin and M. Merritt, Encrypted key exchange: Password-based protocols secure against dictionary attacks, In IEEE Symposium on Research in Security and Privacy, pp. 72-84, 1992.
3. V. Boyko, P. MacKenzie, and S. Patel, Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman, In EUROCRYPT 2000, pp. 156-171, 2000.
4. M. Bellare, D. Pointcheval, and P. Rogaway, Authenticated Key Exchange Secure Against Dictionary Attacks, In EUROCRYPT 2000, pp. 139-155, 2000.
5. T. Kwon and J. Song, A Study on the Generalized Key Agreement and Password Authentication Protocol, IEICE TRANS. COMMUN., vol. E83-B, no. 9, pp. 2044-2050, September 2000.
6. T. Kwon, Ultimate Solution to Authentication via Memorable Password, Presented to IEEE P1363a, May 2000.
7. P. MacKenzie and R. Swaminathan, Secure network authentication with password identification, Presented to IEEE P1363a, August 1999.
8. D. Jablon, Extended password key exchange protocols, WETICE Workshop on Enterprise Security, 1997.
9. T. Wu, Secure remote password protocol, Internet Society Symposium on Network and Distributed System Security, pp.97-111, 1998.
10. T. Kwon and J. Song, Secure agreement scheme for gxy via password authentication, Electronics Letters, vol. 35, no. 11, pp. 892-893, May 1999.
11. P. MacKenzie, S. Patel, and R. Swaminathan, Password-authenticated key exchange based on RSA, In ASIACRYPT 2000, pp.599-613, 2000.
12. M. Bellare and P. Rogaway, Random oracles are practical : A paradigm for designing efficient protocols, In 1st ACM Conference on Computer and Communications Security, pp. 62-73. November 1993.
13. IEEE, Standard Specifications for Public Key Cryptography, IEEE1363, 2002.

GRIBB – Branch-and-Bound Methods on the Internet*

Randi Moe

Department of Informatics, University of Bergen, Norway,
randi.moe@ii.uib.no

Abstract. The *GRIBB* project (GReat International Branch-and-Bound search) is seeking to establish a software library for solving a large class of parallel search problems by the use of numerous computers around the world accessed by Internet. *GRIBB* is a lean JAVA-based software package targeting compute intensive combinatorial optimization problems solved by the use of branch-and-bound methods. The Quadratic Assignment Problem has been chosen as the first major test application due to its high computation complexity, and preliminary results from test runs are presented.

1 Introduction

Having witnessed how the right standards and software harnessed the use of Internet as the Information Highway, the research community has turned their eyes to the Internet as a compute engine. The combined computing power of the computers connected by Internet exceeds by far the computing power of any of the expensive teraflop supercomputers now available. Again, establishing standards and developing software is the way to go. We are now seeing a rich and enthusiastic research into these problems labeled as 'grid computing' [4,6]. A wide variety of problems are addressed and various strategies are explored, aiming at numerous potential applications. In the *GRIBB* project (GReat International Branch-and-Bound search) we focus on one particular kind of grid computing aiming at large scale, low-cost computation: Harvesting idle cycles on the Internet for large scale, parallel computations.

Internet computing seeks to exploit otherwise idle workstations and PCs to create powerful distributed computing systems with global reach and supercomputing facilities. Different systems have been introduced to harness idle cycles on computers for useful work. In 1997, the Entropia network [2] was established to apply idle workstations worldwide to problems of scientific interest. An organization, e.g. an Internet access provider, can supply a large amount of computing power to solve a particular application. This can be done by asking computer owners all over the world to donate idle cycles on their computer. Many of these organizations are carrying out computations on "philanthropic" issues, like e.g.

* This work was supported by the NFR grant no. 141772

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1020–1027, 2004.
© Springer-Verlag Berlin Heidelberg 2004

cancer, aids, etc. (ComputeAgainstCancer [1], FightAIDS@home [3]), searching for huge prime number like GIMPS [5] or searching for extra terrestrial life through analyzing radio signals (SETI@home [12]). Others have gone commercial, hoping to make a profit on selling access to the world's idle computers or the software required to exploit idle computers within a company. They all attack specific problems using their own software based on the same fundamental idea of "recycling" otherwise wasted cycles on Internet connected computers.

The *GRIBB* project is based on an ongoing project called *GRISK* [7,10]. In the *GRISK* project software was written for solving a very particular problem, that of finding K-optimal lattice rules for numerical integration, using an Internet based search. Hundreds of computers from more than 20 different countries have been used since the first search was started in 1998 and more than 100 CPU-years have been consumed to produce new scientific results which would have been beyond reach without Internet based computing. The software written to accomplish this is very compact and efficient making use of tools like JAVA and RMI (Remote Method Invocation). Participants have been recruited partly by the help of colleagues around the world and partly through a webpage where visitors are encouraged to download an executable code [7].

2 GRIBB

The *GRIBB* project is built on the experiences from *GRISK,* applying the same basic philosophy and core technology to a wider range of problems. However, the framework can not be used on any kind of applications. Our target has been branch-and-bound methods for solving large scale combinatorial optimization problems. The global problem must be decomposable into a large number of independent subproblems of a certain size. Coarse grained parallelism is needed in order to keep the communication costs low. The *GRIBB* framework is a lean software package which has several advantages due to its very simple structure and very little overhead. How general can we make the software framework and still keep the simple and effective structure? And how much work will be needed to apply the framework to new application problems? These questions can not be fully answered until several applications have been tested within this framework.

2.1 Software Structure

The design of the *GRIBB* software is based on a server-client model where the server process is running on a local computer and client processes are possible spread all over the world. The client processes can run on arbitrary computers as long as they are Internet-connected. The server keeps a pool of independent tasks and assigns tasks to the clients as they report to be idle, i.e. when they request a task. Tasks are sent between the server and the clients (see Fig. 1) and contain all necessary information to be transfered between the server and the individual clients in order to solve the given task. When the client has completed the computation the task is returned to the server with the computed result.

The *GRIBB* software package is not dependent on any other software components than JAVA, RMI, and the problem solving code. In order to maintain the philosophy of running clients on arbitrary computers connected to Internet, the problem solving code has to be either self-developed or public domain software in order to avoid any licensing problems.

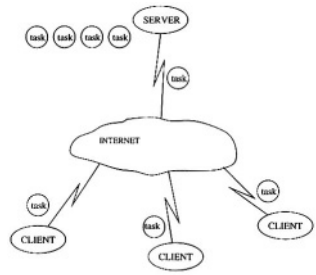


Fig. 1. The GRIBB structure.

The server does not keep any information about the clients except for logging the hostname after requests are received. The clients have no knowledge whatsoever of other clients. Communication is only between the server and the individual clients and the initiative is completely in the hands of the clients. Consequently, the server has no knowledge of the capacity of the clients and does not perform any explicit resource management. Tasks are assigned in the order they appear in the pool-of-tasks without considering the size of the task compared to the capacity of the client. In addition, the *GRIBB* software package does not have any migration of code in the case of client machines being overloaded by other tasks or being closed down. This reduces the overhead and does not create a lot of potentially unnecessary net traffic.

In order to be fault tolerant, the *GRIBB* software has to deal with potential breakdowns either on the server side or on the client side. In the case of a server breakdown, the server will have to be restarted based on logfiles. Clients who try to contact the server during a breakdown period, retries after a given interval until contact is reestablished. In the case of a client breakdown no immediate action is taken as the server has no means to determine a breakdown. However, when all tasks as assigned but not completed, the uncompleted tasks are reassigned to clients which are requesting a new task. In this way both potential client breakdown as well as the case of overloaded or slow clients are dealt with. Other large Internet computing projects have experienced frauds on the client side. The *GRISK* project added simple correctness control of returned answers in order to prevent fraud on the client side. Similar problem dependent control can also be included in *GRIBB*.

The server process is the single point of contact for all client processes and hence the risk of the server process being a bottleneck is obvious. But as the initiative is taken by the clients very few unfulfilled requests are experienced.

Very little computations are needed when handling requests for new tasks as well as when finished tasks are returned. In large simulations the size of each task is expected to be so large that the communication between each client and the server is so infrequent that the server can facilitate a large number of clients without being overloaded.

The client processes are supposed to run on low priority on computers where other users are in control. The client process does not require any disk capacity for data locally and should not be too memory intensive as it will be moved to the background whenever higher priority jobs are started.

2.2 The Server and Client Processes

The server is expected to be mostly idle after the initial setup phase where input data are read, initial subtasks are defined, and communication links established by the use of RMI (Remote Method Invocation). Based on the problem to be solved an initial pool-of-tasks is established. In contrast to the GRISK problem where the complete pool-of-tasks was established prior to any computation, the pool-of-tasks is now being dynamically defined throughout the computations. A predefined minimum number of tasks are established in the pool-of-tasks prior to involving remote clients in the processing. Hence the server is solving a set of subtasks initially in order to have a suitable set of tasks for distributed processing. This ends the "active" part of the server as it is always the clients who initiate further action. The server enters a passive mode where it waits for requests from clients for a new task or the return of solved tasks.

The client processes go through an endless loop of fetching a task from the server, solving it, and returning the solution to the server. If the client process fails to establish contact with the server, it will retry after a predefined interval.

The communication between the server and the clients is taken care of by two methods for requesting and returning tasks in both the server and the client class. These methods relate by having the client method calling the corresponding server method. In order to avoid simultaneous update of variables by several clients, the methods are declared with the keyword **synchronized.** Collisions are prevented by making these methods **synchronized,** i.e. when one **synchronized** method of an object is called, no other **synchronized** method of that object can be called until the first one finishes and releases the lock.

2.3 Branch-and-Bound Strategies

The server and the client processes are meant to handle as general as possible branch-and-bound methods and will be improved in that direction step by step. The initial problem is described by one single task in the pool-of-tasks, i.e. the root of the enumeration tree (see Fig. 2). When a node in the enumeration tree is solved without having found a feasible solution or having established that no solution is available, a set of new tasks corresponding to the children in the enumeration tree are created and added to the pool-of-tasks. However, a task consisting of only one node of the branching tree, i.e. one relaxation, will in most

cases be too small in practice. By distributing the task to a client the amount of communication will be too large compared to the computation costs. Hence the tasks have to consist of subtrees of a certain size. In order to establish a number of subtrees to distribute, the branching tree has to have a certain width. In order to obtain that, the server process must resolve a minimum number of nodes itself.

If a task assigned to a client corresponds to a subtree, also the client has to perform a branch-and-bound method and not only the relaxation method. Searching strategies can be defined differently on the server and the clients. Width-first search is currently used on the server in order to establish a large number of potentially large subtrees while the clients are using a depth-first search. The idea is to descend as quickly as possible in the enumeration tree to find a first feasible solution. It is only possible to prune the tree significantly with a feasible solution, giving a hopefully good lower bound. Candidates for further decomposition are currently chosen in the order of input.

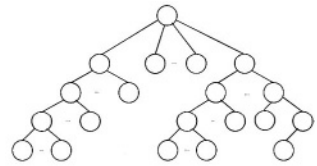


Fig. 2. A branch-and-bound enumeration tree.

3 Quadratic Assignment Problem (QAP)

The Quadratic Assignment Problem (QAP) covers a broad class of problems which involve the minimization of a total pair-wise interaction cost among N facilities. These problems include finding the assignment of factories to fixed locations which minimizes transportation cost and the location of sub-assemblies on a chassis in order to minimize the length of interconnecting wiring. The quadratic assignment problem is among the most complex combinatorial optimization problems. It belongs to the class of NP complete problems and therefore the time spent by any exact algorithm will grow exponentially with the size of the problem.

One version of the problem can be stated with reference to a practical situation where it is desired to locate *N* facilities among *N* fixed locations, where for each pair of facilities (i, k) a certain flow of commodities a_{ik} is known and for each pair of locations (j, n) a corresponding distance b_{jn} is known. The two-way transportation costs between facilities i and k, given that i is assigned to location j and k is assigned to location n, are $a_{ik} \cdot b_{jn} + a_{ki} \cdot b_{nj}$. The objective is to find an assignment minimizing the sum of all such transportation costs. Given $N \times N$ matrices $A = (a_{ij}), B = (b_{ij})$, and $C = (c_{ij})$, QAP(A,B,C,) can be stated as:

$$\min_{\pi \in S_N} \left(\sum_{i=1}^{N} \sum_{j=1}^{N} a_{\pi(i)\pi(j)} b_{ij} + \sum_{i=1}^{N} c_{\pi(i)i} \right),$$

where c_{ij} is the cost of assigning facility i to location j, and S_n is a set of permutations $\{1,2,\ldots,N\}$. The implementation of the branch-and-bound algorithm in *GRIBB* for solving the QAP is based on the method described by Hahn et al [8,9]. The algorithm is based on a Dual Procedure (DP) extending to the Hungarian method for solving the Linear Assignment Problem to QAPs. The DP is utilized within the branch-and-bound algorithm as the auxiliary process for computing lower bounds. The calculation of lower bounds is an iterative process which permits stopping early. The lower bound calculation for a given partial assignment can be stopped as soon as the lower bound on the assumed partial assignment exceeds an upper bound on the original problem. In the reported test cases the DP has been run maximum two iterations.

4 Results

In order to analyze the efficiency of having an increasing number of clients, results from running a Nugent [11] test case, i.e. NUG17 - a problem of size N=17, are presented. The server has been run on a Dell Pentium4 PC running Linux while the clients have been run on either a R10000 node of an SGI Origin 2000 or a Dell Pentium4 PC running Linux. All clients have been within a local area network. The communication is only between the server and the individual clients. A homogeneous set of clients has been chosen in each test to illustrate scalability but it is no requirement. Load balancing is handled implicitly by having a much larger number of tasks than clients.

In order to reduce search effort, the symmetry inherent in the Nugents test cases are taken advantage of in the branch-and-bound method by eliminating "mirror image" partial assignments. The server has computed the nodes of the branch-and-bound enumeration tree two levels down in a width-first manor in order to establish a substantial pool-of-tasks. The server has computed 239 nodes and established a pool of 3366 tasks to be solved by the clients. Two different initial bounds have been used - firstly, an initial bound based on the simplest heuristic, i.e. the given order of input, and secondly, the best known bound for the case. The latter bound indicates the situation when a solution, e.g. given by some heuristic, is to be proved optimal.

In order to evaluate the efficiency of the distributed part of the code, i.e. the computations after the server has established a sufficient pool-of-tasks, Fig. 3 presents the speedup of this part of the code going from 1 to 16 clients in the case of using Pentium4 clients and 1 to 32 clients in the case of using R10000 clients. The initial phase on the server takes in this case only an additional 2 minutes runtime.

In the case of proving the optimality of a computed bound, introducing several clients does not affect the performance of the branch-and-bound method much as shown in the first part of Fig. 3. In the case of a simple initial bound one

has to expect some increase in the number of nodes computed when distributing the subtrees as the current best lower bound is not immediately transferred to all clients. This is shown in practice in Table 1. One has to expect a decrease in the parallel efficiency as experienced in the cases illustrated in the second part of Fig. 3. The running time of the NUG17 case has been reduced from almost 9 hours on a single Pentium4 client to about 40 minutes using 16 clients.

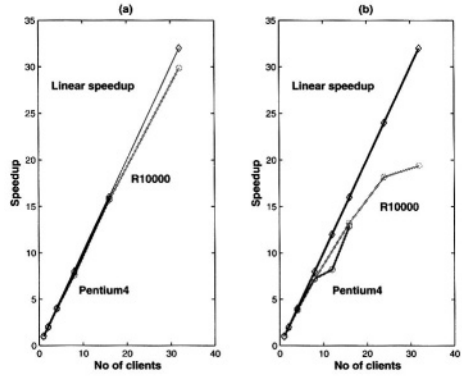


Fig. 3. Speedup for the NUG17 test case on workstations and nodes of a SGI Origin 2000 with (a) a tight initial bound and (b) a simple initial bound.

Table 1. Number of fathomed nodes and load on the server with increasing number of clients for test case NUG17.

No. of clients	No. of fathomed nodes	No. of requests per sec.
1	4,286,737	0.10
2	4,394,464	0.21
4	4,597,281	0.40
8	5,103,829	0.76
12	5,416,841	0.86
16	5,518,330	1.35

The computations have been up to nine times faster on the Pentium4 clients than on the R10000 clients. This gives two quite different computation-communication ratios as the same local network is used, however, the speedup is comparable for similar numbers of clients. This indicates that the communication costs are relatively small and also supports our claim that the loss of efficiency in our test cases is mainly due to an increase in the number of nodes computed.

A large portion of the subtasks of NUG17 are quite small. This can be illustrated by the very frequent requests sent to the server even with a modest number of clients (see Table 1). However, the computational complexity grows

rapidly with increasing problem sizes and the NUG20 (N = 20) needed around 20 clients in order to finish within the same time as NUG17 on a single client. The size of each subtask was larger and as a consequence the load on the server was less than doubled. This indicates that with larger test cases quite a few more clients can be utilized without fearing the server being a bottleneck.

5 Conclusions

The QAP is a special type of problem well suited for distributed computing as it can be separated into a large number of independent tasks. The current GRIBB software shows promising scalability for solving the Nugent test cases of the Quadratic Assignment Problem. Based on the test runs so far it seems reasonable to expect that a lot more clients can be utilized without having communication costs destroying the performance or suffering congestion in the server when larger problems are attacked. More applications are needed in order to fully evaluate the generality of the framework.

References

1. Compute Against Cancer, *http://www.parabon.com/cac.jsp/.*
2. Entropia, *http://www. entropia. com/.*
3. FightAIDS@home, *http://www.fightaidsathome.org/.*
4. I. Foster and C. Kesselman, eds.: The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publisher, Englewood Cliffs, New Jersey, 1998.
5. GIMPS - Great Internet Mersenne Prime Search, *http://www.mersenne.org/.*
6. The Global Grid Forum, *http://www.gridforum.org/.*
7. GRISK: the GReat International Search for K-optimal lattices, *http://www.ii.uib.no/grisk/.*
8. P. Hahn and T. Grant: Lower Bounds for the Quadratic Assignment Problem Based Upon a Dual Formulation. Operations Research, 46 (1998).
9. P. Hahn, T. Grant, and N. Hall: A Branch-and-Bound Algorithm for the Quadratic Assignment Problem Based on the Hungarian Method. European Journal of Operational Research, (1998).
10. J. F. Myklebust and T. Sørevik: GRISK: An Internet based search for K-optimal Lattice Rules, in Lecture Notes in Computer Sciences 1947, T. Sørevik, F. Manne, R. Moe, and A. Gebremedhin, eds., Philadelphia, 2000, Springer Verlag, pp. 196–205. Proceedings from PARA2000, Bergen, June, 2000.
11. QAPLIB - A Quadratic Assignment Problem Library, *http://www.seas.upenn.edu/qaplib/.* R.E. Burkard, E. Çela, S.E. Karisch and F. Rendl, Eds.
12. SETI@home, *http://www.setiathome.ssl.berkeley.edu/.*

Parallel Modular Multiplication Algorithm in Residue Number System

Hyun-Sung Kim[1], Hee-Joo Park[1], and Sung-Ho Hwang[2]

[1] Kyungil University, Computer Engineering,
712-701, Kyungsansi, Kyungpook Province, Korea
kim@kiu.ac.kr
[2] Pohang University of Sci. and Tech., Dept. of Computer Eng. and Sci.,
790-784, Pohangsi, Kyungpook Province, Korea

Abstract. This paper presents a novel method for the parallelization of the modular multiplication algorithm in the Residue Number System (RNS). The proposed algorithm executes modular reductions using a new lookup table along with the Mixed Radix number System (MRS) and RNS. MRS is used because algebraic comparison is difficult in RNS, which has a non-weighted number representation. Compared with the previous algorithm, the proposed algorithm only requires L moduli which is half the number needed in the previous algorithm. Furthermore, the proposed algorithm reduces the number of MUL operations by 25 %.

1 Introduction

Many Cryptosystems employ modular multiplications with very large numbers of more than 512 bits [1]. Various algorithms have been proposed for such operations in previous literature [4] through [14]. Some of these algorithms use a weighted number system to perform the multiplications [12] [13]. However, the Residue Number System (RNS) is of particular interest here because of its inherent property that both addition and multiplication can be executed extremely fast without the need for any carry propagation [2-11][14]. A variety of RNS multipliers are already available including pure table lookup multipliers, quarter square multipliers, index transform multipliers, and array multipliers [4] [6] [7] [10].

Since RNS is not a weighted number system where each digit corresponds to a certain weight, magnitude comparison and division are both hard problems. Accordingly, an efficient method for modular reduction is needed for the performance of modular multiplication. Because such a method will require magnitude comparison operations, the RNS would not seem to be well suited for this algorithm. However, if the Mixed Radix number System (MRS) is used in combination with RNS, an efficient modular reduction can be obtained. Furthermore, this association of MRS with RNS can be defined using the same moduli base. A RNS Montgomery modular multiplication algorithm (MMM) was previously proposed in [5] where Montgomery's method applied to MRS. An auxiliary residue system is also used to solve the data loss problem that occurs at each

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1028–1033, 2004.
© Springer-Verlag Berlin Heidelberg 2004

loop. As a result, this algorithm requires $2L$ moduli because of the additional L moduli needed for the auxiliary base.

This paper proposes a parallel modular multiplication algorithm in RNS, which uses a new lookup table in conjunction with MRS and RNS for the modular reduction. MRS is employed because algebraic comparison is difficult in RNS, which has a non-weighted number representation. The proposed algorithm only requires L moduli, which is half the number of moduli used previously, because there is no need for an additional L moduli for the auxiliary base.

2 Number System

RNS is a carry free system and is potentially very fast even though the advantages of the weighted number system do not carry over. However, algebraic comparison, overflow detection, and sign detection are all difficult, and division is awkward. RNS is defined as follows : The vector $m_1, m_2, ..., m_L$ forms a set of moduli where the m_i's are relatively prime; M is the value of the product of $m_1 \times m_2 \times ... \times m_L$; The vector $m_1, m_2, ..., m_L$ is the RNS representation of X, an integer less than M, where $x_i = X \bmod m_i$.

Let the bits for a prime number N be k, denoted by $\mathrm{bit}(N) = k$. Integer A and B are represented using k-1 bits. In order to simplify the analysis, let $k = lw$. For efficiency reasons, each m_i is selected so that each residue x_i is represented using at most w bits or 1 word, i.e., $\mathrm{bit}(x_i) = d_i < w$. The word size w depends on the computer and is usually taken as 8, 16, or 32. Therefore, a RNS representation is constructed by identifying the L pairwise relatively prime m_i, each of which consists of w bits, such that $\mathrm{bit}(M) = Lw \geq 2k$. As $k = sw, L \geq 2s$. The reason for selecting the RNS range as twice the size of the input is related to the need to represent the product result of the two operands uniquely.

The bits for the result of the addition or subtraction cannot be larger than the maximum bits of the operands. However, in the multiplication $C = A \times B$ the bits required for the resulting integer increases. Therefore, the integer needs to be reduced modulo to a prime number N in order to obtain the product $C = A \times B$ in $\mathrm{bit}(C) = k$. This, if residue arithmetic is to be used for the multiplication modulo N, a method needs to be devised to reduce the resulting integer modulo N. Accordingly, a table lookup reduction method is used to perform this reduction, which is described and analyzed in the following sections.

MRS associated with RNS can be defined using the same moduli base. Assuming that $< x_L, x_{L-1}, ..., x_1 >$, $0 \leq x_i < m_i$ is the MRS representation of X, an integer less than M, then $X_{MRS} = x_L m_{L-1} m_{L-2} ... m_1 + ... + x_3 m_2 m_1 + x_2 m_1 + x_1$ where m_i are the radices and x_i are the mixed radix digits and $0 \leq x_i < m_i$. The conversion from a RNS to a weighted representation is based on the mixed radix conversion procedure [2][3]. Given an RNS representation, the Mixed Radix Number Conversion (MRC) algorithm is used to compute X_{MRS}.

[Algorithm 1] Mixed Radix Number Conversion Algorithm
Input : $X_{RNS} = (x_1, x_2, ..., x_L)$
Output : $X_{MRS} =< x'_L, x'_{L-1}, ..., x'_1 \geq x'_L m_{L-1} m_{L-2} ... m_1 + ... + x'_2 m_1 + x'_1$
Auxiliary : $\overline{m_i} = \frac{1}{m_i} mod m_j, j = i+1, ..., L$
Step 1. for $i = 1$ to L-1 do
Step 2. $x'_i = x_i$
Step 3. $X_{RNS} = X_{RNS} - x_i \bmod m$
Step 4. $X_{RNS} = X_{RNS} * \overline{m_i} \bmod m$
Step 5. $x'_L = x_L$
Step 6. return $X_{MRS} =< x'_L, x'_{L-1}, ..., x'_1 >$

For efficient expression, the symbol < > denotes that the enclosed digits are mixed radix digits, whereas ( ) denotes residue digits.

3 Table Lookup Method for Modular Reduction

To construct the reduction table, first, using a prime integer *N*, $\text{bit}(N) = k$, all multiples with bits less than $k+w$ are computed. Consider the set Q_w and I_w of all integers of length w as $Q_w = I_w = \{0, 1, 2, ..., 2^w - 1\}$. Let $q_i \in Q_w$ and $i \in I_w$. q_i is determined by integer i as $\sum_{j=0}^{w-1} i_j x^j$. The table T which contains 2^w rows can be constructed for $V_i = q_i N$ for $i \in I_w$. The most significant word of V_i is then used for the index of the table T. It is important that the most significant words of V_i for all $i \in I_w$ are all unique.

MRS is a weighted number system and hence a magnitude comparison can be easily performed. The conversion from a RNS to a particular MRS is relatively fast in residue computers. Table *T,* which stores the RNS and MRS forms of the multiples of *N* is used for reducing the integer modulo *N*. Let the multiplication result *C* be the number of length $lw + w$ denoted by $C =< c_l c_{l-1} ... c_1 c_0 >$ which is to be reduced. To reduce the length of the result to *lw,* the reduction algorithm computes *C* mod *N*. In order to reduce the result, the entry $< v_{i,l-1} v_{i,l-2} ... v_{i,1} v_{i,0} >$ from the table *T* is then selected as follows :

$C =< c_l c_{l-1} ... c_1 c_0 > - < v_{i,l-1} v_{i,l-2} ... v_{i,1} v_{i,0} >=< c'_{l-1} c'_{l-2} ... c'_1 c'_0 >$
where $c'_j = c_j - v_j$ for $0 \leq j < l-1$.

A table for the modular reduction can then be constructed by taking a number q_i from Q_w, multiplying it by N to obtain $V_i = q_i N$, and then placing V_i in the table using the most significant word as the index. The relation table between an MRS and an RNS representation, T_{MRS} and T_{RNS}, is shown in Table 1. All of the most significant words are zeros of the T_{MRS} in Table 1. The T_{MRS} can then be reduced to T'_{MRS}, which uses the second word of T_{MRS} as the index.

4 Modular Multiplication Algorithm

This section presents a new parallel modular multiplication algorithm on RNS using the table lookup reduction method. The parallel modular multiplication

Table 1. Table T for modular reduction.

q_i, i	T_{MRS}	T_{RNS}	T'_{MRS}
0	<0,0,0,0>	(0, 0, 0, 0)	< 0, 0 >
1	<0,1,0,6>	(6, 6, 9, 3)	< 0, 6 >
2	<0,2,0,12>	(12,12, 2, 6)	< 0, 12 >
3	<0,3,1,5>	(5, 3,11, 9)	< 1, 5 >
4	<0,4,1,11>	(11, 9, 4, 1)	< 0, 11 >
5	<0,5,2,4>	(4, 0,13, 4)	< 2, 4 >
6	<0,6,2,10>	(10, 6, 6, 7)	< 2, 10 >
7	<0,7,3,3>	(3,12,15,10)	< 3, 3 >
8	<0,8,3,9>	(9, 3, 8, 2)	< 3, 9 >
9	<0,9,4,2>	(2, 9, 1, 5)	< 4, 2 >
10	<0,10,4,8>	(8, 0,10, 8)	< 4, 8 >
11	<0,11,5,1>	(1, 6, 3, 0)	< 5, 1 >
12	<0,12,5,7>	(7,12,12, 3)	< 5, 7 >
13	<0,13,6,0>	(0, 3, 5, 6)	< 6, 0 >
14	<0,14,6,6>	(6, 9,14, 9)	< 6, 6 >
15	<0,15,6,12>	(12, 0, 7, 1)	< 6,12 >

algorithm is given as following algorithm 2.

[Algorithm 2] Parallel Modular Multiplication Algorithm based on RNS
Input : $A_{RNS} = (a_1, a_2, ..., a_L), B_{RNS} = (b_1, b_2, ..., b_L)$
Output : $C_{RNS} = (c_1, c_2, ..., c_L)$
Auxiliary : $T, MAX[i]$
Step 1. $C_{RNS} = A_{RNS} * B_{RNS}$
Step 2. $M_{MRS} = MRC(C_{RNS})$
Step 3. for $i = 2l - 1$ downto l do
Step 4. $C_{RNS} = C_{RNS} - T[x'_i] * MAX[i]$
Step 5. $X_{MRS} = X_{MRS} - T[x'_i] \times MAX[i]$
Step 6. return C_{RNS}

Let A and B be two input with at most $k-1$ bits. In Step 1, each digit in a RNS representation is multiplied in parallel. After Step 1, C is represented at most $2k-2$ bits. The representation is still unique in RNS because the bits for required M are at least $2k$ and the MRC algorithm will yield a unique result. However, the result C cannot be used as an input for next multiplications. Thereby, a modular reduction operation is necessary. Steps 2-5 can achieve this operation. First the MRS representation, X_{MRS} for a weighted number system is computed from C using algorithm 1. Then the table lookup reduction makes bit(C) becomes less than k. The array element $MAX[2l-1]$ at Step 4 and 5 stores the maximaximum modulus, $\max(m_i)$, $i = 1, 2, ..., L$ and the others have

1. The symbol * denotes that the multiplication operation is computed on RNS, whereas × denotes on MRS.

Table 2. Comparison

	J. C. Bajard in [5]	Proposed
TREAD	0	$2l$
MUL	$4l$	$3l+1$
XOR	$4l$	$4l$
DIV	$2l$	0

5 Analysis

An analysis is made of the proposed algorithm and its performance is compared with that of the MMM algorithm by J. C. Bajard in [5]. The MMM algorithm is based on Montgomery's method using MRS and RNS. Since each loop of the MMM algorithm requires division, the use of RNS alone is very difficult, as such, MRS is used for the division. However, division of the ith residue cannot be computed because m_i is not relatively prime to itself. Thus the ith residue is lost. An auxiliary residue system is therefore used to solve the loss of residue, however, it needs twice the number of moduli, $2L$ moduli, compared with the proposed approach because of the additional L moduli for the auxiliary base. Although the algorithm in [5] does not require table lookup operations, it still needs divisions. In contrast, the table lookup operation was easily implemented with a combinatorial logic of reasonable complexity. Accordingly, the operation counts indicated that the proposed parallel modular multiplication algorithm was computationally more efficient. For the perspective of the table size, table T have 2^w rows containing a number of length ks. Each entry in table T contained MRS and RNS numbers. So, the total table size is $2 \times 2^w \times k$ bits, which are mainly depending on the word size of the computer.

6 Conclusions

This paper proposed a parallel modular multiplication algorithm in RNS through the use of a new lookup table as a tool for modular reduction. MRS is used for the magnitude comparison since MRS is a weighted number system. The proposed algorithm only requires L moduli, which is half the number of moduli used in the MMM algorithm in [5]. From Table 2, the proposed parallel multiplication algorithm reduced the number of MUL and DIV operations by 25% and 100%, respectively. Further work is currently underway on improving the proposed algorithm and implementing hardware design of the proposed parallel modular multiplication algorithm.

References

1. W. Diffie, M. Hellman, New Directions in Cryptography, IEEE Trans. on Info. Theory, vol. IT-22(6), pp. 644-654, 1976.
2. N.S. Szabo, R.I. Tanaka, Residue Arithmetic and Its Applications to Computer Technology, McGraw-Hill, New York, 1967.
3. F.J. Taylor, Residue Arithmetic: A Tutorial with Examples, Computer, pp. 50-62, May 1984.
4. K.M. Elleithy, M.A. Bayoumi, A Systolic Architecture for Modulo Multiplication, IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 42, no. 11, pp. 725-729, Nov. 1995.
5. J.C Bajard, L.S. Didier, P. Kornerup, An RNS Montgomery Modular Multiplication Algorithm, IEEE Trans. on Computers, vol. 47, no. 7, pp. 766-776, July 1998.
6. D. Radhakrishnan, Y. Yuan, Novel Approaches to the Design of VLSI RNS Multipliers, IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 39, no. 1, pp. 52-57, Jan. 1992.
7. G. Alia, E. Martinelli, A VLSI Modulo m Multiplier, IEEE Trans. on Computers, vol. 40, no. 7, pp. 873-878, July 1991.
8. F.J. Taylor, A VLSI Residue Arithmetic Multiplier, IEEE Trans. on Computers, vol. C-31, no. 6, pp. 540-546, June 1982.
9. G.A. Jullien, Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms, IEEE Trans. on Computers, vol. C-29, no. 10, pp. 899-905, Oct. 1980.
10. M. Soderstrand, W.K. Jenkins, G.A. Jullian, F.J. Taylor, Residue Number Systems: Modern Applications in Digital Signal Processing, New York, IEEE, 1986.
11. V.S. Dimitrov, G.A. Jullien, W.C. Miller, A Residue Number System Implementation of Real Orthogonal Transforms, IEEE Trans. on Signal Processing, vol. 46, no. 3, pp. 563-570, March 1998.
12. H.S. Kim, S.W. Lee, K.Y. Yoo, Partitioned Systolic Multiplier for $GF(2^m)$, Information Processing Letter, vol. 76, pp. 135-139, 2000.
13. H. S. Kim, Bit-Serial AOP Arithmetic Architecture for Modular Exponentiation, Ph. D. Thesis, Kyungpook National Univ., 2002.
14. A. Halbutogullari, C.K. Koc, Parallel Multiplication in $GF(2^k)$ using Polynomial Residue Arithmetic, Design, Codes and Cryptography, vol. 20, no. 2, pp. 155-173, 2000.

A Combined Fractional Step Domain Decomposition Method for the Numerical Integration of Parabolic Problems

Laura Portero, Blanca Bujanda, and Juan Carlos Jorge

Dpto. Matemática e Informática, Universidad Pública de Navarra,**
Campus Arrosadía s/n, 31.006, Pamplona (Spain)
{laura.portero, blanca.bujanda, jcjorge}@unavarra.es

Abstract. In this paper we develop parallel numerical algorithms to solve linear time dependent coefficient parabolic problems. Such methods are obtained by means of two consecutive discretization procedures. Firstly, we realize a time integration of the original problem using a Fractional Step Runge Kutta method which provides a family of elliptic boundary value problems on certain subdomains of the original domain. Next, we discretize those elliptic problems by means of standard techniques. Using this framework, the numerical solution is obtained by solving, at each stage, a set of uncoupled linear systems of low dimension. Comparing these algorithms with the classical domain decomposition methods for parabolic problems, we obtain a reduction of computational cost because of, in this case, no Schwarz iterations are required. We give an unconditional convergence result for the totally discrete scheme and we include two numerical examples that show the behaviour of the proposed method.

1 Introduction

It is well known that the numerical resolution of multidimensional parabolic problems by using standard methods requires a strong computational effort, specially if very accurate solutions are wanted. The development and analysis of such methods can be done by means of the combination of two discretization stages: a time integration procedure (f.e. via Runge-Kutta or multistep methods) and a spatial discretization method like finite differences or finite elements. For the time integration it is usually chosen between the two following possibilities: an explicit method that provides totally discrete schemes which are cheap per time step in terms of computational cost, but have the disadvantage of being conditionally convergent (that is to say, there is a strong restriction between the time step and the spatial grid sizes to obtain convergence) or an implicit method. In the last case we obtain totally discrete algorithms that are unconditionally

** This research is partially supported by the MCYT research project num. BFM2000-0803 and the research project resolution 134/2002 of Government of Navarra.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1034–1041, 2004.
© Springer-Verlag Berlin Heidelberg 2004

convergent but have the drawback of involving one or several large linear systems per time step, whose resolution implies a high computational cost.

In order to accelerate the resolution of such systems, specially if we use parallel computer devices, one of the most successful techniques is the domain decomposition (see [7]). Such technique permits to reduce the original system to a set of systems of lower dimensions, as well as to parallelize the resolution of them. When classical implicit methods are used to discretize the time variable, the user must assume the cost of a Schwarz iterative process if overlapped subdomains are taken; in order to get a fast convergence for these iterative processes, it is essential to choose carefully the dimensions of the overlapping zones (see [7]). In the case of non-overlapped subdomains, another kind of iterations are needed; these ones are related to the transmission conditions that must be imposed between the boundaries of the subdomains.

In this paper we propose to use a Fractional Step Runge-Kutta (shortly FSRK, see [3]) method for the time discretization process, in which the operator splitting is subordinated to the decomposition of the domain considered. After that, the numerical algorithm will be easily obtained if we consider a standard discretization method for the spatial variables (for example finite difference or finite element methods). Following this technique, we only have to solve several linear systems per stage of low dimensions that can be computed in parallel without the need of doing any kind of iteration. In [5], the authors develop a similar idea for the case of combining classical (low-order) FSRK methods with finite differences to integrate linear parabolic problems with constant coefficients.

In section 3, we prove that a numerical method of this kind preserve the property of unconditional convergence, which is typical for suitable implicit methods. The last section contains two numerical experiments for one-dimensional and two-dimensional problems integrated both with this kind of methods.

2 Obtaining the Totally Discrete Scheme

Let us consider the linear parabolic initial boundary value problem that consists in finding $u : \Omega \times [t_0, T] \to \mathcal{H}$ such that

$$\begin{cases} \dfrac{du}{dt} = A(\overline{x}, t)u + f(\overline{x}, t), \quad (\overline{x}, t) \in Int(\Omega) \times (t_0, T], \\ u(\overline{x}, t_0) = u_0(\overline{x}) \in \mathcal{H}, \quad \overline{x} \in \Omega, \\ Bu(\overline{x}, t) = g(\overline{x}, t) \in \mathcal{H}^b, \quad (\overline{x}, t) \in \partial\Omega \times (t_0, T], \end{cases} \tag{1}$$

where $\mathcal{H}$ and $\mathcal{H}^b$ are Hilbert spaces and, for each $t \in [t_0, T]$, $A(\overline{x}, t) : \mathcal{D} \subseteq \mathcal{H} \to \mathcal{H}$ is an unbounded elliptic differential operator that involves the partial derivatives of the unknown u with respect to the spatial variables.

In order to dicretize in time problem (1) using an FSRK method, we introduce a partition for the elliptic operator in the form $A(\overline{x}, t) = \sum_{i=1}^{m} A_i(\overline{x}, t)$ and for the source term $f(\overline{x}, t) = \sum_{i=1}^{m} f_i(\overline{x}, t)$ that will be specified later. Classically, when the elliptic operator does not contain any crossed derivative, the splitting is

taken by grouping in each addend the derivatives with respect to each coordinate (obtaining a method of type Alternating Directions).

An FSRK method, considering τ as (constant) time step, provides a numerical approximation of the solution of the semidiscrete problem at each time t_n, $u_n(\overline{x}) \approx u(\overline{x}, t_n)$, by solving

$$\begin{cases} \begin{cases} U^{n,j} = u_n + \tau \displaystyle\sum_{k=1}^{j} a_{jk}^{i_k} \left(A_{i_k}(\overline{x}, t_{n,k}) U^{n,k} + f_{i_k}(\overline{x}, t_{n,k}) \right), \\ B_{i_j} U^{n,j} = g(\overline{x}, t_{n,j}), \qquad \text{for } j = 1, \ldots, s, \end{cases} \\ u_{n+1} = u_n + \tau \displaystyle\sum_{j=1}^{s} b_j^{i_j} \left(A_{i_j}(\overline{x}, t_{n,j}) U^{n,j} + f_{i_j}(\overline{x}, t_{n,j}) \right), \end{cases} \tag{2}$$

where $t_n = t_0 + n\tau$ and $t_{n,j} = t_n + c_j \tau$. With the aim of using a more comfortable matrix notation, we can consider an FSRK method as an Additive Runge-Kutta method (see [4]) with many null columns (in particular, we consider $a_{jk}^i = b_j^i = 0$ for $i \neq i_k$) and, using a notation similar to Butcher's table for classical Runge-Kutta methods, we can express it in the following compacted form

$$\begin{array}{c|c|c|c|c} c & \mathcal{A}_1 & \mathcal{A}_2 & \ldots & \mathcal{A}_m \\ \hline & b_1^T & b_2^T & \ldots & b_m^T \end{array},$$

where $\mathcal{A}_i = (a_{jk}^i) \in \mathbb{R}^{s \times s}$ and $b_i = (b_j^i)$, $c = (c_1, \ldots, c_s)^T \in \mathbb{R}^s$ for $i = 1, \ldots, m$.

The advantage that these methods may provide, in comparison with classical implicit methods, comes from the fact that the calculus of each stage $U^{n,j}$ is done by solving linear elliptic boundary value problems of the form

$$\begin{cases} (\mathcal{I} - \tau\, a_{jj}^{i_j} A_{i_j}(\overline{x}, t_{n,j})) U^{n,j} = F^{n,j}, \\ B_{i_j} U^{n,j} = g(\overline{x}, t_{n,j}), \end{cases} \tag{3}$$

where $F^{n,j}$ is computed from the data of the problem and the results of previous stages. If operators $A_i(\overline{x}, t)$ are simpler than the global operator $A(\overline{x}, t)$, then a suitable spatial discretization of these problems permits the obtaining of numerical algorithms which are much cheaper than classical implicit methods.

In this work we propose a smooth splitting of $A(\overline{x}, t)$ related to a decomposition of domain Ω in the form $\Omega = \bigcup_{i=1}^{m} \Omega_i$, where each subdomain Ω_i consists of a set of m_i disjoint components Ω_{ij} satisfying $\Omega_i = \bigcup_{j=1}^{m_i} \Omega_{ij}$. Concretely, we consider $A_i(\overline{x}, t) = \psi_i(\overline{x}) A(\overline{x}, t)$, $f_i(\overline{x}, t) = \psi_i(\overline{x}) f(\overline{x}, t)$, where $\psi_i(\overline{x})$ is a sufficiently smooth function which satisfies

$$\begin{array}{l} \psi_i(\overline{x}) = 0 \ \text{ if } \overline{x} \in \Omega \setminus \Omega_i, \quad \psi_i(\overline{x}) = 1 \ \text{ if } \overline{x} \in \Omega_i \setminus \bigcup_{\substack{j=1 \\ j \neq i}}^{m} (\Omega_i \cap \Omega_j), \\ \text{and } \psi_i(\overline{x}) = h_i(\overline{x}) \ \text{ if } \overline{x} \in \bigcup_{\substack{j=1 \\ j \neq i}}^{m} (\Omega_i \cap \Omega_j), \\ \text{with } 0 \leq h_i(\overline{x}) \leq 1 \text{ and } \sum_{i=1}^{m} h_i(\overline{x}) = 1 \ \forall\, \overline{x} \in \bigcup_{\substack{j=1 \\ j \neq i}}^{m} (\Omega_i \cap \Omega_j). \end{array} \tag{4}$$

We include below, in Figure 1, an example of a two-dimensional domain which is decomposed in four subdomains and each one of them consists of four components, i.e. $m = 4$, $m_i = 4\ \forall i = 1, \ldots, 4$. We have also included, in the same figure, a graphic representing a possible function $\psi_4(x, y)$ that would be associated to the subdomain Ω_4.

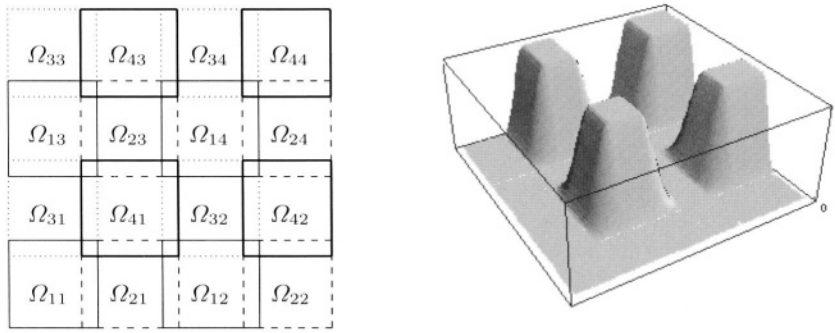


Fig. 1. Domain decomposition in four subdomains

After the time integration, we proceed to discretize in space the resultant family of elliptic boundary value problems (2) by means of classical methods. Let us consider a spatial discretization parameter h which is going to tend to zero. For each value of h, we consider a mesh Ω_h (of the original domain Ω) which has been constructed taking into account the boundaries of every subdomain Ω_i and we denote with $\mathcal{H}_h$ a finite dimensional space made up of discrete functions in Ω_h, if we use finite differences, or piecewise polynomial functions associated to the meshing Ω_h, if we use finite elements. We denote with $A_{ih} : \mathcal{H}_h \to \mathcal{H}_h$ the operators that discretize A_i and with $f_{ih} = \pi_h f_i$, $g_h = \pi_h g$ the restrictions or projections of the continuous functions f_i, g, depending on the spatial discretization method used.

In this context, the totally discrete scheme can be expressed as follows

$$\begin{cases} \begin{cases} U_h^{n,j} = u_{h,n} + \tau \displaystyle\sum_{k=1}^{j} a_{jk}^{i_k} (A_{i_k h}(t_{n,k}) U_h^{n,k} + f_{i_k h}(t_{n,k})), \\ B_{i_j h} U_h^{n,j} = g_h(t_{n,j}), \qquad \text{for } j = 1, \ldots, m, \end{cases} \\ u_{h,n+1} = u_{h,n} + \tau \displaystyle\sum_{j=1}^{s} b_j^{i_j} (A_{i_j h}(t_{n,j}) U_h^{n,j} + f_{i_j h}(t_{n,j})). \end{cases} \tag{5}$$

Looking at (5) it is clear that the calculus of each stage involves a linear system of the form

$$(\mathcal{I}_h - a_{jj}^k A_{kh}(t_{n,j}) U_h^{n,j}) = F_h^{n,j}, \quad (k = i_j)$$

which, in practice, implies the resolution of m_k uncoupled linear systems whose sizes depend of the number of mesh points that belong to the components Ω_{ki} of

subdomain Ω_k; this permits a straightforward parallelization for their resolution. On the other hand, it is important to notice that the main advantage of our method is that no Schwarz iterations are required to obtain $u_{h,n+1}$.

3 Global Convergence of the Totally Discrete Scheme

To study the convergence of the totally discrete scheme, we suppose that the elliptic operators $-A(\overline{x},t)$ are maximal, monotone and satisfy the following bound

$$\|A(\overline{x},t)\,u - A(\overline{x},s)\,u\| \leq |t-s|\,M\,\|A(\overline{x},t)u\|, \quad \forall\, t,s \in [t_0,T]. \tag{6}$$

Operators $-A_i(\overline{x},t)$, due to their construction, are also maximal and monotone and verify bounds similar to (6). Henceforth, we denote with $r_h(t), r_{i,h}(t) : \mathcal{D} \to \mathcal{H}_h$ certain restrictions or projections, depending on the spatial discretization used, and with C any constant independent of τ and h.

Moreover, we suppose that the spatial discretization is stable, i.e., the discrete operators $-A_{ih}(t)$ are monotone, and that such discretization is consistent of order r, that is to say, for sufficiently smooth functions $u(\overline{x},t)$, we have

$$\begin{aligned} &\|A_{ih}(t)r_{i,h}(t)u(\overline{x},t) - \pi_h A_i(\overline{x},t)u(\overline{x},t)\|_h \leq C\,h^r, \\ &\|B_{ih}r_{i,h}(t)u(\overline{x},t) - \pi_h B_i u(\overline{x},t)\|_h \leq C\,h^r, \end{aligned} \tag{7}$$

where $\|.\|_h$ denotes a suitable norm in $\mathcal{H}_h$.

We define the global error associated to the total discretization in the moment t_n in the usual form $E_{h,n} = \|r_h(t_n)\,u(\overline{x},t_n) - u_{h,n}\|_h$ and it is said that the totally discrete scheme (5) is convergent, of order p in time and order r in space, if $E_{h,n} \leq C(\tau^p + h^r)$. If the spatial discretization is stable, the totally discrete scheme possesses unique solution which can be expressed in the form (see [3])

$$u_{h,n+1} = \widetilde{R}\big(\tau\,\hat{A}^n_{1h},\ldots,\tau\,\hat{A}^n_{sh}\big)\,u_{h,n} + \widetilde{S}\big(-\tau\,\hat{A}^n_{1h},\ldots,-\tau\,\hat{A}^n_{sh},\tau\,\hat{F}^n_{1h},\ldots,\tau\,\hat{F}^n_{sh}\big),$$

where we group the evaluations of the partitioned source terms and operators as follows $\hat{F}^n_{ih} = (f_{ih}(t_{n,1}),\ldots,f_{ih}(t_{n,s}))^T \in \mathcal{H}_h^s$ and $\hat{A}^n_{ih} = diag\,(A_{ih}(t_{n,1}),\ldots, A_{ih}(t_{n,s})) \in \mathcal{H}_h^{s\times s}$, for $i = 1,\ldots,s$ and $n = 1,2,\ldots$

When operators $A_{ih}(t)$ preserve bounds of type (6) and we choose FSRK methods satisfying certain linear absolute stability properties, the following bound for the linear transition operator can be obtained (see [1])

$$\big\|\widetilde{R}\big(\tau\,\hat{A}^n_{1h},\ldots,\tau\,\hat{A}^n_{sh}\big)\big\|_h \leq e^{\gamma\tau}. \tag{8}$$

In [3] it is proven that, if the time discretization is accomplished with an FSRK method of order p and the problem data in (1) are sufficiently regular and compatible, the scheme (2) is uniformly consistent of order p, that is to say,

$$\|u(t_n) - \breve{u}_n\| \leq C\,\tau^{p+1}, \tag{9}$$

where $\breve{u}_n$ is obtained after a step of the semidiscrete scheme (2) starting from $\breve{u}_{n-1} = u(t_{n-1})$.

To deduce the convergence of (5) we decompose the global error as $E_{h,n} \leq \|r_h(t_n)(u(t_n)-\check{u}_n)\|_h+\|r_h(t_n)\check{u}_n-\check{u}_{h,n}\|_h+\|\check{u}_{h,n}-u_{h,n}\|_h$,where $\check{u}_{h,n}$ is obtained after a step of the totally discrete method (5) taking $\check{u}_{h,n-1} = r_h(t_{n-1})u(t_{n-1})$ as starting point.

The first addend is bounded using (9) together with certain compatibility properties for the continuous and discrete norms (see [3]). The bound for the second addend is deduced combining the consistency and stability of the spatial discretization (see [3]). Finally, using (8), we can write

$$E_{h,n} \leq C\,\tau^{p+1} + C\,\tau\,h^r + e^{\beta\tau}\|E_{h,n-1}\|_h \leq C\,\tau \sum_{k=0}^{n-1} e^{\beta k\tau}\,(\tau^p + h^r) \leq C\,(\tau^p + h^r).$$

4 Numerical Examples

Example a) We consider the parabolic problem

$$\begin{cases} \dfrac{\partial u}{\partial t} = A(x,t)u + f(x,t), \ \forall\,(x,t) \in (0,1)\times(0,2], \\ u(x,0) = u_0(x), \ \forall\, x \in (0,1), \quad u(0,t) = u(1,t) = 0, \ \forall\, t \in [0,2], \end{cases}$$

with $A(x,t) = (1+x)(1+t)e^{-t}\frac{\partial^2}{\partial x^2} - (1+2x)(1+e^{-t})\frac{\partial}{\partial x} - (1+x^2)\mathcal{I}$ and where $f(x,t)$ and $u_0(x)$ are data functions chosen so that $u(x,t) = e^{-t}x^2(1-x)^2$ is the exact solution. In this experiment we consider the spatial domain made up of two overlapped subdomains ($[0,1] = \Omega_1 \cup \Omega_2$, where $\Omega_1 = [0, \frac{1}{2}+d]$, $\Omega_2 = [\frac{1}{2}-d, 1]$, being $d = \frac{1}{8}$).

We integrate this problem following two different ways. On one hand, we use standard SDIRK methods of order 2, 3 and 4 (with 1, 2 and 3 internal stages, respectively) for the time integration together with a central difference discretization of the spatial variables on a uniform grid and we apply the Schwarz iterative method. On the other hand, we combine a time integration by means of FSRK methods of order 2, 3 and 4 (with 2, 6 and 8 implicit stages, respectively, see [2]) with a central difference spatial discretization. Following the ideas presented in the previous sections, we choose a partition for the original elliptic operator $A(x,t)$ in two addends $A_i(x,t) = \psi_i(x)A(x,t)$, $i = 1,2$ with $\{\psi_i(x)\}_{i=1,2}$ a partition of unity (4) associated to the domain decomposition $\{\Omega_i\}_{i=1,2}$, where we have chosen $h_1(x) = \frac{1}{2} - \frac{3}{4d}(x-\frac{1}{2}) + \frac{1}{4d^3}(x-\frac{1}{2})^3$ and $h_2(x) = 1 - h_1(x)$.

In both options we will take the following relations between $N = \frac{1}{h}$ and the time step size: $N^2\tau^2 = 0.16$, $N^2\tau^3 = 8\mathrm{E}-3$ or $N^2\tau^4 = 4\mathrm{E}-4$, depending on wether the consistency order of the time integrator used is 2, 3 or 4, respectively.

The maximum global errors obtained for these six totally discrete methods have been computed as

$$E_{h,n} = \max_{\substack{x_i \in \Omega_h \\ t_n = n\tau,\, n=1,2,\ldots,\frac{2}{\tau}}} |u^i_{h,n} - u(x_i,t_n)|,$$

where $u(x_i,t_n)$ is the exact solution evaluated in the grid node $x_i = ih$ at time $t_n = n\tau$ and $u^i_{h,n}$ is the numerical solution obtained in the same grid point and

at the same time moment. All of them have shown almost the same propagation of global errors and, consequently, the same numerical orders of convergence, so we only include in Table 1 the corresponding errors for the methods with second order in time and in space (FSRK2 refers to the classical Peaceman & Rachford method and RK2 to the Crank-Nicolson method) .

Table 1. Global errors

	N=16	N=32	N=64	N=128	N=256	N=512	N=1024
Example a)							
FSRK2	8.4121E-4	2.0942E-4	5.2304E-5	1.3072E-5	3.2678E-6	8.1695E-7	2.0424E-7
RK2	8.3353E-4	2.1018E-4	5.2758E-5	1.3213E-5	3.3063E-6	8.2692E-7	2.0678E-7
Example b)							
FSRK2	1.6364E-2	4.6055E-3	1.2801E-3	3.3917E-4	8.6694E-5	2.1794E-5	5.4278E-6
RK2	1.2747E-2	3.1979E-3	8.0116E-4	2.0049E-4	5.0130E-5	1.2501E-5	3.1254E-6

Table 2. Number of systems to solve

FSRK	RK	N=16		N=32		N=64		N=128		N=256		N=512		N=1024	
Example a)															
order 2		2	6	2	6	2	6	2	6.1	2	6	2	6	2	4
order 3		6	30.9	6	34.5	6	31.8	6	32.6	6	28.5	6	24.4	6	24.1
order 4		8	76.4	8	81	8	82.2	8	77.6	8	77.7	8	72	8	73.2
Example b)															
order 1		16	96	16	80	16	48	16	48	16	32	16	32	16	32
order 2		24	80	24	88	24	64	24	64	24	48	24	64	24	64

Finally, in order to show the reduction of computational cost achieved with our proposal, we show in Table 2 the average number of linear systems of size $\frac{N}{2} + Nd$ involved, per time step, for each method. This average has been computed in the time interval [0,0.2] which is the most relevant zone attending to the variation of the solution and the time steps considered. In a variable time step integration procedure this average should be watched in the whole interval $[0, 2]$ to compare suitably the efficiencies. Concretely, when we use an FSRK method for the time integration, we have to solve a linear system of $\frac{N}{2} + Nd$ unknowns per stage (and we don't need any kind of iterations), whereas when we use classical domain decomposition techniques we have to solve $2k$ linear systems of size $\frac{N}{2} + Nd$ per stage, where k is the number of Schwarz iterations required (the tolerance imposed for this iterative method has been $10^{-1}\tau h^2$).

Example b) We consider now a two-dimensional problem associated to the equation $\frac{\partial u}{\partial t} = (1+e^{-t})\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - (1+2xy)e^{-2t}u + f(x,y,t)$ in the domain $\Omega = (0,1)\times(0,1)$for $t \in (0,2]$ with homogeneous Dirichlet boundary conditions,

where the source term f and the initial condition u_0 are chosen in order to have $u(x,y,t) = e^{-t+7.5}(\frac{1+e}{e} - e^{-x} - e^{-1+x})^2(\frac{1+e}{e} - e^{-y} - e^{-1+y})^2$ as exact solution.

Now we have considered the spatial domain Ω decomposed as the union of 4 overlapped subdomains (an example of a decomposition of this type can be observed in Figure 1). In this numerical experiment we use a first and a second order FSRK method (with 4 and 6 implicit stages, respectively) both combined with a second order spatial discretization using central differences. We compare their results with the ones obtained with standard RK methods of order 1 and 2 (both with 1 internal stage) combined with a central difference spatial discretization, using the Schwarz iterative method with tolerance $10^{-1}\tau h^2$.

Let us define the following functions of one variable

$$i_1(x) = \begin{cases} 1 \text{ if } x \in [0, \frac{1}{4} - d] \cup [\frac{1}{2} + d, \frac{3}{4} - d], \\ 0 \text{ if } x \in [\frac{1}{4} + d, \frac{1}{2} - d] \cup [\frac{3}{4} + d, 1], \\ \frac{1}{2} - \frac{3}{4d}(x-\alpha) + \frac{1}{4d^3}(x-\alpha)^3 \text{ if } x \in [\alpha - d, \alpha + d], \text{ with } \alpha = \frac{1}{4}, \frac{1}{2}, \frac{3}{4} \end{cases}$$

and $i_2(x) = 1 - i_1(x)$. The splitting operators used in this example are $A_1(x,y) = i_1(x)i_1(y)A$, $A_2(x,y) = i_2(x)i_1(y)A$, $A_3(x,y) = i_1(x)i_2(y)A$ and $A_4(x,y) = i_2(x)i_2(y)A$ and the overlapping zones are again determined by $d = \frac{1}{8}$. In this experiment we consider the relations $N^2\tau = 5.12$ and $N^2\tau^2 = 0.1024$ for the time integrators of first and second order, respectively. Due, again, to the fact that the four totally discrete methods provide very similar maximum global errors, we only include in Table 1 the global errors for the methods with second order in time and in space. At last, in Table 2, we show the difference between the number of linear systems that we have to solve in each case. This number will again depend on the number of internal implicit stages of the time integrator used and on the average number of Schwarz iterations per stage (only for the case of classical domain decomposition techniques).

References

1. Bujanda, B., Jorge, J.C.: Stability Results for Fractional Step Discretizations of Time Dependent Evolutionary Problems. Appl. Numer. Math. **38** (2001) 69–86
2. Bujanda, B.: Métodos Runge-Kutta de Pasos Fraccionarios de Orden Alto para la Resolutión de Problemas Evolutivos de Convección-Difusión-Reacción. Tesis, Universidad Pública de Navarra (1999)
3. Bujanda, B., Jorge, J.C.: Fractional Step Runge-Kutta Methods for Time Dependent Coefficient Parabolic Problems. Appl. Numer. Math. **45** (2003) 99–122
4. Cooper, G.J., Sayfy, A.: Additive Runge-Kutta Methods for Stiff Ordinary Differential Equation. Math. of Comp. **40** (1983) 207–218
5. Mathew, T.P., Polyakov, P.L., Russo, G., Wang, J.: Domain Decomposition Operator Splittings for the Solution of Parabolic Equations. SIAM J. Sci. Comput. **19** No. 3 (1998) 912–932
6. Peaceman, D.W., Rachford, H.H.: The Numerical Solution of Parabolic and Elliptic Differential Equations. J. Soc. Indust. Appl. Math. **3** (1955) 28–42
7. Quarteroni, A., Valli, A.: Domain Decomposition Methods for Partial Differential Equations. Clarendon Press, Oxford (1999)

Incomplete Cholesky Factorization in Fixed Memory

Sergey Saukh

G.Y.Pukhov's Institute of Modeling Problems in Power Engineering,
National Academy of Sciences
General Naumov Street 15, 03164 Kiev, Ukraine
saukh@svitonline.com

Abstract. We propose an incomplete Cholesky factorization for the solution of large positive definite systems of equations and for the solution of large-scale trust region sub problems. The factorization proposed essentially reduces the negative processes of irregular distribution and accumulation of errors in factor matrix and provides the optimal rate of memory filling with the greatest modulo elements. Test results show reducing the number of conjugate gradient iterations even in case of small range of memory usage for Cholesky factor matrix.

1 Introduction

We will consider only the problems with large sparse symmetric matrices. In order to solve them the conjugate gradient iteration methods with preconditioner, which is built on the base of incomplete Cholesky factorization, are used.

Initially proposed for positive definite systems, iteration methods with preconditioner on the base of incomplete Cholesky factorization are now being developed to be applicable to indefinite systems. Indefinite systems may appear in n–measured function minimization problem, where the solution process is reduced to the sequence of solutions of sub problems:

$$\min\left\{B^T X + \frac{1}{2}X^T AX : \|DX\|_2 \le \Delta\right\}, \tag{1}$$

where Δ is the trust region radius, $B \in \mathrm{R}^n$ is the gradient of the function at the current iterate, $A \in \mathrm{R}^{n*n}$ is an approximation to the Hessian matrix, $D \in \mathrm{R}^{n*n}$ is a nonsingular scaling matrix. To solve (1) we generally need to solve indefinite system of linear equations $AX + B = 0$.

To find an approximate solution of large-scale problem (1), it is used a conjugate gradient method with preconditioner, which takes into account the restriction on radius of trust region and the possibility that matrix A is indefinite. If $\|DX_k\|_2 \le \Delta$, then the conjugate gradient method generates sequences $\{X_k\}$ and directions $\{P_k\}$ until one of the next three conditions is satisfied:

$$\|AX_k + B\|_2 \le \delta\|B\|_2, \quad P_k^T AP_k \le 0, \quad \|DX_{k+1}\|_2 > \Delta. \tag{2}$$

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1042–1051, 2004.
© Springer-Verlag Berlin Heidelberg 2004

In all three cases an approximate solution of problem (1) satisfying the three convergence conditions of iteration methods in n-measured trust region is defined. In most problems where Δ is comparatively small, the condition (3) can be satisfied on the first few iterations. For the satisfaction of the first two conditions a higher number of iterations of conjugate gradient method is needed, particularly when A is nearly singular. In this case the second condition can be satisfied only if A is not positively defined. Then P_k is a direction of negative curvature.

Our aim is to reduce the number of iterations that are needed to satisfy the first two conditions in expression (2).

To solve the problem (1) we, as in paper [1], transform the ellipsoidal trust region in the sphere trust region and obtain the following problem

$$min\{b^T x + \frac{1}{2} x^T a x : \|x\|_2 \leq \Delta\} , \tag{3}$$

where $b = D^{-T}B$, $a = D^{-T}AD^{-1}$ Then the approximate solution x of the problem (3) corresponds to the solution X of the problem (1) with relationship $X = D^{-1}x$. As the scaled matrix D clusters the proper values of matrix A, the conjugate gradient method provides the solution of problem (3) in a small number of iterations. Matrix D is formed on the base of Cholesky factorization.

2 Incomplete Factorization

The factor Cholesky L clustering property depends on the choice of the filling pattern S. It is confirmed that matrix L must be a lower-triangle matrix and satisfy the next conditions:

$$A = LL^T + R, \quad l_{ij} = 0 \text{ if } (i,j) \notin S \text{ and } r_{ij} = 0 \text{ if } (i,j) \in S \tag{4}$$

There are different strategies of forming filling pattern S. Overall strategies proposed are based on two following ideas:

- Pattern S is initially fixed;
- Pattern S is formed in the process of forming L.

The first strategy, where S is initially fixed, is rather attractive due to the following reasons:

- Pattern S of matrix L is easily obtained from pattern of matrix A;
- The amount of memory needed for matrix L is predictable;
- There is no need to control drop-tolerance.

There are many variants of pattern S initial fixation. For example, one could define a pattern S so that matrix L has become a band fixed-width matrix. Another way is to get pattern S and pattern matrix A identical. The most promising variant was first used for so-called $ILU(p)$ factorization of asymmetric matrix A. In this case for parameter $p = 0$ the pattern S is set on the pattern of

matrix A. If $p > 0$, additional filling of L and U columns is allowed. Actually, an additional number of non-zero elements placed in L and U during their creation are limited by parameter p.

The disadvantage of initial fixation of pattern S consists in uncontrollability losses of non-zero elements of L and U. These losses in the formation of preconditioners often make it impossible to cluster the proper values of matrix A.

The other pattern S formation strategy in the process of matrix A factorization is so-called τ-drop-tolerance strategy. In this case on a v- step formation L and U correspondent elements are considered to be unimportant and they are not stored in memory (as zero elements) if they satisfy the condition $|a_{ij}^{(v)}| \leq \tau\sqrt{a_{ii}^{(v)}a_{jj}^{(v)}}$. The τ parameter defines the limit of losses.

The τ-drop-tolerance strategy has two disadvantages:

- Unpredictability of memory requirements on factor matrices storage, which depends on the value of a chosen parameterτ;
- Elasticity of parameter τ vagueness influence on the ability of factor matrices to cluster the proper values of matrix A.

Different combinations of pattern S formation strategies described earlier are used at present. For example, a modified p-strategy for symmetric matrices. The modification is based on forming the filling pattern S independently from the position n_j of non-zero elements in j column of matrix A. Only $n_j + p$ elements of j column of L are stored. There is approach on the base of combining p- and τ- strategies. The idea of this two-parameter strategy is a gradual application of τ-drop-tolerance strategy to the elements of j column of matrix L formed on j step. According to the p-strategy the $n_j + p$ elements with the biggest absolute value are stored in memory. There are some other approaches.

There is a considerable disadvantage in traditional strategies for forming pattern S. These strategies generate considerably irregular distribution of errors, which appears during matrix factorization process in fixed amount of memory. As Cholesky factorization procedure is a gradual procedure and can be performed either by rows or by columns, only errors initially inputted in matrix L in the first few rows or columns correspondingly can be controlled. Once initially set during the first step of factorization, such errors can quickly spread on the rest of matrix L during the next steps of factorization procedure. This process involves not only multiplication of errors but also their accumulation. Besides the assumption that the formed filling pattern S don't satisfy the desirable conditions (4) is a mistake. In general case, the pattern S does not satisfy the last condition, as $r_{ij} \neq 0$ if $(i, j) \in S$.

As an example, let us show this on *jki* version of *LLT* Cholesky factorization implemented so that the elements of lower-triangle part of matrix L are computed at the positions of elements of input matrix A. Note, that for symmetric

input matrix A the equation

$$A = LL^T = L^{(0)} = \begin{bmatrix} l_{11}l_{11} & l_{21}l_{11} & l_{31}l_{11} & \dots \\ l_{21}l_{11} & l_{21}l_{21} + l_{22}l_{22} & l_{31}l_{21} + l_{32}l_{22} & \dots \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}l_{31} + l_{32}l_{32} + l_{33}l_{33} & \dots \\ l_{41}l_{11} & l_{41}l_{21} + l_{42}l_{22} & l_{41}l_{31} + l_{42}l_{32} + l_{43}l_{33} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{5}$$

is solved gradually by columns respectively to elements l_{ij} of matrix L. During solving procedure of the equation (5) matrix A is being gradually transformed in Cholesky factor L. As a result the matrix $L^{(0)} = LL^T$ can be presented as following:

$$L^{(1)} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & \dots \\ l_{21} & l_{22}l_{22} & 0 & 0 & \dots \\ l_{31} & l_{31}l_{21} + l_{32}l_{22} & l_{32}l_{32} + l_{33}l_{33} & 0 & \dots \\ l_{41} & l_{41}l_{21} + l_{42}l_{22} & l_{41}l_{31} + l_{42}l_{32} + l_{43}l_{33} & l_{42}l_{42} + l_{43}l_{43} + l_{44}l_{44} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{6}$$

$$L^{(2)} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & \dots \\ l_{21} & l_{22} & 0 & 0 & \dots \\ l_{31} & l_{32} & l_{33}l_{33} & 0 & \dots \\ l_{41} & l_{42} & l_{41}l_{31} + l_{42}l_{32} + l_{43}l_{33} & l_{43}l_{43} + l_{44}l_{44} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{7}$$

and so on till $L^{(n)} = L$.

Assume that in the result of the first step factorization from matrix $L^{(0)}$ we have obtained matrix $L^{(1)}$, with insignificantly small element, for example, l_{21} which should be dropped according to the chosen pattern S formation strategy. Therefore the transformation from $L^{(0)}$ to $L^{(1)}$ introduces a contortion to the diagonal element l_{22} of matrix $L^{(1)}$. During the next step from $L^{(1)}$ to $L^{(2)}$ the small error l_{21} will be multiplied with the lower elements of the first column of matrix $L^{(1)}$ and spread towards the whole second column of matrix $L^{(2)}$ in the form of unconsidered components. During the next steps of factorization the contorted elements of the second column generate deviation of matrix elements at the right lower position from the element l_{21}. Moreover the further operations on the contorted elements result in unpredictable accumulation of errors in elements of final matrix $L^{(n)} = L$. Such errors are non-uniformly spread, so that their lowest level is observed in the first columns of matrix L, and the highest one is in the last columns.

Significant level of accumulated errors is often a reason of the formation of such a matrix L, that is being substituted in the first equation of expression (4), could not guarantee the admissible level of errors. So, the matrix L obtained does

not inherit properties of matrix A and so could not be a good preconditioner for conjugate gradients method.

Authors in paper [1] propose an interesting variant of *ILLT(p)* Cholesky factorization, based on a modified p-drop-tolerance strategy. This variant of factorization is worth a particular attention, as it partially blocks error distribution in matrix being formed. The *ILLT(p)* factorization differs from other algorithms as it contains two loops in j-step factorization instead of traditional one. In the first loop j column of forming matrix $L^{(j)}$ are calculated with respect to the elements l_{ij} for $i > j$ and a necessary correction of diagonal elements l_{ii} for $i > j$ is done in the second loop. A drop-tolerance strategy is implemented between two loops so that formed in the first loop insignificant elements do not alter the diagonal elements l_{ii}. Therefore the contortions admissible while dropping unimportant elements do not influence the corresponding diagonal elements and the result errors $R = A - LL^T$ become smaller. In this algorithm an indirect error distribution is possible only through non-diagonal elements. This helps to form better preconditioners for conjugate gradient method.

The *ILLT(p)* factorization has three essential disadvantages:

- Incomplete usage of limited amount of memory (limit on memory usage is set by p parameter) for such a j- column of matrix $L^{(j)}$, which has a filling less than $n_j + p$, where n_j – is a number of non-zero elements in j- column of matrix A.
- As parameter p can not take negative values, the formed preconditioner L can not be placed in less amount of memory than is needed to store matrix A;
- Incomplete losses of elements in different columns of matrix L, does not permit the exhaustive usage of the memory resources provided for high-quality computation of incomplete Cholesky factorization.

We propose an *ILLT(m,p)* Cholesky factorization with adaptive τ-drop-tolerance strategy, without disadvantages described above. The errors $R = A - LL^T$ are proved to be less than in case of *ILLT(p)* factorization.

The *ILLT(m,p)* factorization allows to form matrix L within an amount of memory provided without positioning of the initial matrix A. The memory size provided to matrix L is chosen independently from the memory size occupied by matrix A and is set by parameter m, which fixes the relation of memory sizes occupied by both matrices. Consider a lower-triangle part of symmetric matrix A contains nnz non-zero elements. Then the parameter m can assume any values so that $m \geq \frac{n}{nnz}$. Obviously in a partial case if $m = \frac{n}{nnz}$, the matrix L is diagonal matrix with n elements.

The fact that an input matrix A and preconditioner L are stored in memory simultaneously is not a desirable condition to implement an iterative solution of problem (1) by conjugate gradient method. That is why the refusal to implement factorization in place of positioning of matrix A is not only justified, but it also permits even more effective usage of memory resources, provided.

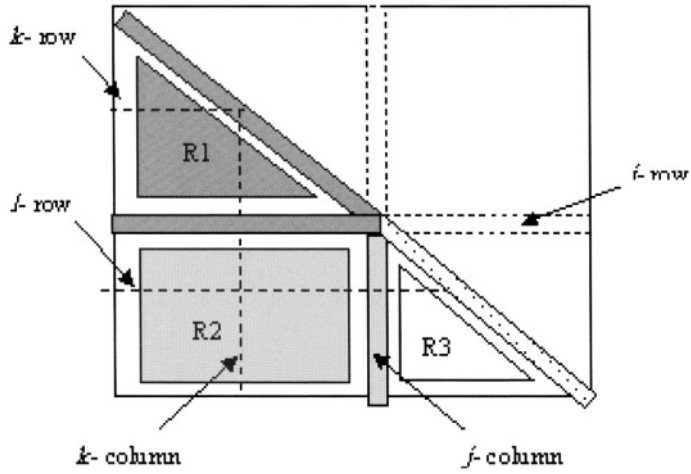


Fig. 1. Matrix $L^{(j)}$ after j steps of factorization.

The structural analysis of elements of intermediate matrices $L^{(j)}$ (5) – (7) formed by *LLT* Cholesky factorization confirms the existence of such memory resources.

Without taking into account the positioning of zero elements in matrix $L^{(j)}$, we distinguish three groups of elements *R*1, *R*2 and *R*3, forming the lower-triangle partition of matrix, and lower and upper diagonal elements. Diagonal elements marked in dark-grey, elements of *R*1 group and elements of row j, contain result elements of matrix L, which do not change and are not used during the further process of formation of matrices $L^{(j+1)}$, $L^{(j+2)}$, ... $L^{(n)} = L$. Elements of R2 group marked in light grey and elements of j column are also the result elements, which do not change but they are still needed to form the elements of *R*3 group and lower diagonal elements. Ineffectiveness of memory usage is estimated by measuring the size of *R*3 group, where corresponding elements of *A* placed initially are stored. It is obvious that the size of *R*3 group depends on the step j of factorization process. During the first steps the size of *R*3 group is comparable to the size of the whole lower-triangle partition of matrix *A*, but it reduces linearly to zero during the next steps.

As *R*3 group contains of elements of separately stored matrix *A*, there is no need to store the same elements twice. The released amount of memory can be used to store a bigger number of elements of matrices $L^{(0)}$, $L^{(1)}$, ... $L^{(n)} = L$ and therefore compute an incomplete Cholesky factorization with lower level of losses. We will denote the stored matrices that do not contain *R*3 group as $\mathbf{L^{(0)}}$, $\mathbf{L^{(1)}}$, ... $\mathbf{L^{(n)}}=L$.

Another special feature of proposed *ILLT(m,p)* Cholesky factorization is the usage of an adaptive τ-drop-tolerance strategy. This drop-tolerance strategy allows to coordinate memory requirements for matrices $\mathbf{L^{(0)}}$, $\mathbf{L^{(1)}}$, ... $\mathbf{L^{(n)}}=L$ formed gradually with the possibility to place the elements of these matrices

Table 1. Characteristics of the test matrices

Matrix	n	nnz(A)	λ_{min}	λ_{max}	cond(A)
bcsstk08	1074	7017	2.9e+03	7.6e+10	4.7e+07
bcsstk09	1083	9760	7.1e+03	6.7e+10	3.1e+04
bcsstk10	1086	11578	8.5e+01	4.4e+07	1.3e+06
bcsstk11	1473	17857	2.9e+00	6.5e+08	5.3e+08
bcsstk13	2003	42943	–	–	4.6e+10
bcsstk17	10974	219812	–	–	–
bcsstk18	11948	80519	1.2e-01	4.3e+10	–
bcsstk19	817	3835	1.4e+03	1.9e+14	2.8e+11
bcsstk23	3134	24156	–	–	6.9e+12
bcsstk24	3562	81736	–	–	–
1138bus	1138	2596	3.5e-03	3.0e+04	1.0e+02
nasasrb	54870	1366097	–	–	–

Table 2. The memory used by factorization algorithms: $nnz(L)$ / $nnz(A)$

Matrix	*ILLT(p)*			*ILLT(m,p)* in case of $p = n$			
	$p = 0$	$p = 2$	$p = 5$	$m = 0.75$	$m = 1.00$	$m = 1.25$	$m = 1.50$
bcsstk08	1.000	1.291	1.735	0.733	0.9631	1.1820	1.351
bcsstk09	1.000	1.219	1.550	0.750	0.9487	1.2365	1.500
bcsstk10	1.000	1.185	1.461	0.748	0.9981	1.2489	1.499
bcsstk11	1.000	1.157	1.391	0.741	0.9906	1.2385	1.494
bcsstk13	1.000	1.092	1.230	0.743	0.9907	1.2415	1.496
bcsstk17	1.000	1.094	1.233	0.749	0.9988	1.2483	1.498
bcsstk18	1.000	1.228	1.565	0.750	0.9998	1.2498	1.499
bcsstk19	1.000	1.408	2.019	0.750	0.9987	1.2446	1.498
bcsstk23	1.000	1.253	1.638	0.749	0.9994	1.2494	1.401
bcsstk24	1.000	1.087	1.217	0.731	0.9381	1.2091	1.397
1138bus	1.000	1.508	2.190	0.750	0.9992	1.2500	1.500

in memory actually provided. Normally, during the first steps of factorization memory requirements for formed $\mathbf{L^{(0)}}$, $\mathbf{L^{(1)}}$, ... matrices storage do not exceed a provided memory limit. Consequently, at the beginning any element losses may be admissible and we may actually start with *LLT* Cholesky factorization. Factorization with $\tau = 0$ continues until the deficit of memory for distributed of following $\mathbf{L^{(j)}}$ matrix is arisen. Then if we increase τ and set $l_{ik} = 0$ for the such elements l_{ik} that

$$\{|l_{ik}| \leq \tau l_{kk}; i \in (k+1, n), k \in (1, j)\} \tag{8}$$

we can decrease $\mathbf{L^{(j)}}$ matrix's filling to an appropriate level and provide an opportunity to place the rest of non-zero elements in provided memory. It is obvious that τ should be increased gradually and filling of matrix $\mathbf{L^{(j)}}$ with non-zero elements should be strictly observed. A new value of τ parameter should be selected so that we can find an admissible filling of matrix $\mathbf{L^{(j)}}$ as soon as possible and, from the other side, do not permit redundant losses and unjustified

Table 3. The number of iterations for the conjugate gradients method

Matrix	*ILLT(p)*			*ILLT(m,p)* in case of $p = n$			
	$p = 0$	$p = 2$	$p = 5$	$m = 0.75$	$m = 1.00$	$m = 1.25$	$m = 1.50$
bcsstk08	14	13	9	13	10	10	9
bcsstk09	24	14	17	33	27	17	10
bcsstk10	35	20	14	43	22	11	4
bcsstk11	704	689	673	624	337	228	137
bcsstk13	112	93	79	61	48	46	44
bcsstk17	309	229	222	138	78	57	49
bcsstk18	555	235	159	148	56	31	23
bcsstk19	465	27	22	406	26	20	18
bcsstk23	1291	1145	1093	1116	1079	1068	121
bcsstk24	1544	1169	823	915	701	564	495
1138bus	97	43	23	179	94	65	46

Table 4. The number of iterations for the conjugate gradients method

Matrix	*ILLT(m,p)* in case of $m = 0.75$		
	$p = 0$	$p = 5$	$p = 50$
bcsstk08	156	12	11
bcsstk09	33	35	33
bcsstk10	52	41	43
bcsstk11	818	707	624
bcsstk13	114	83	63
bcsstk17	313	235	136
bcsstk18	560	168	148
bcsstk19	465	405	403
bcsstk23	1291	1123	1121
bcsstk24	1551	916	909
1138bus	156	184	180
nasasrb*	2234	2046	1986

*– *ILLT(m,p)* factorization was done for $m = 0.5$

big errors in matrices $\mathbf{L^{(j)}}$, $\mathbf{L^{(j+1)}}$, ... $\mathbf{L^{(n)}}$=L. All these conflicting requirements satisfy the following correspondence for τ

$$\Delta\tau = max(\tau_s, \tau\frac{n-j}{n}) \tag{9}$$

Here the initial value of incremental growth $\Delta\tau = \tau_s \neq 0$ is set by expert way. Usually it is sufficiently small and does not lead to any significant losses. Further, the relative growth rate of τ parameter is set to be equal to $\frac{\Delta\tau}{\tau} = \frac{n-j}{n}$ for during the first steps of factorization for $j \ll n$ and the low value of τ closed to τ_s we can set admissible level of losses quickly but not accurately. During the

last steps of factorization when $j \to n$ and the value of τ is high one must not allow unjustified losses.

Two blocking strategies, current and retrospective, resist against further distribution and accumulation of errors occurring as a result of using the adaptive τ-losses strategy. We use the blocking strategies to decrease negative influence of dropped non-zero elements of $R2$ group (Figure 1), as they directly influence the elements of the lower diagonal group and indirectly influence the elements of adjacent columns from the left of $R2$ group.

The current blocking strategy is needed on the current j step of factorization with regard to the elements of j column and drop-tolerance criteria $\{|l_{ij}| \leq \frac{\tau}{2} l_{jj}; i \in (j+1, n)\}$ can be used, which differs from criteria (8), as τ is twice less. The correction of diagonal elements l_{ii} on the loss value l_{ij} is not allowed according to the *ILLT(p)* algorithm and the explanations in paper [1]. Note, that the drop-tolerance strategy with regard to elements of j row is constant.

The retrospective blocking strategy is used after the next incrementation of τ parameter. This strategy supposed a partial recovering value of the elements of lower-diagonal group $\{l_{kk}; k \in (j+1, n)\}$. The square values of l_{ki} elements, which now satisfy the condition $\{l_{ki} \leq \tau; k \in (j+1, n); i \in (1, j)\}$, were subtracted from $\{l_{kk}; k \in (j+1, n)\}$ earlier. The recovering of corresponding values of l_{kk} involves the addition of value l_{ki}^2.

In the *ILLT(m,p)* algorithm τ and p drop-tolerance strategies are used concurrently. The p-strategy is additional and is used in special cases to influence the amount of calculations for *ILLT(m,p)* factorization. If parameter p is small, then p-drop-tolerance strategy predominates and *ILLT(m,p)* algorithm becomes close to *ILLT(p)* algorithm in respect of its possibilities. If parameter p satisfies condition $p+n_j = n-j$, p-drop-tolerance strategy does nothing. Such an unusual application of p-strategy could be ineffective in combination with equation- and variable- sorting algorithms, which are not considered.

3 Computational Experiments

In our experiments we have researched the particularities of the suggested *ILLT(m,p)* algorithm as a function of two parameters: m - memory used and p - computation accelerator.

All experiments were conducted on the samples from "Harwell-Boeing" collection of sparse matrices. Matrices bcsstk08 - bcsstk11, bcsstk18, bcsstk19 and 1138bus were used in [1] and therefore we have chosen them for results comparison. Besides, we have expanded the list of matrices being tested in order to achieve more complete estimation of features of the algorithm suggested. In the columns n and nnz(A) of Table 1 the dimensions of test matrices and the number of non-zero elements in their lower-triangle part are provided. The minimal and the maximal values of eigenvalues located in columns λ_{min} and λ_{max} correspondingly are taken from [1]. The condition numbers of the matrices given in the column cond(A) are taken from the web-pages of Harwell-Boeing collection.

The preconditioning conjugate gradients method with *ILLT(m,p)* algorithm was used for solving the system of form $AX + B = 0$ with matrices A taken from the test examples and with the unity vector B. Furthermore the iterative procedures begun with the zero-vector X and ended when $\|AX + B\| \leq \delta\|B\|$. In all test examples the value $\delta = 10^{-3}$ was chosen. The results represented in the tables 2-3 demonstrate the advantages of the suggested incomplete Cholesky factorization *ILLT(m,p)* algorithm. Here one can observe the unpredictability of the memory consumed in fact by the *ILLT(p)* algorithm at different values of p. On the contrary, the memory demand of the *ILLT(m,p)* algorithm correlates good with the memory actually consumed, moreover the memory needed can be substantially less than the memory occupied by matrices A, which is not possible to achieve in case of using *ILLT(p)* algorithm.

The convergence of the iterative procedures in the conjugate gradients method with the preconditioner L, that are formed by *ILLT(m,p)* algorithm is substantially higher than those preconditioner L, that are formed by the *ILLT(p)* algorithm. Such a difference is especially noticeable for badly conditioned matrices, which one can see from the results of bcsstk11, bcsstk13, bcsstk17, bcsstk18, bcsstk23, bcsstk24 tests.

Thus the suggested incomplete Cholesky factorization *ILLT(m,p)* algorithm provides high efficiency of available memory usage. It's clear that achievement of such efficiency requires increasing the computation costs, that is increasing the time needed for predictors forming. In time-critical cases the value of the p can be decreased, such a decreasing makes it possible to achieve the required intensity in forming predictors L in a memory area of fixed size. However, it must be noted that the predictors obtained in such fast way possess worse clustering of eigenvalues and therefore decrease the convergence speed of the iterative procedures. That can be seen from the experimental results, provided by the table 4.

Here we don't consider any ordering algorithm as a radical means to accelerate process of forming of the preconditioners. Usage of an ordering algorithm in incomplete Cholesky factorization *ILLT(m,p)* algorithm is more desirable.

In conclusion note the flexibility of the suggested incomplete Cholesky factorization *ILLT(m,p)* algorithm. The control parameters m and p as much as possible permit take into account the limitations of the computer resources on the solution of large positive definite systems of equations and for the solution of large-scale trust region sub problems.

References

1. C.-J. Lin, J. J. Morë. *Incomplete Cholesky factorizations with limited memory,* SIAM Journal on Sci. Comput., No.1, 1999, pp. 24 – 45.

A Multigrid Poisson Solver on General 3-Dimensional Domains

Marjan Šterk and Roman Trobec

Jožef Stefan Institute, Ljubljana, Slovenia,
marjan.sterk@ijs.si

Abstract. In this paper we present our practical experience with solving the Poisson equation on arbitrary 3-dimensional domains using finite difference approximation and Neumann boundary conditions. The equation is presented and arguments for the choice of numerical methods are given. Discretization is described and the resulting system of linear equations is analysed. Our practical implementation of the multigrid method for the presented problem on general domains is described. Results of convergence tests are given and analysed for multigrid and other, simpler methods.

1 Introduction

The Poisson equation

$$\nabla^2 u(\mathbf{x}) = f(\mathbf{x}) \tag{1}$$

is an elliptic time-independent partial differential equation (PDE) that appears in many computations, notably in heat conduction and incompressible fluid flow simulations. The latter requires coupling the Navier-Stokes equation with the continuity equation, resulting in the need to solve the Poisson equation for pressure correction at each time-step [1], which becomes the most computationally intensive part of the simulation.

For internal flow problems Neumann boundary conditions are prescribed for the pressure correction [2], i.e. the normal derivative

$$\frac{\partial u}{\partial n} = 0 \qquad \text{on all boundaries.} \tag{2}$$

There are infinitely many solutions u that satisfy (1) and (2). These solutions differ only in an additive constant. Because the absolute values of pressure are not important in this kind of problems, any particular solution has to be found.

An important field where fluid dynamics simulations are used is medicine, such as blood flow in vessels and water circulation in open heart surgeries where water is used to cool the heart muscle [3]. Simulations in 3 dimensions and irregular domains are required. The models of the body organs are usually created using bitmaps from the Visual Human Dataset or similar sources, which

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1052–1058, 2004.
© Springer-Verlag Berlin Heidelberg 2004

produces 3-dimensional bitmap models [4]. It is thus natural to discretize the problem using finite differences.

The discretization with finite differences results in a sparse system of linear equations, whose sparseness pattern does not lend itself to the use of direct solvers. Iterative methods are thus needed to efficiently solve the system, such as the well-known Gauss-Seidel (GS) and SOR methods or the more sophisticated conjugate gradient (CG) method [5].

The weakness of GS is that although it reduces the high-frequency components of the error vector quickly, i.e. it smoothes the solution, many iterations are needed to reduce the low-frequency components. It is therefore beneficial to do parts of the calculation on coarser grids, where these components become high-frequency and are thus easily reduced. The solution is then interpolated to the original fine grid, where a few additional iterations are performed to obtain the final solution [6]. This idea is the basis of multigrid methods, which are generally regarded as best suited to the problems of this sort.

In the following section the discretization of the domain and the Poisson equation is described. The resulting system of linear equations is analysed. Section 3 focuses on the details of the multigrid solver for this particular problem, i.e. the interpolation and restriction operators. In Section 4, convergence rates of multigrid and other, simpler methods are given and analysed on a regular cubic domain as well as on an irregular domain.

2 Discretization

The domain is discretized to cubes of size $h \times h \times h$. Each internal cube can be either liquid or solid, while all boundary cubes are solid to form a closed cavity. The pressure correction u is defined in the centres of liquid cubes.

The second-order accurate central difference second derivative approximation is used to discretize (1) for a cube with 6 liquid neighbours to:

$$\frac{u_{x-1,y,z} - 2u_{x,y,z} + u_{x+1,y,z}}{h^2} + \frac{u_{x,y-1,z} - 2u_{x,y,z} + u_{x,y+1,z}}{h^2} + \\ + \frac{u_{x,y,z-1} - 2u_{x,y,z} + u_{x,y,z+1}}{h^2} = f_{x,y,z}, \qquad (3)$$

where $u_{x,y,z}$ stands for $u(xh, yh, zh)$. The discrete Neumann boundary conditions (2) state that $\frac{\partial u}{\partial n} = 0$ on the faces of all solid cubes. If e.g. the cube centred at $(x, y, z-1)$ is solid then the boundary condition is $\frac{\partial u}{\partial z} = 0$ on its upper face, i.e. at $(x, y, z-0.5)$. Using the central difference approximation we obtain

$$\left.\frac{\partial u}{\partial z}\right|_{x,y,z-0.5} = \frac{u_{x,y,z} - u_{x,y,z-1}}{h} = 0 \;\Rightarrow\; u_{x,y,z} = u_{x,y,z-1}. \qquad (4)$$

The latter form allows us to remove both $u_{x,y,z}$ and $u_{x,y,z-1}$ from (3) in this case so that no values outside the domain appear in the solution. In general,

numbering all the liquid cubes in the domain gives the linear system $A\mathbf{u} = \mathbf{b}$, where $\mathbf{u} = \{u_{x,y,z}\}$, $\mathbf{b} = \{h^2 f_{x,y,z}\}$ and

$$A = [a_{i,j}]_{N\times N} \;;\quad a_{i,j} = \begin{cases} 6-s & i=j, \text{the cube } i \text{ has } s \text{ solid neighbours} \\ -1 & i\neq j, \text{cubes } i \text{ and } j \text{ are neighbours} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

N stands for the total number of liquid cubes in the domain. The discretization for the case of different resolutions for the dimensions x, y and z can be derived in a similar manner.

The matrix A is symmetric. Its rank is $N - 1$ because the sum of rows is 0. Also $\sum b_i = 0$ so, as expected, the system has infinitely many solutions, any of which is to be found. Iterative methods based on the Gauss-Seidel iteration, such as GS itself, SOR and multigrid, can be used on this singular system, which follows from the eigenvectors of the Gauss-Seidel iteration matrix

$$G = M^{-1}N = (D + L)^{-1}(-U), \tag{6}$$

where D is the diagonal of A, and U and L the strict upper and lower triangular portions of A, respectively. The convergence criterion requests that all the eigenvalues of G have moduli less than 1 [5]. In our case, all the eigenvalues' moduli are less than 1 except for a single eigenvalue 1 with the corresponding eigenvector $\mathbf{v}_1 = (1, 1, \ldots, 1)^T$. The error will thus be reduced through the iterations in all directions except for $\mathbf{v}_1$. Since our goal is to find any of the solutions differing in an additive constant, i.e. in a multiple of $\mathbf{v}_1$, changes of the approximate solution in the direction of $\mathbf{v}_1$ are irrelevant.

3 Multigrid Implementation

We implemented the full multigrid method [7,8]. First, the system is restricted to the coarsest grid. The resulting small system is solved directly. The solution is then interpolated to twice-finer grid, where the smoother (a few iterations of GS method) is applied. Further transitions between grid levels are shown in Figure 1. Theoretically this method achieves constant number of iterations, independent of the system size.

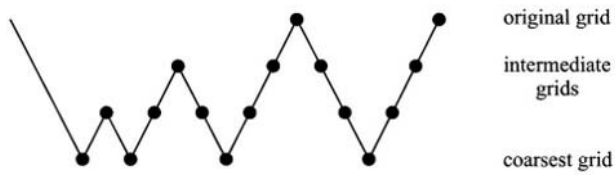


Fig. 1. The full multigrid method consists of restriction (descending lines), interpolation (ascending lines) and applying the smoother (dots).

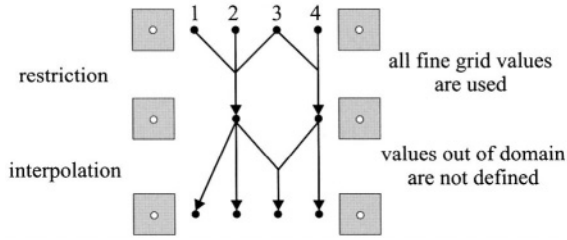


Fig. 2. The restriction and interpolation operators.

The restriction and interpolation operators have to be chosen carefully in order not to introduce a larger error into the solution than can be reduced by the subsequent application of the smoother. We used operators that are shown in Figure 2. Restriction uses a weighted average of all fine grid values to obtain coarse grid values. Note that on an irregular domain the system matrix A is derived implicitly from the domain shape, i.e. the solid-liquid pattern of the cubes. This pattern is restricted using the same restriction operator, which gives the domain shape on the coarser grid. The system matrix on the coarser grid is again derived implicitly from the shape.

In the inner parts of the domain the interpolation is taken as the transpose of the restriction operator. Fine grid values next to a boundary must be obtained without using values outside the domain, which are not prescribed by Neumann boundary conditions. Fine grid values next to a boundary are thus equal to those $1.5h$ away from the boundary, which most closely follows the boundary conditions (see the lower left arrow in Figure 2).

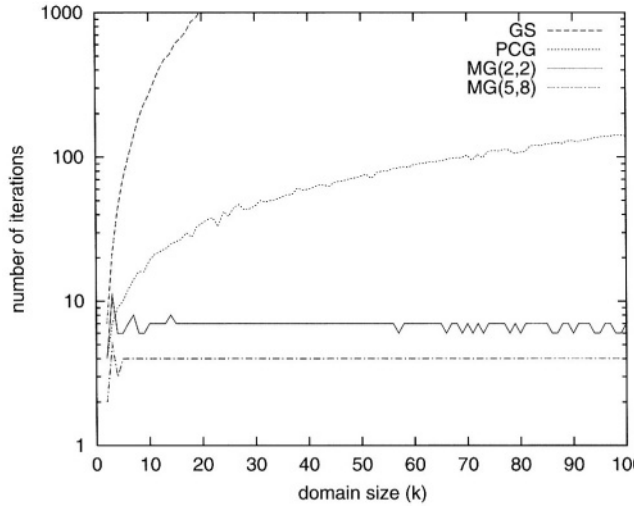


Fig. 3. Comparison of methods on a regular cubic domain.

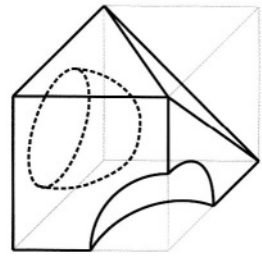


Fig. 4. The test irregular domain.

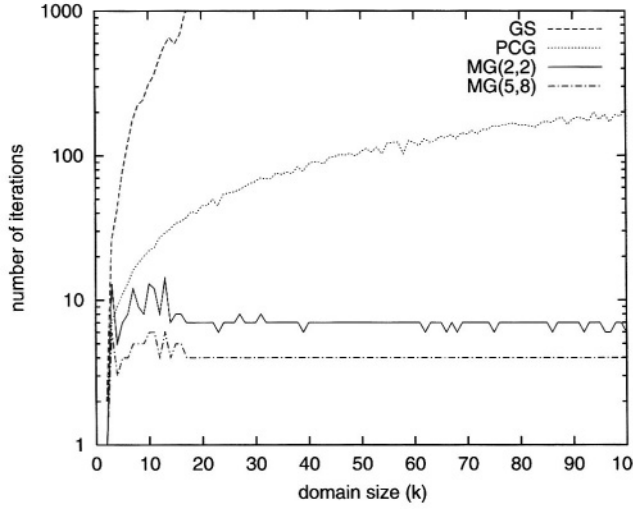


Fig. 5. Comparison of methods on the irregular domain.

4 Results

The solver was first implemented in Matlab for regular cubic domains in order to analyse the linear system, test various methods, and verify the results by comparing with those obtained by the built-in solver. The following methods were tested: Gauss-Seidel (GS), conjugate gradient with incomplete Cholesky preconditioning (PCG), which reduces the number of iterations for the CG method approximately by a factor of 3, and full multigrid (MG). High-performance general-domain versions of the methods were then implemented in C++ as a part of a fluid-flow simulation project [9].

Figure 3 shows the number of iterations needed to solve the Poisson equation on a regular cubic domain consisting of $k \times k \times k$ points. $\mathrm{MG}(i,j)$ stands for full multigrid method with i Gauss-Seidel iterations at each grid level and j or more grid points at the coarsest level. The iteration stopping criterium was $||\mathbf{r}||_\infty \leq 10^{-6}$.

The Gauss-Seidel and PCG methods require approximately $1,5k^{2,1}$ and $1,4k$ iterations, respectively. Both are outperformed by the full MG method, where the number of iterations is independent of k. MG(5,8) solves the system in about 20 % less time than MG(2,2).

The methods were also tested on an irregular domain consisting of a trimmed cube hollowed out by two spheres, which is shown in Figure 4.

The narrow bands, e.g. in the far right corner, are potential trouble areas for multigrid because their shape will inevitably be lost on coarse grids. However, Figure 5 shows that the number of iterations of the full MG method on the irregular domain remains the same as for a regular domain. The number of iterations for GS and PCG methods increases.

5 Conclusions

In this work a multigrid solver for the Poisson equation with Neumann boundary conditions is described. It uses finite difference approximation and works on arbitrary 3-dimensional domains. It was developed together with Gauss-Seidel and conjugate gradient solvers as a part of a fluid flow simulation project. The performance of all solvers is compared. The results show that the multigrid outperforms other methods because the number of iterations is independent of the domain size, provided that the restriction and interpolation operators are implemented correctly. It is also shown that irregularity of the domain is not a significant problem even though the shape cannot be adequately represented on the coarser grids.

Directions for future work include improving the performance of the presented solver by using an optimised numerical library, e.g. Blitz++ [10]. A further improvement of the fluid-flow simulation would be the use of finite element method, which would presumably result in a smaller but less regular Poisson system matrix.

References

1. C. W. Hirt and J. L. Cook. Calculating three-dimensional flows around structures. *J. Comput. Phys.,* 10:324–340, 1972.
2. C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics.* Springer Verlag, 1988.
3. R. Trobec, B. Slivnik, B. Geršak, and T. Gabrijelčič. Computer simulation and spatial modelling in heart surgery. *Computers in Biology and Medicine,* 4:393–403, 1998.
4. R. Trobec, G. Pipan, P. Trunk, and J. Močnik. Spatial heart model derived from VHD. In *Bioimages for Europe '99, 2nd International Workshop of the Visible Human Dataset,* Milan, 1999.
5. M.T. Heath. *Scientific Computing: An Introductory Survey, 2nd Ed.* WCB/ McGraw-Hill, 2002.
6. A. Brandt. Multi-level adaptive solutions to boundary value problems. *Math. Comput.,* 31:333–390, 1977.

7. G. Golub and J. M. Ortega. *Scientific Computing - An Introduction with Parallel Computing.* Academic Press Inc., Boston, 1993.
8. P. Wesseling. *An Introduction to Multigrid Methods.* John Wiley and Sons, 1991.
9. M. Šterk, R. Trobec, and M. Praprotnik. Comparison of incompressible fluid flow simulation methods. In *Parallel Numerics '02, Theory and Applications.* Jožef Stefan Institute and University of Salzburg, 2002.
10. T. Veldhuizen. Blitz++ user's guide, 2001.

Solving Linear Recurrence Systems Using Level 2 and 3 BLAS Routines

Przemysław Stpiczyński

Department of Computer Science, Marie Curie–Skłodowska University
Pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland
przem@hektor.umcs.lublin.pl

Abstract. The aim of this paper is to present a new efficient BLAS-based algorithm for solving linear recurrence systems with constant coefficients, which can be easily and efficiently implemented on shared or distributed memory machines and clusters of workstations. The algorithm is based on level 3 and level 2 BLAS routines _GEMM, _GEMV and _TRMV, which are crucial for its efficiency even when the order of a system is relatively high. The results of experiments performed on a dual-processor Pentium III computer are also presented and discussed.

1 Introduction

The critical part of several numerical algorithms [4,8,10,12,13] reduces to the solution of a linear recurrence system of order m for n equations with constant coefficients:

$$x_k = \begin{cases} 0 & \text{for } k \leq 0 \\ f_k + \sum_{j=1}^{m} a_j x_{k-j} & \text{for } 1 \leq k \leq n. \end{cases} \tag{1}$$

Various algorithms for the solution of the problem (1) have been designed for parallel and vector computers (see [4,7,9,13,15] for more references). However these algorithms like *cyclic reduction, Wang's method* and *recursive doubling* lead to a substantial increase in the number of floating-point operations, what makes them unattractive in classical serial systems (just like Intel Pentium) or parallel computers with a limited number of processors [5]. On the other hand, it is well known that reducing costs of memory access is essential for achieving good performance of numerical software [6]. Usually, codes based on level 2 and 3 BLAS (Basic Linear Algebra Subprograms [5]) routines achieve good performance because they allow to reuse data stored in cache memory [3].

In our earlier work [14] we have presented a new algorithm for solving (1) based on the BLAS operations _AXPY and _GEMM. Unfortunately, the efficiency of this algorithm decreases when the order of the solved system (the value of m) increases. The aim of this paper is to present a new high performance BLAS-based algorithm for solving linear recurrence systems with constant coefficients based on level 3 and level 2 BLAS routines _GEMM, _GEMV and _TRMV, which are crucial for its efficiency even when the order of a system is relatively high. The

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1059–1066, 2004.
© Springer-Verlag Berlin Heidelberg 2004

results of experiments performed on a dual processor Intel Pentium running under Linux with Intel Fortran 95 Compiler and the Math Kernel Library (the optimized version of BLAS which supports the streaming SIMD extensions and some other features of Intel processors [1,2]) show that the sequential version of the algorithm is up to four times faster than a simple algorithm based on (1). Moreover, the algorithm can be easily and efficiently parallelized for shared or distributed memory machines and clusters of workstations.

2 *Divide and Conquer* Approach

In this section we briefly discuss the idea of the *divide-and-conquer* approach for solving (1). More details can be found in [9,14]. Let us assume that $n \gg m$, i.e. the order of a recurrence system is rather small, and choose positive integers r and s such that $rs \leq n$ and $s > m$. It is clear that the numbers $x_1, \ldots, x_{rs}$ satisfy the following block system of linear equations

$$\begin{pmatrix} L & & & \\ U & L & & \\ & \ddots & \ddots & \\ & & U & L \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_r \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_r \end{pmatrix}, \tag{2}$$

where for $j = 1, \ldots, r$

$$\mathbf{x}_j = (x_{(j-1)s+1}, \ldots, x_{js})^T, \ \mathbf{f}_j = (f_{(j-1)s+1}, \ldots, f_{js})^T \in \mathbb{R}^s \tag{3}$$

and

$$L = \begin{pmatrix} 1 & & & & \\ -a_1 & \ddots & & & \\ \vdots & \ddots & \ddots & & \\ -a_m & & \ddots & \ddots & \\ & \ddots & & \ddots & \ddots \\ & & -a_m \cdots -a_1 & 1 \end{pmatrix}, \ U = \begin{pmatrix} -a_m \cdots -a_1 & & \\ & \ddots & \vdots \\ & & -a_m \\ 0 & & \end{pmatrix} \in \mathbb{R}^{s \times s}. \tag{4}$$

Obviously, the system (2) reduces to the form

$$\begin{cases} \mathbf{x}_1 = L^{-1}\mathbf{f}_1 \\ \mathbf{x}_j = L^{-1}\mathbf{f}_j - L^{-1}U\mathbf{x}_{j-1} \text{ for } j = 2, \ldots, r. \end{cases} \tag{5}$$

When we consider the special structure of the matrix U, we can rewrite (5) as

$$\begin{cases} \mathbf{x}_1 = L^{-1}\mathbf{f}_1 \\ \mathbf{x}_j = L^{-1}\mathbf{f}_j + \sum_{k=1}^{m} \alpha_j^k \mathbf{y}_k \text{ for } j = 2, \ldots, r \end{cases} \tag{6}$$

where $\mathbf{e}_k$ denotes k-th unit vector of $\mathbb{R}^s$, $L\mathbf{y}_k = \mathbf{e}_k$ and

$$\alpha_j^k = \sum_{l=k}^{m} a_{m+k-l} x_{(j-1)s-m+l}. \tag{7}$$

Note that to compute vectors $\mathbf{y}_k$ we need to find the solution of the system $L\mathbf{y}_1 = \mathbf{e}_1$, namely $\mathbf{y}_1 = (1, y_2, \ldots, y_s)^T$. Then we can form vectors $\mathbf{y}_k$ as follows

$$\mathbf{y}_k = (\underbrace{0, \ldots, 0}_{k-1}, 1, y_2, \ldots, y_{s-k+1})^T. \tag{8}$$

The algorithm proceeds as follows. In the first step we find $\mathbf{y}_1$ and all vectors $\mathbf{z}_j = L^{-1}\mathbf{f}_j$, $j = 1, \ldots, r$. In the second step, we use (6) to calculate m last entries of each vector $\mathbf{x}_j$, $j = 2, \ldots, r$. Then again we apply (6) to find $s - m$ first entries of all vectors $\mathbf{x}_j$ (except for $\mathbf{x}_1$). Finally we calculate $x_{rs+1}, \ldots, x_n$ using (1). Note, that the first and the third step can be easily parallelized, while the second and the final step are sequential.

3 New *BLAS-Based* Algorithm

The idea of our new algorithm is to express the *divide-and-conquer* approach presented in the previous section in terms of level 2 and 3 BLAS operations. During the first three steps of the algorithm we have to find the matrix

$$X = (\mathbf{x}_1, \ldots, \mathbf{x}_r) \in \mathbb{R}^{s \times r}.$$

For the sake of simplicity we introduce the following notation. Let $M \in \mathbb{R}^{m \times n}$. Then $M_{i:j,k:l}$ denotes the submatrix of M formed by intersection of rows i to j and columns k to l. Moreover, $M_{i:j,*} = M_{i:j,1:n}$, $M_{*,k:l} = M_{1:m,k:l}$ and $M_{i:j,k} = M_{i:j,k:k}$, $M_{i,k:l} = M_{i:i,k:l}$. Next, using the numbers $a_1, \ldots, a_m$ introduced in (1), we define the following upper triangular matrix

$$C = \begin{pmatrix} a_m & a_{m-1} & \cdots & a_1 \\ 0 & a_m & \cdots & a_2 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & a_m \end{pmatrix} \in \mathbb{R}^{m \times m}. \tag{9}$$

In the first step of the algorithm we find $\mathbf{y}_1$ and all vectors $\mathbf{z}_j$. Obviously, it is equivalent to the problem of solving the block system of linear equations

$$LZ = F, \tag{10}$$

where

$$Z = (\mathbf{z}_1, \ldots, \mathbf{z}_r, \mathbf{y}_1), \; F = (\mathbf{f}_1, \ldots, \mathbf{f}_r, \mathbf{e}_1) \in \mathbb{R}^{s \times (r+1)}. \tag{11}$$

Initially, we assign $Z = F$ and then we can calculate the solution of (10) row by row using

$$Z_{k,*} = \begin{cases} \mathbf{0} & \text{for } k \le 0 \\ Z_{k,*} + \sum_{j=1}^{m} a_j Z_{k-j,*} & \text{for } 1 \le k \le s. \end{cases} \tag{12}$$

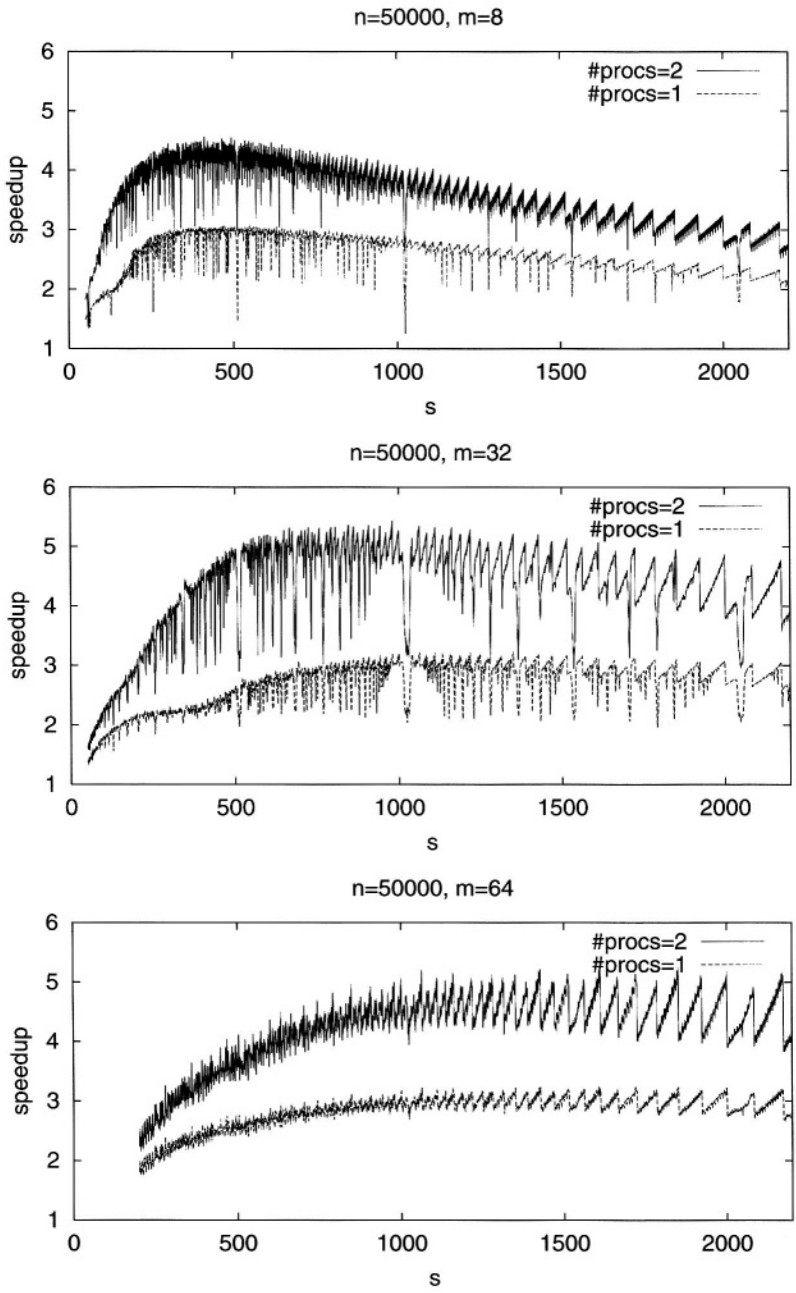


Fig. 1. Speedup of the BLAS-based algorithm (various m and s). Predicted optimal values of s are 611, 1253 and 1787 respectively.

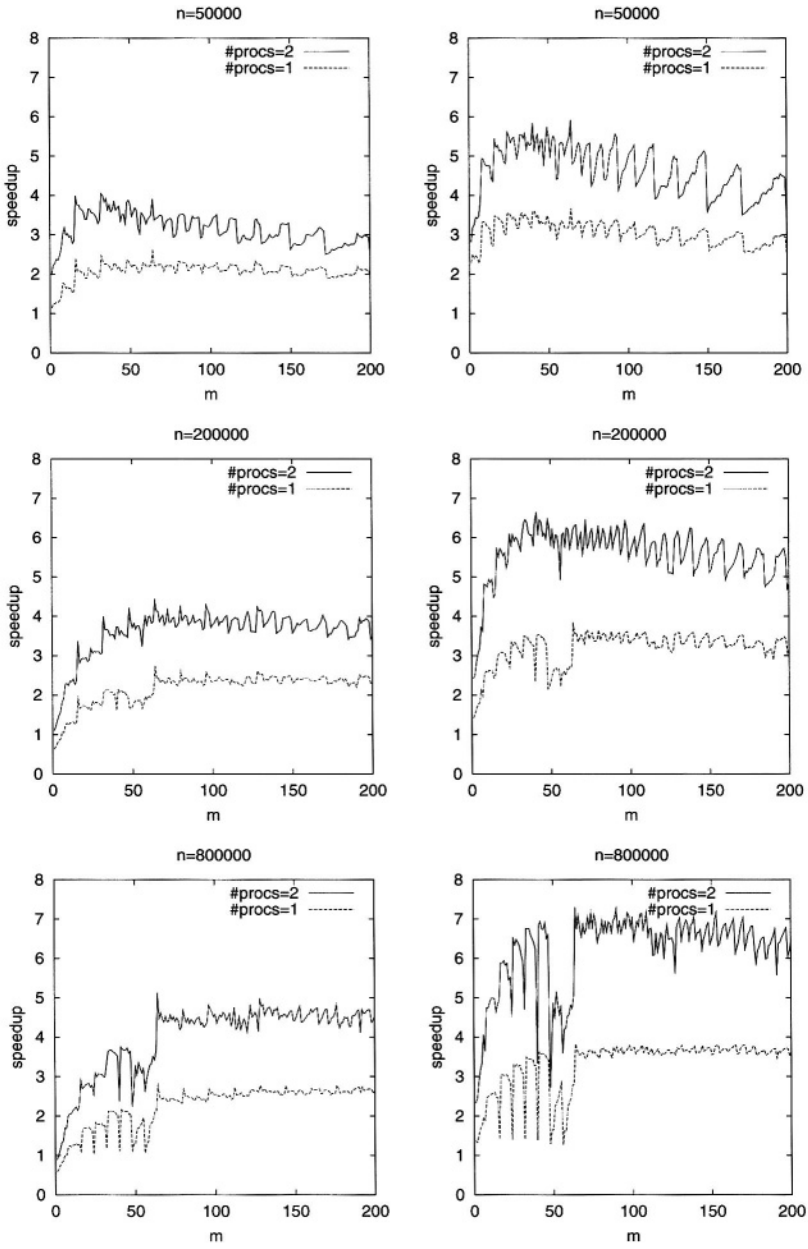


Fig. 2. Speedup of the BLAS-based algorithm (various m and n): full solution (left) and partial solution (right).

Note that (12) can be easily formulated in terms of the matrix-vector multiplication (the operation _GEMV from level 2 BLAS). Thus after the sequence of operations

$$Z_{k,*} \longleftarrow Z_{k,*} + C_{1,\max\{1,m-k+2\}:m} Z_{\max\{1,m-k\}:k-1,*} \tag{13}$$

performed for $k = 2, \ldots, s$, we have $Z = (\mathbf{z}_1, \ldots, \mathbf{z}_r, \mathbf{y}_1)$. Note that

$$Z_{*,1} = X_{*,1} = (x_1, \ldots x_s)^T .$$

Now let $X = Z$. In the second step of the algorithm we compute m last entries of all vectors $\mathbf{x}_2, \ldots, \mathbf{x}_r$ (i.e. $X_{s-m+1:s,2:r}$), and form the matrix

$$A = \begin{pmatrix} \alpha_2^1 & \cdots & \alpha_r^1 \\ \vdots & & \vdots \\ \alpha_2^m & \cdots & \alpha_r^m \end{pmatrix} \in \mathbb{R}^{m \times (r-1)}, \tag{14}$$

where coefficients α_j^k are given by (7). Thus, when we define $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_m) \in \mathbb{R}^{s \times m}$, then we can complete this step after the sequence of operations

$$\begin{cases} A_{*,j-1} \longleftarrow C X_{s-m+1:s,j-1} \\ X_{s-m+1:s,j} \longleftarrow X_{s-m+1:s,j} + Y_{s-m+1:s,*} A_{*,j-1} \end{cases} \tag{15}$$

performed sequentially for $j = 2, \ldots, r$. Note that (15) consists of two calls to the routines from level 2 BLAS, namely _TRMV ("triangular matrix"-vector multiplication) and _GEMV.

In the third step of the algorithm we have to calculate $s - m$ first entries of all vectors $\mathbf{x}_2, \ldots, \mathbf{x}_r$ (i.e. $X_{1:s-m,2:r}$). It can be done using one call to the level 3 BLAS routine _GEMM (matrix-matrix multiplication) [14]:

$$X_{1:s-m,2:r} \longleftarrow X_{1:s-m,2:r} + Y_{1:s-m,*} A. \tag{16}$$

It should be pointed out that sometimes (see [4,13]) it is sufficient to find only the numbers $x_{n-m+1}, \ldots, x_n$ (partial solution) instead of finding all numbers $x_1, \ldots, x_n$ (full solution). In such a case, the third step is simply omitted. The algorithm can be easily parallelized. Especially, in the first and the third step which are based on matrix-vector and matrix-matrix multiplication, each processor can be responsible for computing a block of columns. The second step is sequential.

4 Complexity

Now let us consider the number of floating-point operations required by the simple algorithm based on (1) and the BLAS-based algorithm.

Proposition 1 ([11]). *The number of floating-point operations required by the simple algorithm based on (1) is*

$$T_{seq}(n, m) = 2m \left(n - \frac{m+1}{2} \right). \tag{17}$$

Proposition 2. *The number of floating-point operations required by the BLAS-based algorithm defined by (13), (15), (16) is*

$$T_{BLAS}(n,m;r,s) = 2mn + 2mrs - 2m^2 - mr - m. \tag{18}$$

Proof. Counting the number of flops required by BLAS routines _GEMV, _TRMV, _GEMM used in our algorithm and the number of flops required to find the numbers $x_{rs+1},\ldots,x_n$ using (1), we get

$$T_{BLAS}(n,m;s,r) = 2\left(s - \frac{m+1}{2}\right)(r+1)m + 3m^2(r-1) + 2(s-m)m(r-1) + 2m(n-rs) = 2mn + 2mrs - 2m^2 - mr - m.$$

■

Now we shell consider the problem: *how to choose values of* r, s. Clearly, the total number of flops depends on the value of these parameters. However, the last step of the algorithm (16) can be done by one call to the routine from level 3 BLAS, thus we can expect that this step will be the most profitable. So let us find the value of s by minimizing the total number of flops in the first and the second step of the algorithm. Thus, we find the minimum of the function

$$T_{1+2}(n,m;s,r) = 2\left(s - \frac{m+1}{2}\right)(r+1)m + 3m^2(r-1) \tag{19}$$

with the constraint $rs = n$, and because these numbers should be integers, we get the optimal values of r and s

$$r^* = \left\lfloor \sqrt{\frac{2n}{2m-1}} \right\rfloor, \quad s^* = \left\lfloor \sqrt{\frac{2mn-n}{2}} \right\rfloor. \tag{20}$$

Note that when the optimal values of the parameters r and s are used, then the number of flops required by the BLAS-based algorithm is twice as many as for the simple algorithm (1).

5 Results of Experiments

All numerical experiments were carried out on a dual Intel Pentium III 866MHz processor workstation running under Linux with Intel Fortran 95 Compiler which supports OpenMP [1] and the Math Kernel Library (the optimized version of BLAS which supports the streaming SIMD extensions of Intel processors [2]). Results of the experiments can be summarized as follows.

1. The BLAS-based algorithm achieves the best performance for the value of the parameter s close to the predicted optimal value (20) (see Figure 1). When $s = a2^k$ (for integers a, k), the performance rapidly decreases. This is the effect of cache-misses. Thus we suggest to use $s = s^*$ when s is odd, and $s = s^* - 1$ otherwise. Such a choice of s has been used for various tests presented in Figure 2.

2. On one processor, the BLAS-based algorithm for finding a partial solution is up to 4 times faster than the simple algorithm, while the parallel version of this algorithm is up to 7.3 times faster. For a full solution these values are 2.7 and 5.3 respectively (see Figure 2). However for some problem sizes, the performance of the algorithm rapidly decreases.
3. The performance of the BLAS-based algorithm (full solution, two processors) is up to 1400 Mflops, while the performance of the simple algorithm based on (1) is only up to 145 Mflops.

References

1. Intel Fortran programmer's manual. (http://www.intel.com)
2. Intel Math Kernel Library. reference manual. (http://www.intel.com)
3. Baker, A., Dennis, J., Jessup, E.R.: Toward memory-efficient linear solvers. Lecture Notes in Computer Science **2565** (2003) 315–238
4. Bario, R., Melendo, B., Serrano, S.: On the numerical evaluation of linear recurrences. J. Comput. Appl. Math. **150** (2003) 71–86
5. Dongarra, J., Duff, I., Sorensen, D., Van der Vorst, H.: Solving Linear Systems on Vector and Shared Memory Computers. SIAM, Philadelphia (1991)
6. Dongarra, J., Hammarling, S., Sorensen, D.: Block reduction of matrices to condensed form for eigenvalue computations. J. Comp. Appl. Math **27** (1989) 215–227
7. Larriba-Pey, J.L., Navarro, J.J., Jorba, A., Roig, O.: Review of general and Toeplitz vector bidiagonal solvers. Parallel Computing **22** (1996) 1091–1126
8. Murli, A., Rizzardi, M.: Algorithm 682: Talbot's method for the Laplace inversion problem. ACM Trans. Math. Soft. **16** (1990) 158–168
9. Paprzycki, M., Stpiczyński, P.: Parallel solution of linear recurrence systems. Z. Angew. Math. Mech. **76** (1996) 5–8
10. Stoer, J., Bulirsh, R.: Introduction to Numerical Analysis. 2nd edn. Springer, New York (1993)
11. Stpiczyński, P.: Parallel algorithms for solving linear recurrence systems. Lecture Notes in Computer Science **634** (1992) 343–348
12. Stpiczyński, P.: Efficient data-parallel algorithms for computing trigonometric sums. Ann. Univ. Mariae Curie-Sklodowska Sect. A **56** (2002) 85–96
13. Stpiczyński, P.: Fast parallel algorithms for computing trigonometric sums. In Tudruj, M., Jordan, A., eds.: Proceedings of PARELEC 2002 – International Conference on Parallel Computing in Electrical Engineering, IEEE Computer Society Press (2002) 299–304
14. Stpiczyński, P.: A new message passing algorithm for solving linear recurrence systems. Lecture Notes in Computer Science **2328** (2002) 466–473
15. Van Der Vorst, H.A., Dekker, K.: Vectorization of linear recurrence relations. SIAM J. Sci. Stat. Comput. **16** (1989) 27–35

Accelerating Optimization of Input Parameters in Wildland Fire Simulation

Baker Abdalhaq, Ana Cortés, Tomàs Margalef, and Emilio Luque

Computer Science Department, ETSE, Universitat Autònoma de Barcelona, 08193-Bellaterra (Barcelona) Spain**
baker@aows10.uab.es,{ana.cortes,tomas.margalef,emilio.luque}@uab.es

Abstract. Fire propagation simulation is seen as a challenging problem in the area of simulation, due to the complexity of the physical models involved, the need for a great amount of computation and the difficulties of providing accurate input parameters. Input parameters appear as one of the major sources of deviation between predicted results and real-fire propagation. Evolutionary algorithms have been used to optimize the input parameters. However, such optimization techniques must be carried out during real-time operation and, therefore, certain methods must be applied to accelerate the optimization process. These methods take advantage of the computational power offered by distributed systems.

1 Introduction

Fire-propagation modeling is a highly active research area that tries to provide models to predict the behavior of fire, so that an accurate forecast of its evolution can be provided. To become genuinely operative, the models must be integrated into simulation tools; taking several input parameters, these estimate fire propagation. In most cases, the results provided by simulation tools do not match real propagation. Thus, that simulation tools are not wholly useful, since predictions are not reliable. One of the most common sources of deviation from real propagation is imprecision in input parameters. A way of overcoming this problem consist of optimizing the input parameters with the aim of finding an input set so that predicted fire propagation matches real fire propagation [1]. From an initial fire and real propagation after a certain time interval, the optimization techniques are applied in order to determine the set of parameters providing the best match between simulation and real behavior. These values are then used to predict the behavior in the next interval. The prediction is once again compared to real behavior and the optimization process is repeated so that the optimization method adapts to the dynamic changing of conditions (wind, moisture content, etc.).

Parameter optimization and prediction must be carried out faster than real time so that the prediction can be useful in deciding which actions need to be

** This work has been supported by the CICYT under contract TIC2001-2592 and by the European Commission under contract EVG1-CT-2001-00043 SPREAD.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1067–1074, 2004.
© Springer-Verlag Berlin Heidelberg 2004

taken in tackling the emergency. However, the number of parameters is quite large and the resulting search space becomes enormous. It is not, therefore, feasible to assess the whole search space, which needs to be reduced by applying certain techniques.

The current state of the art in the computational field offers the required background to be applied. On the one hand, evolutionary computing is a well-established field with several techniques in the literature that are widely accepted (as, for example, Genetic Algorithms [2]). These techniques can be applied to guiding the search over the whole space, so that only certain cases are tested. On the other hand, computing systems based on parallel and distributed platforms offer the required computing power to apply these techniques and to provide successful results in an acceptable time. Typically, these techniques work in an iterative way by improving the obtained solution at each iteration. The computational time consumed for this searching process clearly depends on two elements: the function to be optimized (objective function), which involves the execution of the underlying simulator, and the number of times the optimization process should be iterated to converge.

A clear way of saving time consists of improving the convergence speed of the optimization technique. For this purpose, we propose applying a sensitivity analysis to the input parameters in order to asses their impact on output and, consequently, to determine which parameters are worth spending time on tuning and which are better to avoid spending effort on, maintaining them instead at an estimated value. In order to be more effective in tuning the most sensitive parameters, we also propose introducing a certain degree of knowledge during the optimization process. This knowledge will consist of limiting the range of the tuned parameters around an estimated value (which may be the real measurement) for those parameters.

The rest of the paper is organized as follows. Section 2 describes the basic ideas behind wildland fire simulation. Section 3 is devoted to the sensitivity analysis carried out. Section 4 reports on the experimental study carried out and the results obtained. Finally, section 5 presents the main conclusions of this work.

2 Wildland Fire Propagation Simulation Optimization

Since our objective consists of finding the combination of input parameters that minimizes the deviation of the simulator prediction from the real scenario as fast as possible, we need to compare the simulated firelines against the real fireline and, according to the results of this comparison, assign a quality measurement to the underlying scenario. Each fireline describes a burned area. To compare the simulated and the real firelines we used the area of the XOR between the real and simulated burned areas. This XOR includes the areas that are burned in one of the propagations but not in the other one. This area can be used as a measurement of the prediction error. Minimizing this error improves the prediction of fire behavior. As optimization technique, we use a Genetic Algorithm (GA)

[2]. A GA identifies the parameters vector to be optimized as a chromosome. The chromosome will be weighted according to a fitness function, which in our case consists of the prediction error defined above. We use two operators to alter the chromosomes. The first is mutation, which changes the value of the parameter to another that has been randomly chosen from the corresponding parameter range. The mutation is applied to the parameters according to a determined probability. The second operator is crossover in which two chromosomes (called parents) participate to create other chromosomes (called offspring). The offspring is created by copying certain parameters from the parent and taking the average of the two parents for the remaining parameters. The GA will be iterated until either a preset number of iteration is executed (1000 in our case) or a reasonable solution is reached. At each iteration, the objective function (XOR function) is evaluated for a certain number of guesses (20 in our case). Since each guess can be independently treated, we have applied an optimization framework that works in a master/worker scheme whereby the master executes the GA operators and the workers are responsible for executing the objective function of the different guesses.

3 Sensitivity Analysis

Sensitivity Analysis (SA) classically aims to ascertain how the model depends upon the information fed into it (input model/simulator parameters). The objective of any sensitivity analysis is to identify the most important factor among all inputs, which will be defined as the input that, if determined (i.e. fixed to its true although unknown value), would lead to the greatest reduction in the variance of the model/simulator output. Likewise, we can define the second most important factor, and so on, until all factors are ranked in order of importance.

The method we used here is based on nominal range sensitivity analysis, which is also known as local sensitivity analysis or threshold analysis [5]. Basic nominal sensitivity analysis evaluates the effect on the model output exerted by individually varying only one of the model inputs across its entire range of possible values, while holding all other inputs at their nominal or base-case values. The difference in the model output due to the change in the input variable is referred to as the sensitivity or swing weight of the model to that particular input variable, in that given case.

However, there may be interdependencies among the parameters. Therefore, the effect of one parameter may depend on the values of the fixed parameters. The nominal sensitivity analysis must therefore be repeated for each parameter for all possible cases and combinations of all the other parameters. In the particular case of fire propagation, the number of parameters is quite high and the number of combinations that must be evaluated in order to reach the sensitivity index is enormous.

3.1 Calculating the Sensitivity Index

The sensitivity of the parameters, in our case, depends on the fire propagation model used in the core of the objective function. For a generic study, we studied the effect of the parameters of the model in one dimension on propagation speed, thus the wind has only one scalar value, which is the speed of the wind in the direction of the fire propagation. To calculate the sensitivity index for each parameter, it is necessary to define a minimum and maximum value for the parameter, which are typically obtained from the field and lab measurements. For all the possible combinations of the other parameters, therefore, two simulations are executed considering the minimum and the maximum value of the parameter currently studied. The speed difference between both propagation simulations represents the effect of changing that particular parameter from its minimum to its maximum for that particular combination of the other parameters. Let V_{ik} be the effect of varying factor i from its minimum to its maximum (difference of the speed of the minimum and the speed of the maximum) at case k.

The total effect of parameter i is defined as the addition of the effect of each possible case:

$$V = \sum V_{ik} \tag{1}$$

where k is all the possible cases (combinations of input factors). Thus, V_i will be our index of sensitivity for the parameter i. This index not only reflects the effect of the parameter but also the effect of its range.

In this work, we have used the ISStest forest-fire simulator [3], which incorporates the Rothermel model [4]. Therefore, the Rothermel model input parameters will conform the vector that is to be optimised in our case. Table 1 outlines each one of these parameters and their corresponding minimum and maximum values according to [6], also showing the calculated index. Using the value of the index, we can classify the input parameters by their sensitivity. This table shows that the two most important parameters are the load parameters ($W0$,β); the third is wind (U), followed by humidity (Mf). The parameters with weakest effect are metal content (St, Se) and heating content (h). This result concords with the results obtained by [7], which also uses the Rothermel set of equations as a forest fire propagation model. Since sensitivity analysis implies a high number of simulations, we have also used the master/worker programming paradigm to evaluate all sensitivity indexes.

4 Experimental Study

The experimental study was carried out on a Linux cluster composed of 12 PCÂ's with Intel Celeron processor 433 MHz, each one with 32 MB RAM, connected to a Fast Ether Net 100 Mb. All the machines were configured to use NFS (Network File System) and MPI was used as a message-passing interface. To properly evaluate the XOR area after executing the ISStest simulator for each guess, we need to have a reference fireline for comparison. For this purpose, a synthetic

Table 1. Ranges used to calculate the sensitivity index.

parameter	Min	Max	Index
$W0$	0,1	4	0,77
β	0,01	0,11	0,86
σ	315	11500	0,56
St	0,001	0,08	0,03
Se	0,0001	0,07	0,16
Mx	0,1	0,4	0,28
Mf	0	Mx	0,61
h	18571429	22000000	0,13
U	0	15	0,71

fireline was obtained by setting the values of all inputs to certain known values, and by using a simulation time equal to 45 minutes. We assumed homogeneous vegetation through the whole terrain, which consisted of a flat area. Once the synthetic real fireline was obtained, it was dismissed and was only used as a comparative member for calculating the XOR during the optimization process. As we have commented, a GA was used as the optimization technique. Since the genetic algorithm has random factors, the global optimization process was performed 10 times and all results were averaged. All the values reported in this study therefore correspond to the mean values of the corresponding 10 different experiments conducted.

4.1 Fixing the Parameters with Smaller Sensitivity Indexes

Considering the definition of the sensitivity index, if we were able to find the real value of the parameters with greatest importance, we would minimize the divergence of the simulator from the reality. Therefore, it is crucial to calibrate the parameters that have a greater sensitivity index while we do not know their real values. Likewise, we can say that calibrating the parameters that have little effect on the result will not improve the simulator results significantly, and this will consume processing time. The impression is therefore created that it is not worth tuning the parameters with a small sensitivity index. We suppose that fewer parameters to be optimized will make the convergence faster and, at the same time, fixing certain unimportant parameters to a given value with a reasonable error will not deviate the optimization process too far from the global minimum. This experiment is designed to observe the effect of removing the parameters that have a small sensitivity index on the convergence of the optimization process. As estimated values for the parameters that are to be fixed, we have their real values plus 10% of its full range. This percentage of variation allows us to experiment the effect of having an error of 10% in the estimation of the parameter. Table 2 shows the real value of the less sensitive parameters, and their corresponding estimated values, when applying this estimation error (10%).

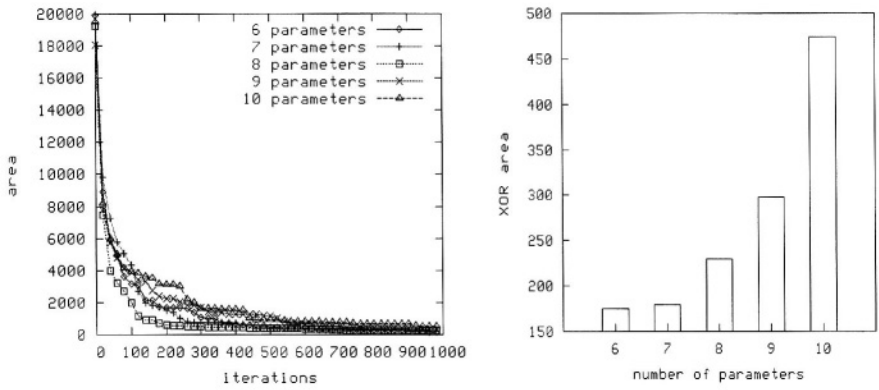


Fig. 1. Optimization Convergence Changing the Number of Parameters (left) and at iteration 1000 (right)

Figure 1 shows the convergence of the optimizing process by reducing the number of parameters to optimized. Each curve differs from the other by omitting one parameter each time, i.e., the curve labeled (10 parameters) shows the convergence of the tuning process when all parameters are considered. The curve labeled (9 parameters) plots the convergence evolution when tuning all the parameters except that with a smaller sensitivity index (St in this case), and so on. As we can observe, we can clearly distinguish two different phases: one phase from the beginning of the optimization process up to iteration 500, approximately, and a second phase, which goes from iteration 500 until the end. During the first phase, independently of the number of parameters optimized, we clearly observe fast improvement and, in particular, the case where two parameters are fixed (the curve labeled 8 parameters) has a superior optimization performance. In contrast, in the second phase, the convergence speed for all cases seems to stabilize. However, since it is well known that GAs have a warm-up phase, we applied statistical hypothesis testing [8] to the results in order to asses whether or not the two observed behaviors can be considered statistically reasonable. We found that there is no statistical difference between the means before iteration 500; consequently, it is irrelevant to discus the behavior of the curves during the first phase of the optimization process. However, at iteration 1000, the results show a statistical difference between optimizing all parameters as opposed to fixing 1,2,3 and 4 parameters. Furthermore, we have observed that there is no statistical difference between optimizing 6 and 7 parameters. Nevertheless, there is a statistical difference between optimizing 6 or 7 parameters and 9 parameters. The mean values of the objective function (XOR area) at the end of the optimization process (iteration 1000) is shown in figure 1(left). As we can see, the objective function for the case of parameters is one third of the mean value obtained for the case of parameters. These results are obtained using an error of estimation equal to 10%. If the error is greater, the practice of fixing the value of the parameters to estimated values will not be good. This method therefore assumes a good estimation of the real parameter value.

Table 2. The real and estimated values of the fixed parameters.

Parameter	Real value	Estimated value
St	0.04	0.04799
H	18971429	19314270
Se	0.02	0.0269
Mx	0.3	0.33

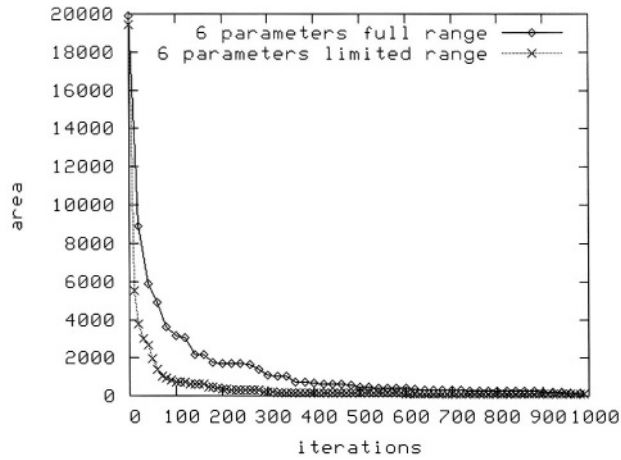


Fig. 2. Optimization convergence comparison using both the full and limited ranges

4.2 Limiting the Search Range

Once we have observed that fixing 4 parameters to a certain estimated value provides a considerable improvement in optimization convergence, we focus on this case to introduce a certain degree of knowledge of the optimized parameters in order to further improve such convergence. We assume that we have some knowledge about the limits within which a parameter can vary, therefore, it is not necessary to search within its full possible range. For the purpose of this experiment, we limited the range of the parameter to 15% above and below its (known value) so as to simulate the expected range. Figure 2 shows the optimization convergence when optimizing 6 parameters using either their full range or a limited searching range. As we can observe, cutting the range of the parameters significantly accelerates optimization convergence. Although from the figure it seems that, at iteration 1000, both situations provide similar results, the limited range at the end of the optimization process provides an objective function (XOR area) equal to 98.71, on average, whereas the final value is 175.47, using the full range.

5 Conclusions

One of the most common sources of fire spread simulation deviation from real fire propagation is imprecision in input simulator parameters. This problem can be approached by applying an evolutionary optimization such as the Genetic Algorithm so as to calibrate input-simulator parameters. Since this approach is a time-demanding task, we have proposed a global sensitivity analysis to accelerate optimization convergence. This technique reduces the search space screened by fixing the less sensitive parameters to an estimated value and by focusing optimization on the most sensitive parameters. We have also reduced the range of each optimized parameter by introducing some degree of knowledge of each of them. This was considered by limiting the variation of these parameters around a known value (field measurement). Both techniques were carried out on a Linux cluster composed of 12 PCs. We used a master/worker programming paradigm, where the master and worker processes communicate with each other using MPI. The results show that, combining both accelerating strategies, the convergence improvement obtained is quite significant.

References

1. Baker Abdakhaq, Ana Cortés, Tomás Margalef, Emilio Luque, "Optimization of Fire Propagation Model Inputs: A Grand Challenge Application on Metacomputers". LNCS 2400, pp. 447-451. (2002).
2. Coley David A.: "An Introduction to Genetic Algorithms for Scientists and Engineers" , World Scientific, 1999.
3. Jorba J., Margalef T., Luque E., J. Campos da Silva Andre, D. X Viegas "Parallel Approah to the Simulation Of Forest Fire Propagation". Proc. 13 Internationales Symposium "Informatik für den Umweltschutz" der Gesellschaft für Informatik (GI). Magdeburg (1999) pp. 69-81
4. Rothermel, R. C., "A mathematical model for predicting fire spread in wildland fuels", USDA FS, Ogden TU, Res. Pap. INT-115, (1972).
5. Satelli, A., K. Chan, M. Scott, Editors. "Sensitivity analysis". John Wiley & Sons publishers, Probability and Statistics series. (2000).
6. André, J.C.S., "A Thery on the propagation of surface fire fronts", PhD Dissertation (in protugues), Universidade de Coimbra, Portugal, 1996.
7. Salvador, R., Piol, P, Tarantola, S. and Pla, E. "Global Sensitivity Analysis and Scale Effects of a Fire Propagation Model used Over Mediterranean Shrub lands". Elsevier, Ecological Modelling 136 pp. 175-189, (2001).
8. Wadsworth, Harrison M. "Handbook of statistical methods for engineers and scientists", McGraw Hill, Inc. (1990).

A Tool to Execute ASSIST Applications on Globus-Based Grids

Ranieri Baraglia, Domenico Laforenza, and Nicola Tonellotto

ISTI - Institute of the Italian National Research Council, Pisa, Italy
{ranieri.baraglia, domenico.laforenza, nicola.tonellotto}@isti.cnr.it

Abstract. This article describes ASSISTCONF, a graphical user interface designed to execute ASSIST applications on Globus-based Grids. ASSIST is a new programming environment for the development of parallel and distributed high-performance applications.
ASSISTCONF hides to the programmer the structure of the grid used and integrates the ASSIST Run Time System with the Globus middleware. The first version of ASSISTCONF was designed to manually configure an ASSIST application and to establish a mapping between the application components and the machines selected for its execution on the Grid. The new ASSISTCONF functionalities, such as authentication and execution authorization on the resources selected in the application mapping phase, and deployment on the selected resources of the ASSIST Run Time Support, the executable application components, and the application input data, allow the semi-automatic execution of an ASSIST application on a such environment.

1 Introduction

Grid programming requires capabilities and properties beyond those of simple sequential programming, or even traditional parallel and distributed programming [6]. Typical parallel/distributed grid applications are modular, written by several development teams using different programming languages, and, in general, use multi-source heterogeneous data. Furthermore, due to the intrinsic dynamic nature of a grid computational environment, grid applications need to be more and more adaptive to run on different type of resources the availability of which and configuration may change at application run-time [7,8].

These peculiarities led to the development of new programming environments, which permit an effective and flexible application composition, and deal with heterogeneous machines and networks with different performance. It is desirable to have programming environments that hide the dynamicity and heterogeneity of the Grid by a common "look and feel" to the programmer, and make available coordination mechanisms to compose and execute an application.

By using coordination languages, new grid applications can be built by writing new components from scratch and/or by reusing existing off-the-shelf components and applications. In the past various coordination languages have been proposed, in which a set of parallel constructs (skeletons) are used as program

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1075–1082, 2004.
© Springer-Verlag Berlin Heidelberg 2004

composition forms [9,10,11]. A skeleton makes all the details involved in the parallel computation structure transparent to the programmer.

By exploiting skeleton-based and component-based technologies a new programming environment called ASSIST (A Software development System based upon Integrated Skeleton Technology) has been proposed.

This paper describes the main functionalities of ASSISTCONF, a graphical user interface designed to execute ASSIST applications on Globus-based Grids [1] . It simplifies the creation of the ASSIST configuration file, giving users a graphical view of the XML file produced by the ASSIST compilation phase. The tool allow an easy identification of the machines to be used for the application execution by automatically finding the resources candidate to execute an ASSIST application. Finally, the configuration file produced by ASSISTCONF is used to launch the execution of the ASSIST program on the Grid. The first version of ASSISTCONF [13] was designed to manually configure an ASSIST application and to establish a mapping between the application components and the machines selected for its execution on the Grid. The new ASSISTCONF functionalities support the semi-automatic execution of ASSIST applications on a such environment.

This paper is organized as follows. Section 2 outlines the main characteristics of the ASSIST programming environment. Section 3 describes the ASSIST runtime support, and how an ASSIST program have to be configured to run on Grid. Section 4 describes the ASSISTCONF main functionalities. Finally, the current and future work are summarized in Section 5.

2 The ASSIST Programming Environment

ASSIST is a new programming environment oriented to the development of parallel and distributed high-performance applications according to a unified approach, which matches the features of component technology and the features of structured parallel programming technology [5]. The proposal originates from previous research conducted in the structured skeleton–based parallel programming field [2,3]. The main goals of ASSIST are allowing high-level programmability and software productivity for complex multidisciplinary applications, and performance portability across different platforms, including homogeneous parallel machines and cluster/Beowulf systems, heterogeneous clusters and network computers, and computational Grids.

The structure of an ASSIST program is a generic graph, the nodes of which are software components and the arcs are abstract interfaces that support streams, i.e. ordered sequences, possibly of unlimited length, of typed values. Streams are the structured way to compose components into an application. Moreover, components can also interact as external objects (e.g. CORBA components). An ASSIST application is implemented by using the ASSIST-CL coordination language that defines and regulates the interaction among components. The latter ones can be parallel or sequential ASSIST modules. A sequential module is the simplest component that can be described within ASSIST (cur-

rently, the sequential programming language hosted by the ASSIST-CL are C, C++, and FORTRAN). It has an internal state, and it is activated by the input stream values according to a non-deterministic data-flow behavior. A parallel component may be an ASSIST subgraph, e.g. an independently designed ASSIST program, or a parallel module implemented with a *parmod* construct. The latter is a sort of generic skeleton that can be programmed to emulate the most common specific skeletons, and also to easily express new forms of parallelism (e.g. optimized forms of task + data parallelism, nondeterminism, interactivity), as well as their variants and combinations [5].

ASSIST-CL is a coordination language aimed to increase software productivity for complex multidisciplinary applications. ASSIST-CL allows programmers to declare specific forms of parallelism (skeletons) that can be used to hierarchically compose sequential/parallel components.

The target parallel architectures supported by the ASSIST programming environment range from homogeneous/heterogeneous clusters of sequential/SMP workstations to computational Grids. Therefore, to ensure *code and performance portability,* programs need to be reconfigured on the basis of the specific features of each target architecture. For example, decisions such as degree of parallelism of data-parallel modules, and number of replicated modules should be postponed until loading time, when the features of the target architecture (e.g. number and type of processors available) are known. The ASSIST run-time support (ASSIST RTS) provides the functionalities needed to reconfigure an ASSIST program. A detailed description of the programming model of ASSIST, and the constructs provided by ASSIST-CL can be found in [5].

The current implementation of the ASSIST environment is based on a flexible abstract machine, that exploits the underlying mechanisms of ACE [14] and Distributed Shared Memory libraries [15]. The first version of the implementation currently run on homogeneous parallel machines and clusters (Linux), and also contains basic interfaces for experimenting ASSIST in heterogeneous Grids. Work is in progress to define and to realize the next version of ASSIST, that will progressively remove some constraints in order to fully exploit heterogeneous large-scale platforms and Grids.

For the purposes of this paper, we are concerned with the execution of an ASSIST program on a Grid computational environment.

3 The ASSIST Run Time Support and Its Configuration for Grid Execution

The abstract machine that implements the ASSIST RTS is logically structured according to two layers (see Figure 1).

The CLAM (Coordination Language Abstract Machine) layer is utilized to run ASSIST programs, while HALI (Hardware Abstraction Layer Interface) layer exports all functionalities used to implement the CLAM layer. In this way the ASSIST compilation mechanisms are independent from the machine used, and an ASSIST application can be run on each machine for which the level HALI

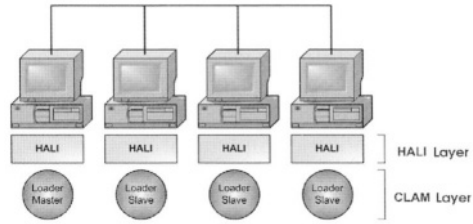


Fig. 1. The ASSIST abstract machine structure.

is available. The Loaders are the main processes of the CLAM layer, and each of them runs on a distinct node of the platform used. The Loader processes are devoted to start up and manage the execution of the modules produced by the ASSIST compiler. One of them acts as a master to coordinate the execution, monitoring, and run-time reconfiguring, in presence of load imbalance, of the various ASSIST modules. After receiving a synchronization message by each slave Loader, the master, by sending a start message, enables all the Loaders (itself included) to launch the execution of the module associated to them.
In Figure 2 an overview of the ASSIST compilation process is given. The compiler produces the various binary modules, and a XML configuration file. The type of the modules produced by the ASSIST compiler is function of the target computational platform used to run the application. When the target computational platform is a Grid, the ASSIST compiler produces executable modules, that embodie functionalities owned by the HALI and CLAM layers. This is due to the dynamic and not reliable nature of the Grid; moreover, the computational nodes available may belong to distinct VOs, that, in general, want to minimize any kind of intrusiveness. The XML configuration file is logically subdivided according to three sections called *Structure, Configuration,* and *Loading*. The first one is set by the ASSIST compiler and contains information about the software modules produced and about the structure of the ASSIST program. Using ASSISTCONF the Configuration and Loading sections are set. These sections specify the configuration of the program (i.e. degrees of parallelism and number of replicated instances of a module), and the mapping information (i.e. mapping information about the ASSIST module instances), respectively.

4 ASSISTCONF

ASSISTCONF is a GUI written in Java that allows the programmer to configure and execute an ASSIST application on Globus-based Grids. It hides the structure of the grid used to the programmer, and integrates the ASSIST RTS with the Globus middleware.

The ASSISTCONF main functionalities are aimed to:

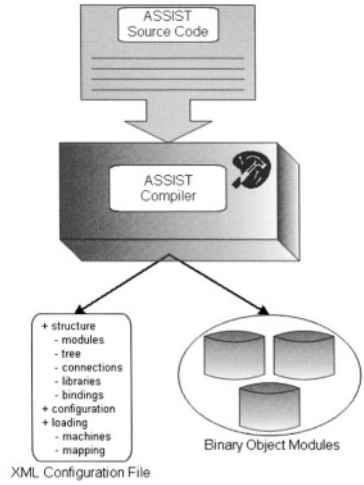


Fig. 2. Scheme of the ASSIST compilation process.

- select the computational resources needed to run an application,
- configure an ASSIST application by setting the parallelism degree of parallel modules and the number of instances of a replicated module,
- assist the user to establish a mapping of the various modules on the selected computational resources,
- stage on the selected computational resources the libraries (e.g. ACE), executable modules, and input files needed to run the ASSIST application,
- activate the execution of the executable modules,
- transfer the output files to the user machines and delete, if required, all the files used to run the application from the machines used.

Figure 3 depicts the ASSISTCONF main windows. A message area is shown at the bottom left, in which errors and information messages are displayed. The Project and Resources Pool areas display the files related to the application under development and the machine pools available to run the ASSIST application, respectively. The menu bar includes the following items: File, Pool, Security, Transfer, and Execution that contain commands to manage all the activities to run the Grid application, and the Globus Toolkit security mechanisms. Several other commands are selectable by specific *context menus* in each screen area.

To configure an ASSIST program, a project has to be created by opening the related XML configuration file (see Figure 3). We refer to this first version of the XML file, as the ASSIST *Configuration Source* (`.acs` extension).

In Figure 3 is shown a configuration file corresponding to a program structured according to a three stages pipe in which the first and the last ones are sequential modules, and the second one is a parallel module. The parallelism degree of a parallel module and the number of instances of a replicated module can be specified, and the configuration file is accordingly modified.

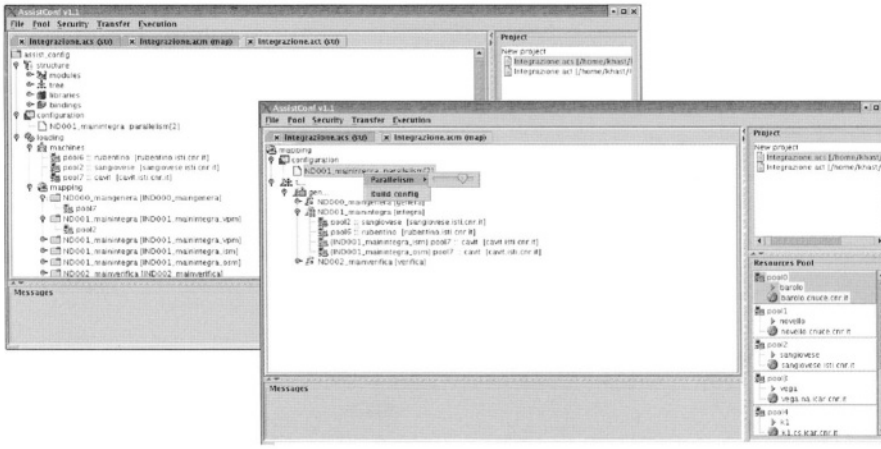


Fig. 3. ASSISTCONF main windows.

In order to carry out a mapping for an application, the user needs to select the suitable machines by accessing a list of available machines obtained directly from ASSISTCONF by accessing the Grid Information Service (GIS). This is implemented by querying the LDAP servers that make up the GLOBUS GIS. The final step of the configuration phase is to establish a mapping between the program modules and the machines in the Grid. This task is accomplished by activating a pool selection context menu in the main GUI panel.

In order to execute an ASSIST application we exploit the Globus staging and execution mechanisms (i.e. Globus GridFTP, RSL and GRAM APIs) which require a valid security proxy. To do this, ASSISTCONF provides functionalities to create and manage a proxy of a valid X.509 certificate. The input files and libraries to be staged can be selected from a list of local files and directories displayed by clicking the Add button (see Figure 4(a)). The executable files are selected by directly accessing the XML configuration file. Moreover, the remote destination directory must be entered in the proposed textbox.

The stage operation can be optionally monitored. An example of stage monitoring operation is given in Figure 4 (b).

To execute the application, ASSISTCONF generates a RSL string for each executable module; the information required to generate RSL strings is obtained by the XML configuration file. The application execution is activated by clicking the Start button in the Execution window (see Figure 5). Moreover, in such window, the modules execution progress is shown.

5 Conclusions and Future Work

In this paper we have presented ASSISTCONF, a GUI designed to provide an easy way to execute ASSIST applications on Globus-based Grids. ASSISTCONF

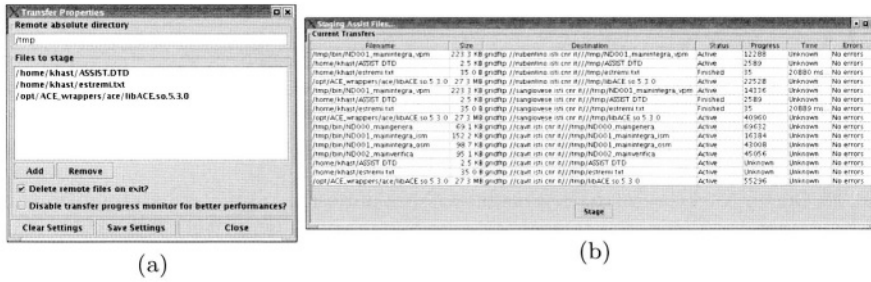


Fig. 4. Selected files (a) and stage windows (b).

Fig. 5. Application execution window.

mainly aims at simplifying the creation of the ASSIST application configuration file, giving users a graphical view of the XML file produced by the ASSIST compilation phase, and permitting an easy identification of the machines to be used for the application execution. The application configuration file is then used to automatically drive the execution of the application on the Grid. The current version of ASSISTCONF only provides a semi-automatic mechanism for selecting the computational resources required to run an ASSIST Grid application. Moreover, the tool does not provide any automatic mapping mechanism. The next version of ASSISTCONF will integrate a Grid *broker* to automatically select computational resources satisfying the computational application requirements (e.g. disk space, memory) and to automatically carry out a mapping scheme of the application modules on the selected machines. Moreover, performance information describing the Grid resources utilization should be exploited by the mapping mechanisms.

Acknowledgments. The authors would like to thank the Prof. Vanneschi's group for their contribution to the understanding of the ASSIST overall architecture. Moreover, they would also like to thank Drs. Raffaele Perego and Paolo Palmerini and Prof. Salvatore Orlando for their contribution to design the first ASSISTCONF version. This work was funded by the Italian Ministry of Education, University and Research (MIUR) as part of the National Project MURST 5% 1999 Grid Computing: Enabling Technology for eScience.

References

1. http://www.globus.org/
2. S. Orlando, B. Bacci, M. Danelutto, S. Pelagatti, M. Vanneschi, P3L: a Structured High-level Parallel Language and its Structured Support", Concurrency: Practice and Experience, 7(3), May 1995, pp. 225-255.
3. B. Bacci, M. Danelutto, S. Pelagatti, M. Vanneschi, SkIE: A heterogeneous environment for HPC applications, Parallel Computing, 25 (1999) 1827-1852.
4. M. Cole, Algorithmic skeletons: structured management of parallel computations, MIT Press, 1989.
5. M. Vanneschi, The programming model of ASSIST, an environment for parallel and distributed portable applications, Parallel Computing, 28 (2002) 1709-1732.
6. GWD-I, Advanced Programming Models Research Group, http://www.eece.unm.edu/apm/docs/APM Primer 0801.pdf, August 2001
7. F. Darema, Next Generation Software Research Directions. In http://www.cise.nsf.gov/eia/ NGS-slide/sld001.htm
8. D. Laforenza, Grid Programming: Some Indications Where We Are Headed. Parallel Computing, North-Holland Elsevier, vol. 28/12, pp. 1701-1720, Dec. 2002.
9. B. Bacci, M. Danelutto, S. Pelagatti, M. Vanneschi, SkIE: A heterogeneous environment for HPC applications, Parallel Computing 25(13-14):1827-1852, 1999.
10. D. B. Skillicorn, Models for Practical Parallel Computation, International Journal of Parallel Programming, 20(2):133-158, April, 1991.
11. M. Cole, Algorithmic skeletons: structured management of parallel computation. MIT Press, 1989.
12. M. Vanneschi, Programming Model of ASSIST, an Environment for Parallel and Distributed Portable Application, To be published on Parallel Computing, North-Holland Elsevier, 2002.
13. R. Baraglia et al., AssistConf: a Grid configuration tool for the ASSIST parallel programming environment, Proc. Euromicro Int. Conf. On Parallel, Distributed and Network-Based Processing, pp. 193-200, Genova, Italy, 5-7 February 2003.
14. D. C. Schmidt, The Adaptive Communication Environment: Object-Oriented Network Programming Components for Developing Client/Server Applications, 11th and 12th Sun Users Group Conference, 1994.
15. F. Baiardi, D. Guerri, P. Mori, L. Moroni, L. Ricci, Two Layers Distributed Shared Memory, Proc. HPCN, 2001.
16. Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane, "A Java Commodity Grid Kit," Concurrency and Computation: Practice and Experience, vol. 13, no. 8-9, pp. 643-662, 2001, http:/www.cogkits.org/.

Adaptive Computation over Dynamic and Heterogeneous Networks

Kaoutar El Maghraoui[1], Joseph E. Flaherty[1], Boleslaw K. Szymanski[1], James D. Teresco[2], and Carlos Varela[1]

[1] Rensselaer Polytechnic Institute, Troy, NY 12180, USA,
szymansk@cs.rpi.edu,
http://www.cs.rpi.edu/
[2] Williams College, Williamstown, MA 01267, USA

Abstract. Over the last two decades, efficient message passing libraries have been developed for parallel scientific computation. Concurrently, programming languages have been created supporting dynamically reconfigurable distributed systems over the heterogeneous Internet. In this paper, we introduce SALSA-MPI, an actor programming language approach to scientific computing that extends MPI with a checkpointing and migration API and a runtime system that manages both periodic checkpoints and process or application migration. The goal is to enable dynamic network reconfiguration and load balancing without sacrificing application performance or requiring extensive code modifications. As driving technology for this effort of unifying parallel and distributed computing, we plan to use adaptive solvers of partial differential equations. Fields as diverse as fluid dynamics, material science, biomechanics, and ecology make use of parallel adaptive computation, but target architectures have traditionally been supercomputers and tightly-coupled clusters. SALSA-MPI is intended to allow these computations to make efficient use of more distributed and dynamic computing resources.

1 Introduction

Large-scale scientific and engineering applications involving the solution of partial differential equations are among the most demanding computational problems, arising in fields including fluid dynamics, material science, biomechanics, and ecology. Adaptivity, where meshes and numerical methods are automatically adjusted to achieve specified levels of solution accuracy, and parallelism are essential tools to solve modern multi-dimensional transient problems. The usual approach to these problems is to distribute a discretization (mesh) of the domain across cooperating processors, then to compute a solution, appraising its accuracy using error estimates at each step. If the solution is accepted, the computation proceeds to the next step. Otherwise, the discretization is refined adaptively, and work is redistributed, if necessary, to correct for any load imbalance introduced by the adaptive step. The adaptive strategies automatically refine, coarsen, and/or relocate meshes and may also change the method with a goal of obtaining a solution to a prescribed level of accuracy as quickly as possible [1]. Adaptivity makes automatic (compiler-directed) parallelization difficult, so dynamic partitioning and load

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1083–1090, 2004.
© Springer-Verlag Berlin Heidelberg 2004

balancing procedures become necessary since the locations where meshes must be refined or simpler numerical procedures replaced by more complex ones are not known *a priori* and are determined as part of the solution process.

The adaptive software with the described above features is complex and hard to develop. Hence, the existing software of this kind is very valuable and difficult to replace, motivating our research on combining middleware written in a new actor programming language SALSA with the existing C++ codes using MPI. We target the adaptive software base developed at Rensselaer's Scientific Computation Research Center that executes in serial and parallel computational environments [2,3]. It has been used successfully by many software packages for classical finite element [4], finite volume [5], and discontinuous Galerkin (DGM) [6,7] methods. *DG* [7] is a software package that implements a parallel adaptive DGM using the *Algorithm Oriented Mesh Database* (AOMD) [8] mesh structures and services. AOMD supports a variety of mesh representations, including hybrid meshes. It is written in C++ using the Standard Template Library [9] for computations and the Message Passing Interface (MPI) [10] for communication. DG is used to solve a wide range of problems including Rayleigh-Taylor flow instabilities [7]. Distributed AOMD meshes [3] use the *Rensselaer Partition Model* (RPM) [11] to aid in data distribution and migration.

The importance of the applications and, perhaps, the cost of access to supercomputers have led to proliferation of solution strategies on other architectures including PC clusters and, most recently, grids [12,13]. Target architectures range from small clusters to the largest supercomputers with interprocessor communication ranging from shared memory to wide-area networks.

As discussed in [14] in this volume, open source SALSA actor programming language and IO middleware provide distribution *transparency* to scientific programmers and support *efficient* message passing. Yet, only software written in SALSA can fully benefit from these features. The main contribution of this paper is the SALSA-MPI middleware that supports dynamic partitioning and load balancing for existing software for parallel adaptive partial differential equation solvers. Our middleware improves also computation fault-tolerance via data and process migration and replication. When fully developed, SALSA-MPI will provide a fully integrated software framework linking the applications layer (programmer interface) with the middleware layer, so that adaptivity and transparency can be simultaneously and efficiently achieved.

2 Programming Abstractions and Technology for Dynamic Grids

The Java [15] platform – which includes the definition of a programming language, a virtual machine, and a set of libraries providing high-level application programming interfaces (API) – is a step forward in portable distributed software engineering. In particular, Java's support for concurrent and distributed programming includes multithreading and remote method invocation APIs. Although a common perception exists that Java's main drawback is its lack of performance caused by its bytecode interpretation overhead, recent advances in JIT (Just In Time) compilation and adaptive compilation make Java a very attractive platform for scientific applications [16].

SALSA [17] is an actor-oriented programming language with high-level constructs for remote messaging, universal naming, migration, and coordination. SALSA programs are compiled into Java code, allowing a heterogeneous network of physical machines to be viewed as a homogeneous network of Java virtual machines. The WWC (World-Wide Computer) run-time architecture consists of naming servers and virtual machines running as Java applications on different Internet nodes. The virtual machines, called *theaters,* provide an environment for execution of universal actors using local resources. High-level programming language abstractions enable actors to create remote communication links with peer actors running on other WWC theaters. Furthermore, actors can easily migrate with their full state to other WWC theaters as they become available, supporting dynamic load balancing and scalability. The naming servers keep track of universal actor locators, so that communication remains transparent to actor location and migration. To enable the existing programs to take advantage of the actor features, we developed a middleware, called SALSA-MPI, that enables MPI-based programs to be viewed as actors by the SALSA/WWC computational environment.

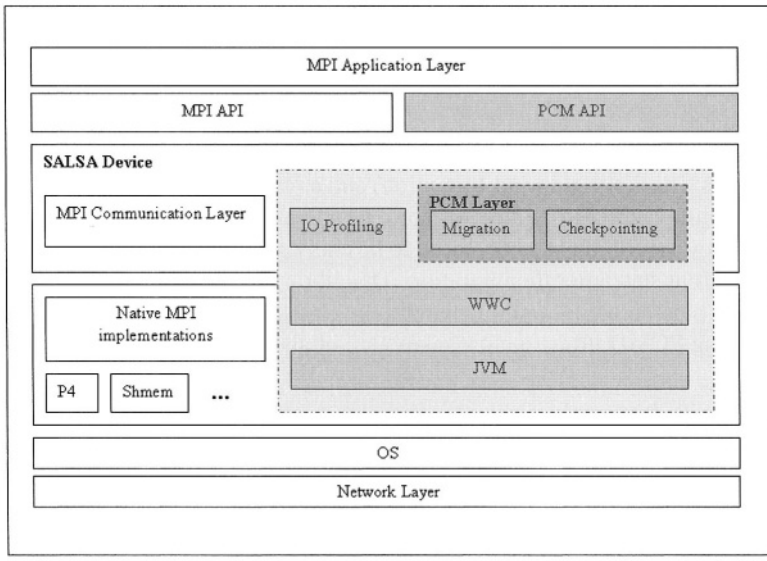


Fig. 1. SALSA-MPI Architecture.

3 SALSA-MPI

3.1 SALSA-MPI Architecture

The SALSA/IO architecture [18] consists of an actor-oriented programming language (SALSA), a distributed runtime environment (WWC), and a middleware infrastructure for autonomous load balancing (IO: Internet Operating System). This infrastructure is

highly dynamic thanks to its support for 1) resource profiling, 2) migration of actors to optimal execution environments, and 3) dynamic addition and removal of nodes in the computation. SALSA-MPI provides an infrastructure to allow the MPI processes to run on dynamic grid environments. This is achieved through the use of the IO middleware that supports dynamic reconfiguration and load balancing. Figure 1 shows the proposed SALSA-MPI architecture. The SALSA-MPI communication driver runs on top of vendor supplied MPI implementations. It uses intra-machine vendor supplied MPI implementations and inter-machine TCP communication. The SALSA device provides also an interface to SALSA/IO proxy actors, which act as profiling actors in the IO network. Every SALSA-MPI proxy-actor has a profiling actor and a decision agent. The profiling actor monitors the communication of running MPI processes and the participating nodes' memory, CPU, and network bandwidth. This profiling information is fed periodically to the decision agent. As processes join and leave the computation, the decision agent tries to reconfigure the running MPI application by migrating some of the MPI processes under its control to improve the theater overall performance. This architecture supports intra-cluster process migration and inter-cluster application migration.

Spreading MPI processes across clusters is expensive because they are usually very tightly coupled. Therefore, it is important to co-locate all the running MPI processes in a given application within one cluster. Inter-cluster application migration on the other hand, can significantly improve the performance if the current cluster experiences failures or increased load.

To support migration, we propose an application-level checkpointing API called PCM (Process Checkpointing and Migration) and a runtime system called PCMD (Process Checkpointing and Migration Daemon). Few PCM calls need to be inserted in MPI programs. They specify the data that need to be checkpointed. They also restore the process to its current state after the migration. This library is semi-transparent because the user does not have to worry about when or how checkpointing and restoration is done. The underlying PCMD infrastructure takes care of all the checkpointing and migration details.

3.2 The PCM API

The PCM API consists of a set of function calls that allow MPI programs to be dynamically reconfigurable. The PCM can be used with any iterative MPI application. The PCM library consists of set of routines with the following functionalities:

- Periodic checkpoints of MPI processes or MPI application data.
- Storage of the checkpointed data in a PCMD daemon (either locally or remotely).
- Restoration of a previously checkpointed data.
- Suspension, restart, or migration of an MPI process or an MPI application.
- Periodic probing of the status of an MPI application or an MPI process

Vadhiyar et al. have devised a similar approach through their SRS library [19]. Our approach differs from Vadhiyar's in two important features. First, our architecture allows for both process and application migration. Second, we use the SALSA-IO middleware to trigger reconfigurability and load balancing when necessary.

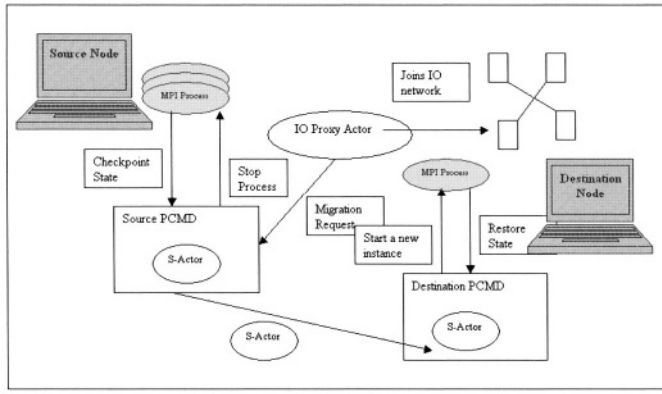


Fig. 2. Interactions among the components of the SALSA-MPI framework.

3.3 PCM Daemons

PCMD daemon needs to be launched in every computational node that joins the MPI parallel computation. The PCMD has a port number on which it listens to incoming requests. It interacts with the running MPI processes, with the IO proxy actor, and with remote PCMD daemons. When the IO decision agent chooses a process for migration, it sends a migration request to the PCMD, which creates a shadow actor (S-Actor) for the migrating process, redirects all messages sent to the migrating process (MP). The MP initiates checkpoints at the PCMD daemon. The S-Actor serializes the state of the MP and migrates to the new destination. On the remote PCMD, the S-Actor starts a new instance of the MP, stores the states of the MP, notifies the source node to stop message redirections and terminates. Once the MP is restarted, it contacts the local PCMD to restore its state. Figure 2 illustrates the interactions between the PCM Daemons and the other components of the SALSA-MPI framework.

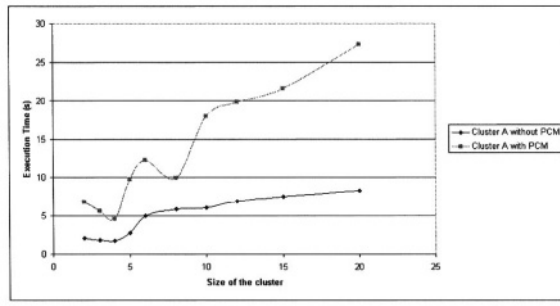


Fig. 3. The overhead that results from instrumenting the heat distribution program with the PCM API calls on cluster A.

3.4 Preliminary Results

The experimental testbed consisted of two clusters at Rensselaer Polytechnic Institute: cluster A (the Sun cluster) consisting of 20 SUN Ultra 10 machines with 256MB of memory and cluster B (the IBM Netfinity cluster) consisting of 40 900Mhz processors with 128MB of memory. Hence, the computation was distributed across two clusters where one has Intel processors and the other has Sparc processors. Our MPI application computed a numerical solution of the classical heat equation in parallel [20]. This is a typical example of an iterative parallel application that requires large volume of communication between the boundaries of the MPI processes. The original MPI code was instrumented by inserting the PCM API calls to allow application reconfiguration and checkpointing by the SALSA-MPI framework. The goal of the first experiment was to determine the overhead incurred by the PCM API. Figure 3 shows the performance of the original and instrumented MPI application running on cluster A with different numbers of nodes. The overhead introduced by the PCM library is reasonable when the number of nodes is small but it increases as the number of nodes increase. This is the result of using a single PCM Daemon in the current prototype implementation that centralizes its services. In the second experiment, the heat distribution program was first run on cluster A. We then substantially increased the load of all the nodes in this cluster by launching several long-running, computationally intensive applications. Figure 4 is based on results from repeating the same experiment, but the MPI application was migrated to cluster B once the load in cluster A increased. As expected, the performance of the MPI application improved when it migrated to a lightly loaded cluster. The performance improves only when there are fewer than 10 nodes. Again this is caused by the limitations of the current prototype implementation. These preliminary results show that reconfigurability and load balancing of parallel MPI applications can improve the performance of the application. The overhead incurred by the PCM library is offset by the application's overall improved performance. The results also show that having a decentralized architecture is necessary to achieve scalability.

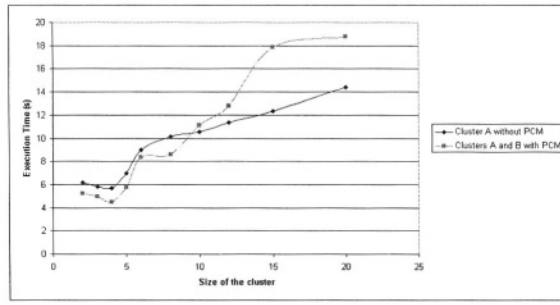


Fig. 4. MPI runs of the Heat Distribution program with and without PCM.

4 Discussion and Future Work

We have investigated programming methodologies that promote a separation of the concerns in the implementation of large scientific computations on a large network of computers. High-level programming abstractions provide a natural interface to scientists so that they can concentrate on their domain of expertise. Programming tools map these high-level abstractions into executable units that support efficient communication, dynamic partitioning and load balancing. Run-time middleware infrastructure supports adaptability of executing systems to an evolving underlying network. The presented programming paradigm, languages, and tools are a first step towards the unification of parallel and distributed computing by enabling systems to adapt to different and evolving execution environments.

Our initial target applications for SALSA-MPI are parallel adaptive scientific computations. We cannot expect scientists to rewrite or even make significant modifications to extensive libraries of C and C++ software that currently use MPI. The SALSA-MPI architecture allows application programs to run using native C/C++ code and continue to use MPI for interprocess communication. Applications that provide the minimal required checkpointing functionality can immediately take advantage of SALSA-MPI functionality for dynamic resource allocation, process migration, and fault tolerance. Large-scale computations often already provide checkpointing functionality, so in many cases, this will be a minimal burden. Scientists may choose to provide more fine-grained checkpointing to allow their applications to improve the efficiency of the dynamic migration capabilities, particularly if the computing environments being used are very dynamic.

The current prototype implementation is still a work in progress and we are working towards making the SALSA-MPI framework a fully distributed system where MPI process or application reconfiguration are triggered by the IO middleware. The IO middleware should be able to dynamically trigger fine-grain or process migration when the computation to communication ratio is high and coarse-grain or whole application migration when this ratio is low. This will allow the SALSA-MPI framework to accommodate a wide range of scientific and engineering parallel applications.

References

1. Clark, K., Flaherty, J.E., Shephard, M.S. Appl. Numer. Math., special ed. on Adaptive Methods for Partial Differential Equations **14** (1994)
2. Remacle, J.F., Karamete, B., Shephard, M.: Algorithm oriented mesh database. Proc. 9th Meshing Roundtable, New Orleans (2000)
3. Remacle, J.F., Klaas, O., Flahery, J.E., Shephard, M.S.: Parallel algorithm oriented mesh database. Eng. Comput. **18** (2002) 274–284
4. Bottasso, C.L., Flaherty, J.E., ¨ Ozturan, C., Shephard, M.S., Szymanski, B.K., Teresco, J.D., Ziantz, L.H.: The quality of partitions produced by an iterative load balancer. In Szymanski, B.K., Sinharoy, B., eds.: Proc. Third Workshop on Languages, Compilers, and Runtime Systems, Troy (1996) 265–277
5. Flaherty, J.E., Loy, R.M., Shephard, M.S., Szymanski, B.K., Teresco, J.D., Ziantz, L.H.: Adaptive local refinement with octree load-balancing for the parallel solution of threedimensional conservation laws. J. Parallel Distrib. Comput. **47** (1997) 139–152

6. Flaherty, J.E., Loy, R.M., Shephard, M.S., Teresco, J.D.: Software for the parallel adaptive solution of conservation laws by discontinuous Galerkin methods. In Cockburn, B., Karniadakis, G., Shu, S.W., eds.: Discontinous Galerkin Methods Theory, Computation and Applications. Volume 11 of Lecture Notes in Computational Science and Engineering., Berlin, Springer (2000) 113–124
7. Remacle, J.F., Flaherty, J., Shephard, M.: An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. SIAM Review **45** (2003) 53–72
8. Remacle, J.F., Shephard, M.S.: An algorithm oriented mesh database. Int. J. Numer. Meth. Engng. **58** (2003) 349–374
9. Musser, D.R., Saini, A., Stepanov, A.: STL Tutorial and Reference Guide: C++ Programming With the Standard Template Library. Addison-Wesley (1996)
10. Gropp, W., Lusk, E., Skjellum, A.: Using MPI. M. I. T. Press (1994)
11. Teresco, J.D., Beall, M.W., Flaherty, J.E., Shephard, M.S.: A hierarchical partition model for adaptive finite element computation. Comput. Methods Appl. Mech. Engrg. **184** (2000) 269–285
12. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. Lecture Notes in Computer Science **2150** (2001) 1–25
13. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
14. Szymanski, B., Varela, C., Cummings, J., Napolitano, J.: Dynamically reconfigurable scientific computing on large-scale heterogeneous grids. In R. Wyrzykowski, et. al., ed.: Proc. Parallel Processing and Applied Mathematics, Czestochowa, Poland, Springer-Verlag (2003)
15. Gosling, J., Joy, B., Steele, G.: The Java Language Specification. Addison Wesley (1996)
16. Bull, J.M., Smith, L.A., Pottage, L., Freeman, R.: Benchmarking java against c and fortran for scientific applications. In: Proceedings of ACM Java Grande/ISCOPE Conference. (2001) 97–105
17. Varela, C., Agha, G.: Programming dynamically reconfigurable open systems with SALSA. ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings **36**(2001) 20–34 http://www.cs.rpi.edu/~cvarela/oopsla2001.pdf.
18. Desell, T., ElMaghraoui, K., Varela, C.: Load balancing of autonomous actors over dynamic networks. In: To appear in Proceedings of the Hawaii International Conference On System Sciences (HICSS-37). (2004)
19. Vadhiyar, S.S., Dongarra, J.J.: Srs – a framework for developing malleable and migratable parallel applications for distributed systems (2002)
20. Wilkinson, B., Allen, M.: Parallel Programming. Prentice Hall (1998)

Deterministic Large-Scale Simulations of the Low-Dimensional Magnetic Spin Systems

Grzegorz Kamieniarz[1] and Ryszard Matysiak[2]

[1] Computational Physics Division, Institute of Physics A. Mickiewicz University, ul. Umultowska 85, 61-614 Poznań, Poland, gjk@amu.edu.pl
[2] Institute of Engineering and Computer Education, University of Zielona Góra, ul. Wojska Polskiego 69, 65-625 Zielona Góra, Poland, r.matysiak@eti.uz.zgora.pl

Abstract. Two types of deterministic simulation techniques are presented and applied to a class of the real low-dimensional magnetic systems described by the Heisenberg hamiltonian which includes the macroscopic Haldane-gap and molecular-based spin $S = 1$ chains, small size magnetic clusters embedded in some supramolecules and other compounds. Using both the exact diagonalization and the transfer-matrix techniques, the spin degrees of freedom are accurately taken into account, yielding the energy spectra and the thermodynamical functions. It is shown that our modelling tool based on our own codes and recent parallelization/vectorization mechanisms is reliable and leads to a quntitative agreement with experiment. We review the results for some isotropic and anisotropic systems, for systems with uniform and non-uniform interactions and those with antisymmetric interactions.

1 Introduction

The low-dimensional spin systems have received much attention in recent years. Here we consider the supramolecular clusters and a number of the macroscopic quasi-one-dimensional magnets.

An example of a large metal ion cluster is $[\mathrm{Mn(hfac)_2NITPh}]_6$. The manganese ions with $S = 5/2$ are connected by organic radicals NITPh with unpaired electron $S = 1/2$. The two types of spins are antiferromagnetically coupled in such a way that in the ground state $S = 12$. Ground states with such large spin values are rare. The structure of this cluster is shown in Fig. 1, where the Mn ions are presented by six large circles.

The other examples of compounds which have high ground spin states are manganese, iron, chromium, vanadium and nickel clusters [1,2]. This family is very rich. There are clusters containing 12 manganese ions $[\mathrm{Mn_{12}O_{12}(carboxylato)_{16}}]$ [3] which form the external ring consisting of eight Mn ions with $S = 2$ and the internal tetrahedron consisting of four Mn ions with $S = 3/2$ (the ground state is $S = 10$). For the planar cluster $[\mathrm{Fe_8O_2(OH)_{12}(tacn)_6}]^{8+}$, six spins are up and two are down (the ground state is $S = 10$) [4].

In the case of clusters with the ground state $S = 0$, the magnetization data at low temperatures provide additional information on the energies of the lowest

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1091–1098, 2004.
© Springer-Verlag Berlin Heidelberg 2004

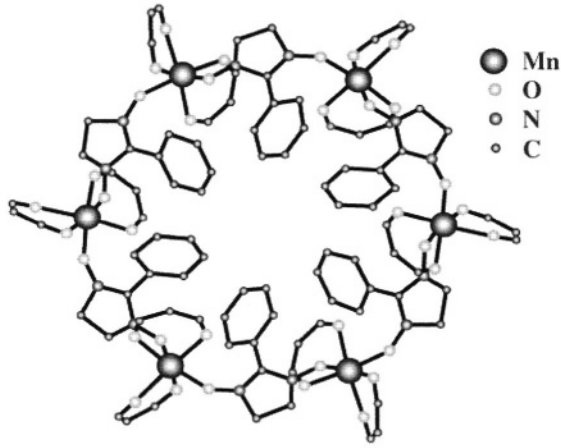


Fig. 1. The structure of Mn_6 cluster. The six largest circles present Mn ions. The ground state can be described as having all the manganese spins up and the radical spins down.

excited levels. At low magnetic field, the magnetization is zero, in agreement with an $S = 0$ ground state, but above some larger value of magnetic field, it rapidly increases, reaching the value $2\mu_B$, appropriate for an $S = 1$ spin. As the field increased, other similar steps are observed, with plateaus at $4\mu_B, 6\mu_B, ...$, indicating that states with total spins $S = 2,3,\ldots$ successively become the ground states [4].

Another class of compounds is that of one-dimensional systems described by the spin $S = 1/2$ and $S = 1$ antiferromagnetic Heisenberg model. For these systems the Haldane conjecture holds and predicts the difference between the ground state of integer and half-integer spins. Development of molecular magnetism has given rise to an increase in the number and in the types of one-dimensional magnetic materials and stimulated research in this field. An interesting aspect of these compounds is that the intrinsic low symmmetry of the building blocks easily affords uniform [5] and non-uniform chains [7].

As to the uniform $S = 1/2$ antiferromagnetic Heisenberg chains, the spin-Peierls compound $CuGeO_3$ and the rare-earth pnictide compound Yb_4As_3 [8] deserve special attention.

In this article we present mainly results of accurate numerical simulations based on the quantum transfer-matrix (QTM) technique and we compare them with known experimental results. We show that the domain of applications of the QTM method is very wide. We also use the exact diagonalization technique exploiting the point-group symmetry and the properties of a shift operator. The description of this method is presented elsewhere [6].

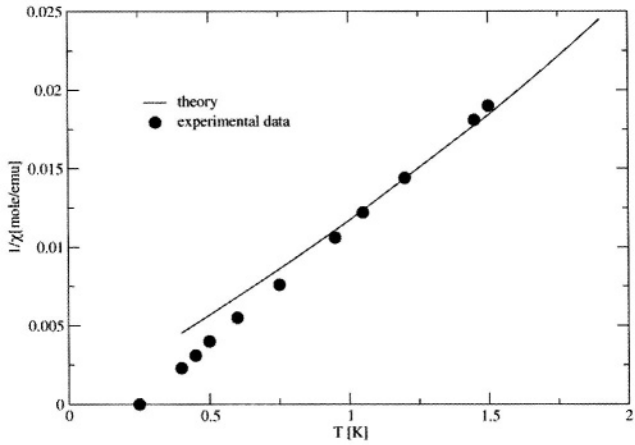


Fig. 2. The behaviour of the inverse of susceptibility for the molecule Ni_{12} as a function of the temperature T. The experimental and theoretical data are plotted by the full circles and the continuous line, respectively.

2 Methodology and Description of Simulations

The advantages of the quantum transfer-matrix simulation method have been demonstrated for the macroscopic Haldane-gap [5,9] and molecular-based [7] magnetic chains. The results are not subject to any statistical nor systematic errors and the macroscopic limit can be directly evaluated from the largest eigenvalue of the transfer matrix. For the finite rings however, this is not the case and all terms in the definition of the partition function bring some contribution, so that the computational complexity of the QTM method increases enormously.

The features of many magnetic materials may be quite accurately represented by Heisenberg model described by the Hamiltonian:

$$\mathcal{H} = -J \sum_{i=1}^{N} (\mathbf{S}_i \cdot \mathbf{S}_{i+1}) - D \sum_{i=1}^{N} (\mathcal{S}_i^z)^2 - g_\nu \mu_B B \sum_{i=1}^{N} \mathcal{S}_i^\nu , \qquad (1)$$

where $\mathbf{S}_i$ is interpreted as the spin located at the i-th site of a one-dimensional lattice of N equally spaced sites. J denotes the nearest neighbor exchange integral (negative for the antiferromagnetic coupling), B is the external magnetic field which can be applied along the chain $(\nu = z)$ or in the perpendicular direction $(\nu = x, y)$, g_ν is the corresponding gyromagnetic ratio and N is the size of a given one-dimensional system (the chain or the ring).

In the framework of QTM [9], the partition function is calculated from the expression

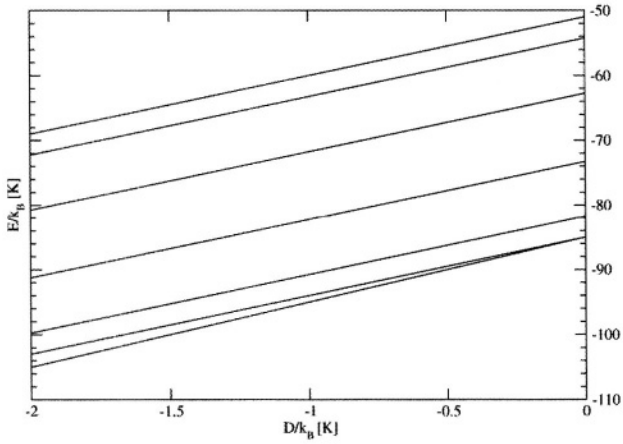


Fig. 3. Energy levels of the $S^z = 10$ and $^z = 9$ states as a function of the anisotropy D for the $N = 10$ ring.

$$\mathcal{Z} = \lim_{m\to\infty} \mathcal{Z}_m = \lim_{m\to\infty} Tr \left[\prod_{i=1}^{N/2} \mathcal{V}_{2i-1,2i} \prod_{i=1}^{N/2} \mathcal{V}_{2i,2i+1} \right]^m , \mathcal{V}_{i,i+1} = e^{-\beta \mathcal{H}_{i,i+1}/m}. \tag{2}$$

We define also a unitary shift operator $\mathcal{D}$, acting in the space $\mathcal{H}^N$ which is a direct product of N single-spin spaces $\mathcal{H}_i$

$$\mathcal{D} \equiv \sum_{S_1^z} \dots \sum_{S_N^z} | S_3^z \dots S_N^z S_1^z S_2^z \rangle \langle S_1^z S_2^z S_3^z \dots S_N^z | , \tag{3}$$

where S_i^z is the eigenvalue of the z component of the spin operator $\mathcal{S}_i$. Using Eq. (3), we may express m-th classical approximant of the partition function (2)

$$\mathcal{Z}_m = Tr \left(\mathcal{W}_1 \mathcal{W}_2 \right)^m , \tag{4}$$

where $\mathcal{W}_i = \left(\mathcal{V}_{i,i+1} \mathcal{D}^+ \right)^{N/2}$, $(i = 1, 2)$.

For the infinite chains (the macroscopic limit) we may reverse the transfer direction and to calculate the partition function from the largest eigenvalue of the transfer matrix [9]. Then the m-th classical approximant to the partition function of Eq. (1) can be written in the following form:

$$\mathcal{Z}_m = Tr \left(\mathcal{W}_1 \mathcal{W}_2 \right)^{N/2} . \tag{5}$$

In the limit $N \to \infty$ the partition function $\mathcal{Z}$ is equal to the highest eigenvalue of the global transfer matrix $\mathcal{W} = \mathcal{W}_1 \mathcal{W}_2$.

The QTM technique can be applied not only to the model (1) with both the uniform and non-uniform spin variables, but also to the uniform and non-uniform exchange interaction constants. In the case of the molecular-based chains with $S = 1$ the model should take into account the alternation of the exchange coupling [7].

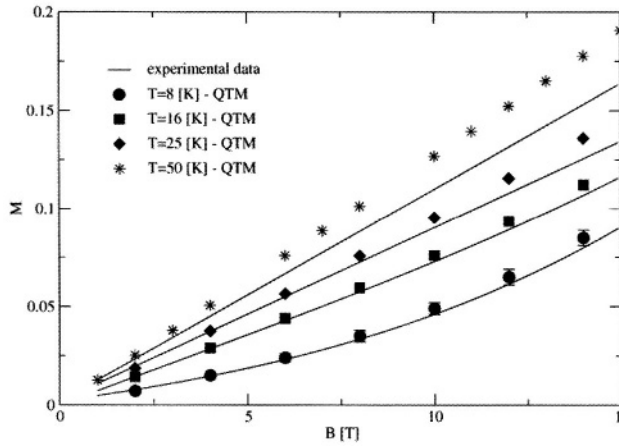


Fig. 4. The magnetization profiles per site and μ_B for NENP against the external field. The QTM data are ilustarted by the corresponding symbols and the experimental data are drawn by the continuous line.

3 Physical Applications and Simulation Results for Some Low-Dimensional Magnetic Systems

First, the exact diagonalization is applied to calculate the energy spectra and the inverse susceptibility of the Ni_{12} cluster with uniform spin variable $S = 1$. In Fig. 2 we present the latter in the low temperature, where our numerical estimates ($J/k_B = 8.5$ K, $D = 0$ and $g = 2.13$) are drawn by the full line and the corresponding experimental data [2] - by the symbols. The anisotropy dependence of the low-lying energy levels is shown for $N = 10$ in Fig. 3. The ground state corresponds to $S^z = 10$ whereas the excited states - to $S^z = 9$.

Our QTM simulation method has been applied to verify some numerical results obtained formerly for the $S = 1$ one-dimensional Heisenberg model. Our results were also compared with the experimental results for compounds with Haldane phase [5]. Representative physical realization of the Haldane-gap spin $S = 1$ system with moderate anisotropy is $Ni(C_2H_8N_2)_2NO_2ClO_4$, abbreviated as NENP [10,5]. An example of our calculations performed for NENP, which can be described by the microscopic model parameters $D/k_B = 7.8$ K, $g_\perp = 2.25$ $g_\parallel = 2.20$ and $J/k_B = 48$ K, is given in Fig. 4.

Refering to the uniform and non-uniform molecular-based chains, reported in literature [11,12,13,14], we draw in Fig. 5 the temperature dependence of the specific heat for a compound with alternating interactions considered by Ribas *et al* [14] for the compound $Ni_2(EDTA) \cdot 6H_2O$.

We have performed calculation of the specific heat for the parameters $J_1 = -10.6 \pm 1.0$ K, $J_2 = -3.3 \pm 0.5$ K, $g = 2.20 \pm 0.02$, $D = 5 \pm 1.0$ K, where the latter value was found from analysis of the zero-field susceptibility. The measured

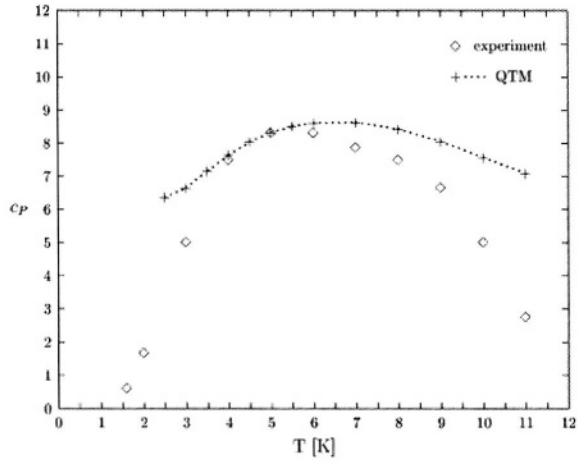


Fig. 5. The specific heat of $Ni_2(EDTA)\cdot 6H_2O$. The measured values are plotted with the diamonds and the best QTM fit is drawn with the dotted line.

values are plotted with the diamonds and the best QTM fit is drawn with the dotted line.

From the category of one-dimensional systems comprising spin $S = 1/2$ chains, we select here the compound Yb_4As_3 . The neutron scattering experiments on Yb_4As_3 actually confirmed that the excitation spectrum is well described by the one-dimensional $S = 1/2$ isotropic Heisenberg model [16] in the absence of magnetic field. Under magnetic field, the gap in the spin excitation spectrum of Yb_4As_3 opens and fulfils the power law $H^{2/3}$ [15] which gives the strong experimental evidence for the existence of a staggered field alternating along the Yb^{3+} chains induced by the Dzyaloshinsky–Moriya interaction.

Our simulations for Yb_4As_3 are performed with the fixed parameters [15, 16]. We have calculated the specific heat of the $S = 1/2$ spin chain both in the absence of the external magnetic field and in the finite external magnetic field perpendicular and parallel to the spin chain ($B = 4 - 25$ [T]). In high temperatures (above 7 [K]) we have performed our simulations for the Trotter number up to $m = 10$ and in low temperatures up to $m = 12$. The specific heat results determined in the magnetic fields are presented in Fig. 6. The open symbols represent experimental results for a polydomain sample with the magnetic field ($B = 4$, 8, 12, 16, 19.5 [T]) applied along the cubic $\langle 111 \rangle$ direction, and the filled symbols are numerical results. For the experimental data the phonon contribution $C_{ph} = 2.05 \cdot 10^{-3}$ [J/(K^{4}mol)] $\cdot\, T^3$ has been subtracted [17]. With increasing magnetic field the molar specific heat maximum *C/T* increases, shifts to the right and the curves intersect at about 9 [K] which is consistent with the new experimental findings. The deviations between our results and experimental data are more likely due to the model deficiency than the accuracy of our simulations.

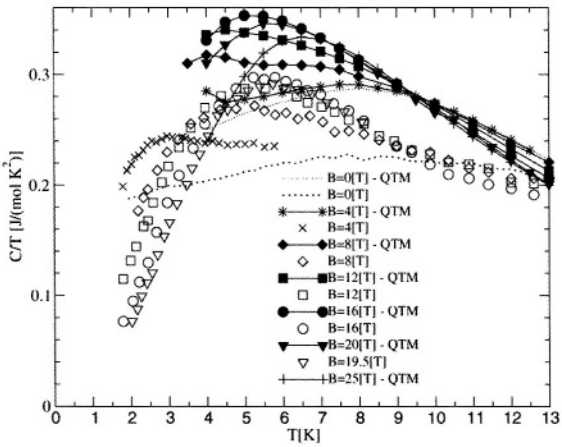


Fig. 6. Comparison of the measured field-dependent magnetic specific heat and QTM calculation for Yb_4As_3 up to $B = 25$ T. The dotted curves represent the zero-field specific heat.

4 Conclusions

We have worked out quantum transfer-matrix approach and exact diagonalization to characterize the finite temperature magnetic properties of the high nuclearity cyclic spin clusters with large and alternating spins and a number of the macroscopic quasi-one-dimensional magnets. We have carried out large-scale computations obtaining high resolution data for the low-dimensional systems down to low temperatures and in the wide range of single-ion anisotropy parameters and external magnetic field.

The computations were carried out on CRAY T3E and SGI Origin 3800 supercomputers using Parallel Virtual Machine (PVM) and Message Parallel Interface (MPI) system libraries, respectively. The computational complexity of our problems is exponential. For the $S = 1/2$ systems, the memory complexity is roughly $S(m) \approx 4^{m-3}$ [kB] and the temporal complexity is of the same order with respect to m. On currently used architectures one could reach $m = 16$ whereas for our purposes $m \leq 12$ was enough. In general, for the quantum chain simulations we encounter both the memory and CPU bound for $k_BT/J \leq 0.1$. For the finite ring simulations, we are limited by the CPU time resources.

Acknowledgements. This work was partially supported by the Committee for Scientific Research via the grant 4 T11F 014 24. Numerical simulations were performed in the Poznań Supercomputing and Networking Center. We would like also to thank Prof. A. Caramico D'Auria and Prof. F. Esposito for some discussions and Mrs M. Haglauer for her technical assistance.

References

1. Gatteschi, D., Sessoli, R., Cornia, A.: J. Chem. Soc., Chem. Commun. (2000) 725.
2. Andres, H., Basler, R., Blake, A. J., Cadiou, C., Chaboussant, G., Grant, C. M., Güdel, H.-U., Murrie, M., Parsons, S., Paulsen, C., Winpenny, R. E. P.: Chem. Eur. J. 8, No. 21 (2002).
3. Caneschi, A., Gatteschi, D. ,Sangregorio, C. , Sessoli, R., Sorace, L., Cornia, A., Novak, M. A., Paulsen, C., Wernsdorfer, W.: J. Magn. Magn. Mat. **200** (1999) 182.
4. Gatteschi, D., Caneschi, A., Pardi L., Sessoli, R.: Science **265** (1994) 1054.
5. Kamieniarz, G., Matysiak, R., Caramico D'Auria, A., Esposito, F., Esposito, U.: Phys. Rev. **B56** (1997) 645.
6. Kamieniarz, G., Matysiak, R., Caramico D'Auria, A., Esposito, F., Benelli, C,: Eur. Phys. J. B **23** (2001) 183.
7. Caramico D'Auria, A., Esposito, U., Esposito, F., Gatteschi, D., Kamieniarz, G., Wałcerz, S.: J. Chem. Phys. **109** (1998) 1613.
8. Schmidt, B., Aoki, H., Cichorek, T., Custers, J., Gegenwart, P., Kohgi, M., Lang, M., Langhammer, C., Ochiai, A., Paschen, S., Steglich, F., Suzuki, T., Thalmeier, P., Wand B., Yaresko, A.: Physica B **300** (2001) 121.
9. Kamieniarz, G., Matysiak R.: Comput. Mat. Science **28** (2003) 353.
10. Asano, T., Ajiro, Y., Mutka, M., Yamazaki, H., Hosoito, N., Shinjo T., Kikuchi, H.: Solid State Comm. **90** (1994) 125.
11. Escuer, A., Vicente, R., Ribas, J., Salah El Fallah, M., Solans, X., Font-Baria, M.: Inorg. Chem. **33** (1994) 1842.
12. Ribas, J., Monfort, M., Diaz, C., Bastos, C., Mer and, C., Solans, X., Font-Baria, M.: Inorg. Chem. **34** (1995) 4986.
13. Escuer, A., Vicente, Salah El Fallah, M., Solans, X., Font-Baria, M.: J. Chem. Soc., Dalton Trans. (1996) 1013.
14. Ribas, J., Monfort, M., Gosh, B. K., Solans, X., Font-Baria, M.: J. Chem. Soc., Chem. Comm. (1995) 2375.
15. Kohgi, M., Iwasa, K., Mignot, J.-M., Fak, B., Gegenwart, P., Lang, M., Ochiai, A., Aoki, H., Suzuki, T.: Phys. Rev. Lett. **86** (2001) 2439.
16. Kohgi, M., Iwasa, K., Mignot, J.-M., Ochiai, A., Suzuki, T.: Phys. Rev. **B56** (1997) R11388.
17. Gegenwart, P., Aoki, H., Cichorek, T., Custers, J., Harrison, N., Jaime, M., Lang, M., Ochiai, A., Steglich, F.: Physica B **312-313** (2002) 315.

Distributed File System for Clusters and Grids*

Olivier Valentin, Pierre Lombard, Adrien Lebre, Christian Guinet, and Yves Denneulin

Laboratoire Informatique et Distribution-IMAG
51 avenue J. Kuntzmann, 38 330 Montbonnot Saint-Martin, France
Olivier.valentin@imag.fr

Abstract. NFSG aims at providing a solution for file accesses within a cluster of clusters. Criteria of easiness (installation, administration, usage) but also efficiency as well as a minimal hardware and software intrusivity have led our developments. By using several facilities such as distributed file systems (NFSP) and a high-performance data transfer utility (GXfer), we hope to offer a software architecture fully compatible with the ubiquitous NFS protocol. Thanks to a distributed storage (especially multiple I/O servers provided by NFSP), several parallel streams may be used when copying a file from one cluster to another within a same grid. This technique improves data transfers by connecting distributed file system at both ends. The GXfer component implements this functionality. Thus, performances only reachable with dedicated and expensive hardware may be achieved.

1 Introduction

Current trends in High Performance Computing have been characterized by an evolution from the super computing towards cluster computing for several years [1], thanks to an ever-increasing performance/price ratio. As clusters have started to appears in several different places, be it two rooms in a same institute or faraway countries, aggregating the large power of all those newly-born 'poor man's super-computer' has been the source of lots of works (one of the most famous being Globus, which became OGSA project a few years ago).

Such environments have some drawbacks inherent to their qualities : as they offer as lot of services (first-grade authentications, job management, reservations, ldots), they tend to become quite heavy and complex to use. Yet, all those functionalities are not always required to run in dedicated and trusted architectures based on VPN networks (see the French VTHD project[1]). Clusters evolving in such an architecture, that is clusters linked by means of high-performance links (several gigabit/s), constitutes a 'cluster of clusters' which somewhat heterogeneous characteristics (such as OS, libraries, ...). Thus, to have a useful system,

* This work is supported by APACHE which is a joint project funded by CNRS, INPG, INRIA and UJF. GXfer is a software component developed for the RNTL E-Toile (http://www.urec.cnrs.fr/etoile/).

[1] See http://www.vthd.org/

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1099–1104, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the requirement of easy installation, easy maintainability and adaptability to commodity hardware appeared soon at the conception phase.

Hence, to summarize the features and characteristics we wanted: a common file tree, shared by all the machines within a grid; a minimal access time to data; working efficiently on commodity hardware; aggregation of the unused disk space of clusters; data availability for all the nodes of a cluster; reading/writing of data allowed; NFS protocol [2] and coherency (temporal coherency). To achieve these aims, we have used two tools developed within our team : the first one being a distributed version of the ubiquitous NFS server for clusters, `NFSP` [3,4], the second one being an inter-cluster transfer tool, GXfer, developed to use efficiently large network pipes without requiring expensive storage systems (SAN based for instance).

After this introductory section, the related works will be shown in section 2. Then, the `NFSG` principles are shown in section 3, followed by a short evaluation of expected results in section 4. Eventually, a conclusion will give hints about future extensions.

2 Related Works

A lot of work has been carried out in the file system field, yet the issues of scalability and data sharing within a grid still is a moot point. Within a local site (LAN for instance), the most prominent is most likely to be NFS [2,5] in the Beowulf world, but cannot solve the constraints of the WAN architectures[2]. Unlike this latter system, the family of systems constituted by AFS [6], CODA [7] try to address certain issues but ignore some others (consistency, high availability, ...) Yet, none of the existing solutions seems adapted to high-performance computing, which often implies the setup of large and expensive machines *à la* GPFS [8] or more complex and intrusive solutions as what may be seen in the rising Lustre [9]. Setting an open, adaptable and efficient solution is still open to developments. Most of the current works consists in optimizing systems so as to provide better performances within cluster-like architectures. The Globus project [10] and its newer versions (OGSA) tackles grid aspects. In a similar way, our works care about the same constraints but with a released level with regards to the security and try to remain compatible with the established standards.

3 The NFSG Proposal

`NFSG` is designed to be a grid file-system. But when we say 'grid', we mean 'cluster of clusters'. In fact, this system should match the needs of several institutions federated into one grid. We think that this strong structure should be taken into account, and thus, that having a two-level system might be a good approach : at the cluster level, a file system that would serve files to local clients, and at the grid level a system to federate the lower level file systems.

[2] NFS4 aims at addressing some of these issues but is still not as widely spread as NFS2 and NFS3.

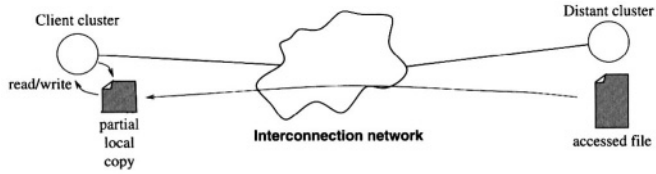


Fig. 1. Local caching strategy

The caching strategy shown in figure 1 will allow the clients of a cluster to access data stored on another cluster of the grid. Data accessed will be first copied locally and then served by the local file system (here, by 'local', we mean 'on the local cluster'). Thus, reads and writes should be more efficient than remote ones, across the interconnection network. But this has several consequences: first, we need a means to efficiently copy data from one cluster to another. As the bandwidth available between two clusters cannot be filled by one node (each node has a bandwidth of 100Mbit/s), data has to be distributed among the clients. To achieve this, we used the `NFSP` [3] cluster file system since it fits our needs such as a seamless integration in the nodes.

`NFSG` itself is represented by one node on each cluster of the grid. These nodes are responsible for locating data on the grid, asking to transfer it locally and then manage the local copies. For instance, when this latter detects a write access on a local copy of a file, it sends the modifications in order to update the master file. It also checks that a local copy is fresh enough when a client reads the file; if it is not the case then the file is updated to have fresh enough data again.

3.1 Typical Use Case

The figure 2 shows a typical use case of `NFSG`. A user of the system (the client) has some data stored on his/her cluster and needs to process it on several other clusters. The user launches jobs on those clusters, and these jobs try try to access input data (as the file hierarchy is unique, the file paths are identical everywhere). For each request of a client and on each cluster, the local file system tells `NFSG` that there is an access to the corresponding data. Whenever data are not available, `NFSG` goes into in a 'page-fault handling' state and starts to retrieve missing them. Once they are available, the computing nodes get local access to the copy (see figure 2.a.) Then the computing nodes write their results using the local file system (see figure 2.b). Eventually, when the job is finished, the data produced locally, that is on the user's cluster, will be fetched on-the-fly to the 'master' cluster by using a similar 'page-fault handling' mechanism (see figure 2.c).

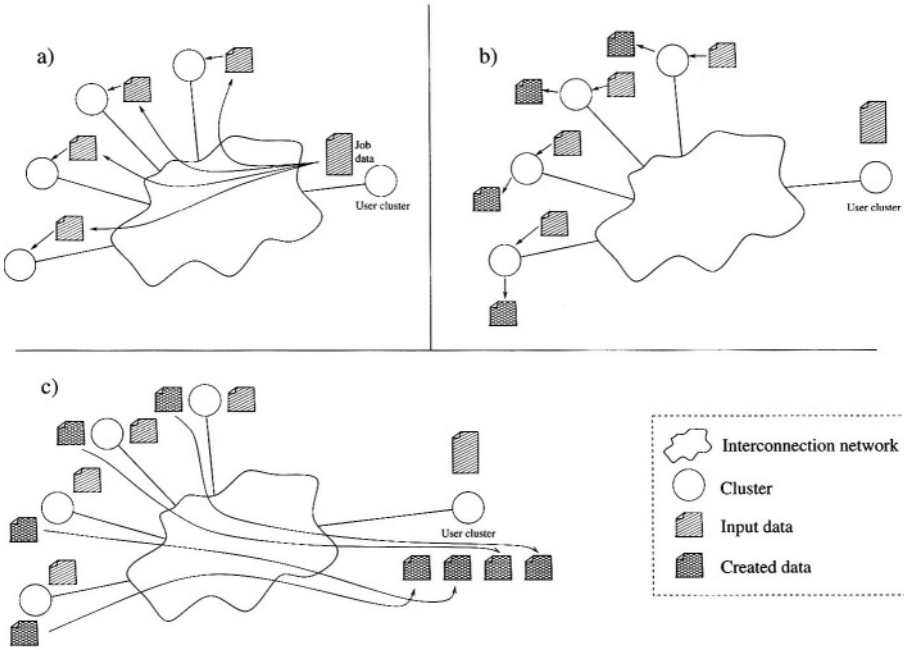


Fig. 2. Typical data usage in NFSG

3.2 A Shared File Hierarchy

In NFSG, each cluster of a grid has the same view of the file tree. As shown in the figure 2, the typical application stores files locally, then processes them on other clusters. This behavior leads to the concept of cluster owner for files and directories: for each of these items, this peculiar cluster is called 'master'. For a cluster, being the owner of:

- a file means that it owns the necessary information to find the data of this file on the local file system or on another cluster.
- a directory means that it owns the files that are in this directory and has pointers to sub-directories.

The figure 3 gives an example of an NFSG file tree. For instance, serv2 is the master of dir2, dir4, dir5, dir6 and file2.

When the NFSG system starts, it is composed of only one cluster. This cluster will be called the root cluster as it will own the '/', that is the root directory, which is serv0 in the example figure 3. Then, when some other clusters join the system, they learn who has become the root when they register.

While resolving to the location of file5, the lookup will have to go through the tree from the root directory to the file itself. First, the owner of the root directory (serv0) is asked to resolve /dir2/dir5/dir7/file5. serv0 answers that it is not owner of dir2 but knows that it is serv2. Then serv2 is contacted and it

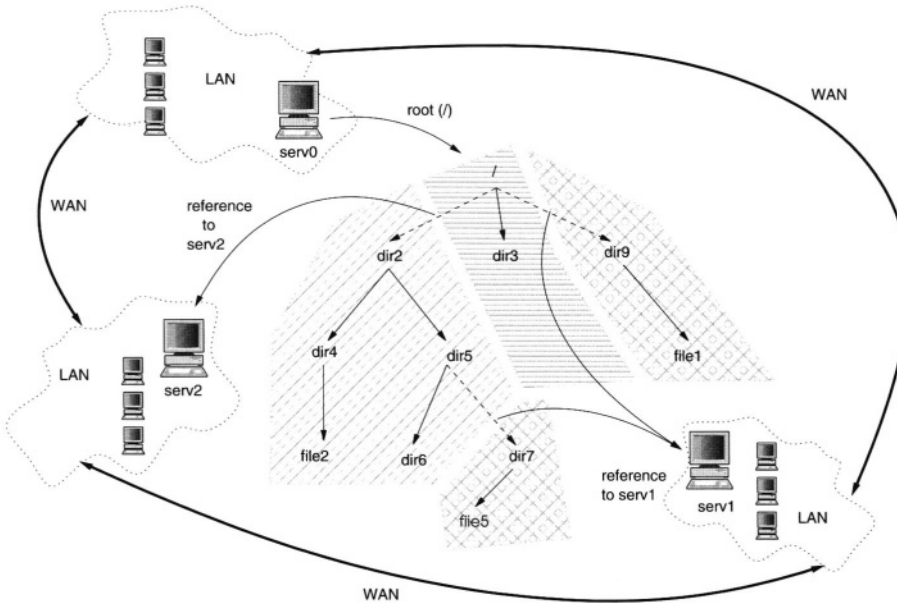


Fig. 3. NFSG file hierarchy

resolves up to dir7 on serv1. Eventually, we find the owner of file5, that is serv1. Now, serv1 will be able to locate the data of the requested file.

This system has some good properties like atomic operations on the file tree. Indeed, the creation or deletion of files or directories are managed by the owner of the parent directory.

After each complete lookup, the `NFSG` servers keep some direct pointers to the owner of a file in order to speed-up the look-up and to avoid the flooding of the root server. This also makes `NFSG` able to keep running even if a temporary failure occurs.

It is important to emphasize that being the owner of a file does not mean 'having the data' but more 'knowing where the data is'. Thus, even though a directory has lots of large files, the cluster owning this directory does not have to store all the files, as they may be stored on several other clusters. This also means that the ownership of a file may be moved from a cluster to another having to move data. Furthermore, such an operation is possible with directories.

4 Early Results

The software is currently in a work-in-progress state and is not fully functional yet. The meta-data handling : update scheduling, synchronizations, … is complete but the integration with the lower level components, `NFSP` and `GXfer` has still to be completed. Some preliminary performance evaluations have been done by assessing how each component performed. Since the proposed architecture is

really lightweight, we expect the performance reachable with the fully-functional system to be really near by those of the individual components.

The NFSP performances are illustrated in [4]. As for GXfer performances [11], this software layer lets a gigabit/s network pipe be saturated by files transfer between two 100Mb/s-based clusters, both using a NFSP storage layer.

5 Conclusion

The NFSG project is still in its infancy and will probably evolve much before becoming a fully functional and reliable distributed file system for clusters of clusters. Of course there are some issues left to tackle, such as the behavior of NFSG in the case of network failures. The developed prototype still have to be deployed on a large scale and tested against real world applications. When fully functional, it will be a lightweight and high performance file system for clusters of clusters.

Further investigations have to be carried out with the migration of the owner rights between clusters, to enable a better level of load balancing among the server nodes. Another future extension may also consists in handling the transfer of only some parts of the files. Such an approach would help saving some bandwidth and hence, gain some more performances.

References

1. Sterling, T., Savarese, D., Becker, D.J., Dorband, J.E., Ranawake, U.A., Packer, C.V.: BEOWULF: A parallel workstation for scientific computation. In: Proceedings of the 24th International Conference on Parallel Processing. (1995)
2. IETF: NFS: Network file system specification. RFC1094 (1989)
3. Lombard, P., Denneulin, Y.: nfsp: A Distributed NFS Server for Clusters of Workstations. In: 16'th IPDPS, IEEE (2002)
4. Lombard, P., Denneulin, Y., Valentin, O., Lebre, A.: Improving the Performances of a Distributed NFS Implementation. In: Proceedings of PPAM'2003. Lecture Notes in Computer Science, Springer-Verlag (2003)
5. Pawlowski, B., Juszczak, C., Staubach, P., Smith, C., Lebel, D., Hitz, D.: NFS version 3, design and implementation. In: Proceedings of the USENIX Summer 1994 Conference. (1994)
6. IBM: OpenAFS Web Site. http://www.openafs.org/ (2003)
7. CMU: Coda File System Web Site. http://www.coda.cs.cmu.edu/ (2003)
8. Schmuck, F., Haskin, R.: GPFS: A Shared-Disk File System for Large Computing Clusters. In: Proceedings of the First Conference on File and Storage Technologies (FAST). (2002)
9. Schwan, P.: Lustre : Building a file system for 1,000-node clusters. In: Proceedings of the Linux Symposium, Ottawa. (2003)
10. Foster, I., Kesselman, C., eds.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann (1999)
11. Denneulin, Y., Guinet, C., Lombard, P.: A Parallel Data Transfer Solution for Grids. In: Slides presented at the Global Grid Forum (GGF8), Seattle, WA, USA (2003)

New Adaptive GMRES(m) Method with Choosing Suitable Restart Cycle m

Kentaro Moriya[1] and Takashi Nodera[2]

[1] Aoyama Gakuin University,
O519 5-10-1 Fuchinobe, Sagamihara, Kanagawa 229-8558, Japan.
[2] Keio University,
3-14-1 Hiyoshi, Kohoku, Yokohama 223,Japan.

Abstract. GMRES method is one of the major iterative algorithms for solving large and sparse linear systems of equations. However, it is difficult to implement GMRES algorithm because its storatege and computation cost are so exceeded. Therefore, GMRES(m) algorithm is often used. In this paper, we propose a new variant of GMRES(m) algorithm. Our algorithm chooses the restart cycle m based both on the convergence test of residual norm and on the distribution of zeros of residual polynomial of GMRES(m) algorithm. From the numerical examples on Compaq Beowulf, we also show the effectiveness of our proposed algorithm.

1 Introduction

We study the linear systems of equations

$$A\boldsymbol{x} = \boldsymbol{b}, \quad A \in R^{n\times n}, \quad \boldsymbol{x},\ \boldsymbol{b} \in R^n \tag{1}$$

where the coefficient matrix A is large, sparse and nonsingular. The systems (1) are occasionally solved by iterative algorithms.

GMRES algorithm [2] uses Arnoldi process [1] and the exact solution can be obtained in at most n iteration steps. However, the computation cost and storatege for orthogonalization grow exceedingly high as the number of iteration steps increases. GMRES(m) algorithm, performing the restart at every m iteration steps, is often used to overcome this drawback. Unfortunately, it's difficult to choose suitable m so that it avoids both the stagnation of the convergence of residual norm and the excessive cost of orthogonalization.

On the other hand, recently Tsuno et al. [5] proposed an algorithm varying the restart cycle m adaptively during the iteration steps. Their algorithm performs the restart when the distribution of zeros of residual polynomial of GMRES(m) algorithm becomes ideal. Moreover, we study a parameter for the convergence test of residual norm and combine this parameter with Tsuno et al. [5]'s restarting algorithm. The major purpose of our algorithm is to choose the restart cycle m based on the condition that involves the parameter for the convergence test of residual norm and the distribution of zeros of residual polynomial of GMRES(m) algorithm.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1105–1113, 2004.
© Springer-Verlag Berlin Heidelberg 2004

GMRES algorithm and GMRES(m) algorithm are introduced briefly in Section 2. We propose a new variant of GMRES(m) algorithm and show how to choose the restart cycle m in Section 3. Two numerical examples from the MIMD parallel computer Compaq Beowulf are reported in Section 4 and conclusions are given in Section 5.

2 The Conventional Algorithms

In this section, GMRES algorithm and GMRES(m) algorithm [2] are introduced briefly by means of their residual polynomials.

2.1 The Residual Polynomial of GMRES Algorithm

We consider the residual polynomial of GMRES algorithm

$$\Psi_{\ell}^{exact}(t) = \Pi_{j=0}^{\ell}(1 - t/\lambda_j^{exact}) \tag{2}$$

where λ_j^{exact} are zeros of the polynomial (2). By using the polynomial (2), the ℓ-th residual vector can be described as

$$\boldsymbol{r}_{\ell} = \Psi_{\ell}^{exact}(A)\boldsymbol{r}_0 \tag{3}$$

where $\boldsymbol{r}_0$ is the initial residual vector. However, it's more difficult to compute λ_j^{exact} as the dimension of coefficient matrix A gets larger.

2.2 The Residual Polynomial of GMRES(m) Algorithm

GMRES(m) algorithm is an alternative to GMRES algorithm. We assume that the restart has been performed i times and k iteration steps have been done after the i-th restart. We also characterize the j-th restart cycle as m_j because each restart cycle is varied adaptively. We consider the residual polynomials of GMRES(m) algorithm,

$$\Psi_k^{(i+1)}(t) = \Pi_{s_1=1}^{k}(1 - t/\lambda_{s_1}^{(i+1)}), \quad (1 \le k \le m_{max}) \tag{4}$$

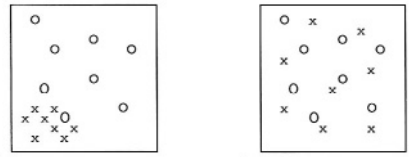
$$\Psi_{m_j}^{(j)}(t) = \Pi_{s_2=1}^{m_j}(1 - t/\lambda_{s_2}^{(j)}), \quad (1 \le m_j \le m_{max}),\ (j = 1, 2, \ldots, i) \tag{5}$$

where m_{max} is the maximum value of m_j and k. While $\Psi_{m_j}^{(j)}(t)$ is the residual polynomial used before the i-th restart, $\Psi_k^{(i+1)}(t)$ is the residual polynomial used after the i-th restart. So the ℓ-th ($= \Sigma_{j=1}^{i} m_j + k$) residual vector $\boldsymbol{r}_{\ell}$ can be described as

$$\boldsymbol{r}_{\ell} = \Psi_k^{(i+1)}(A)\{\Pi_{j=1}^{i}\Psi_{m_j}^{(j)}(A)\boldsymbol{r}_0\} \tag{6}$$

where $\lambda_{s_1}^{(i+1)}$ and $\lambda_{s_2}^{(j)}$ are zeros of residual polynomial (4) and (5), respectively.

In the following sections, whereas we call λ_j^{exact} "the ideal zeros," we call $\lambda_{s_1}^{(i+1)}$ and $\lambda_{s_2}^{(j)}$ "the approximate zeros."



(a) Bad example (b) Good example

Fig. 1. The distributions of zeros of residual polynomials (○: The ideal zeros, x: The approximate zeros)

3 How to Choose the Restart Cycle

We propose a new variant of GMRES(m) algorithm, choosing m adaptively during the iteration steps. The restart is performed based on the following two conditions, the distribution of the approximate zeros and the convergence test of residual norm.

3.1 The Condition for the Distribution of the Approximate Zeros

By choosing m based on the distribution of the approximate zeros, Tsuno et al. [5] improves the performance of GMRES(m) algorithm. Their proposed algorithm is called GMRES($\leq m_{max}$) algorithm, where m_{max} is the maximum value of the restart cycle. We assume that the distribution of the ideal zeros "○" is given like Figure 1(a) or (b). As the approximate zeros are closer to each other like (a), the distribution of the approximate zeros may become more different from that of the ideal zeros. On the other hand, as the approximate zeros are more scattered like (b), it becomes more likely that the distribution of the approximate zeros is similar to that of the ideal zeros. Therefore, it is better that the approximate zeros are more scattered. In order to follow this study, Tsuno et al. [5] chooses the restart cycle when the following condition is satisfied.

[Condition 3.1] As for $\forall\lambda_{s_1}^{(i+1)}$, There are no zeros $\lambda_{s_2}^{(j)}$ in the rectangular

$$T(\lambda_{s_1}^{(i+1)}) := \Big\{ z \in \boldsymbol{C} \mid \mathbf{re}|\lambda_{s_1}^{(i+1)} - z| < \frac{1}{2}\mathbf{M_{re}}/(\ell-1), \\ \mathbf{im}|\lambda_{s_1}^{(i+1)} - z| < \frac{1}{2}\mathbf{M_{im}}/(\ell-1) \Big\},$$

where

$$\mathbf{M_{re}} = \max_{s_1,s_2,j}\{\mathbf{re}(\lambda_{s_1}^{(i+1)}),\ \mathbf{re}(\lambda_{s_2}^{(j)})\} - \min_{s_1,s_2,j}\{\mathbf{re}(\lambda_{s_1}^{(i+1)}),\ \mathbf{re}(\lambda_{s_2}^{(j)})\}$$
$$\mathbf{M_{im}} = \max_{s_1,s_2,j}\{\mathbf{im}(\lambda_{s_1}^{(i+1)}),\ \mathbf{im}(\lambda_{s_2}^{(j)})\} - \min_{s_1,s_2,j}\{\mathbf{im}(\lambda_{s_1}^{(i+1)}),\ \mathbf{im}(\lambda_{s_2}^{(j)})\}$$

For the detail of this algorithm, see Tsuno et al. [5].

3.2 The Condition for the Convergence Test of Residual Norm

We study the convergence test of residual norm [6]. The $\ell(=\tilde{\ell}+k)$-th residual vector is given as follows [2],

$$\boldsymbol{r}_{\ell} = \boldsymbol{r}_{\tilde{\ell}} + \boldsymbol{d}_k, \quad \boldsymbol{d}_k = AV_k\boldsymbol{y}_k \tag{7}$$

where $\tilde{\ell}$ is the iteration steps when the last restart was done and $\boldsymbol{r}_{\tilde{\ell}}$ is the $\tilde{\ell}$-th residual vector. $\boldsymbol{d}_k$ is the search vector and can be obtained during Arnoldi process. We define the following parameter for the convergence test of residual norm.

$$\zeta_{\ell} = \frac{|\left(\boldsymbol{r}_{\tilde{\ell}},\ \boldsymbol{d}_k\right)|}{\|\boldsymbol{r}_{\tilde{\ell}}\|_2\|\boldsymbol{d}_k\|_2} \tag{8}$$

$$= \sqrt{1-\|\boldsymbol{r}_{\ell}\|_2^2\ /\ \|\boldsymbol{r}_{\tilde{\ell}}\|_2^2} \tag{9}$$

The formulas (8) and (9) show the inner product between $\boldsymbol{r}_{\tilde{\ell}}$ and $\boldsymbol{d}_k$. Since $\|\boldsymbol{r}_{\ell}\|_2$ and $\|\boldsymbol{r}_{\tilde{\ell}}\|_2$ are often computed during the iteration steps in order to obtain the stopping criterion, we use formula (9).

As ζ_{ℓ} gets larger, the residual norm converges faster [6]. When the condition

$$\zeta_{\ell} > \varepsilon \tag{10}$$

is satisfied, we consider that the convergence of residual norm does not stagnate. Even if "Condition 3.1" in Subsection 3.1 is not satisfied, we perform the restart when formula (10) is satisfied.

We also choose ε adaptively during the iteration steps because the algorithm becomes useless if the parameter is determined by users. The details are to be given in Subsection 3.3.

3.3 Restarting Process Based on Two Conditions

We choose the restart cycle based on "Condition 3.1" in Subsection 3.1 and the formula (10) in Subsection 3.2. The determination of the restart is carried out in the following sequences.

(Step 1) When the number of iteration steps is even, k approximate zeros and the current ζ_{ℓ} are computed after Arnoldi process. The current ζ_{ℓ} is set to ρ.

(Step 2) The determination of the restart is carried out.

(Step 2.1) If "Condition 3.1" is satisfied or the number of total iteration steps is 2, the restart is performed with setting $\varepsilon = \rho$.

(Step 2.2) If formula (10) is satisfied or the restart cycle is m_{max}, the restart is performed without setting $\varepsilon = \rho$.

At least one zero has real value if the number of approximate zeros is odd. The real zero can not closely resemble any ideal zeros at all if all of the ideal zeros have complex values [5]. Therefore the restart is determined only at every even step in Step 1. Step 2.1 means the restart is performed when "Condition 3.1"

BC-GMRES($\leq m_{max}$) algorithm

Choose $\boldsymbol{x}_0$
$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \quad \ell = 0, \quad \tilde{\ell} = 0, \quad k = 1$
start:
$\ell = \ell + 1$
Compute V_k through Arnoldi process. **Update** $\boldsymbol{x}_\ell$, $\boldsymbol{r}_\ell$ and $\boldsymbol{d}_k$.
if $\|\boldsymbol{r}_\ell\|_2$ is small enough **then**
  Stop iteration steps
endif
if (k mod 2) $= 0$ **then**
  Compute k new zeros and ζ_ℓ. **Set** $\rho = \zeta_\ell$
  if "Condition 3.1" is satisfied **or** $\ell = 2$ **then**
    Fix k new zeros
    $k = 1, \quad \tilde{\ell} = \ell, \quad \boldsymbol{x}_{\tilde{\ell}} = \boldsymbol{x}_\ell, \quad \boldsymbol{r}_{\tilde{\ell}} = \boldsymbol{r}_\ell, \quad \varepsilon = \rho$
    goto start
  endif
  if the condition (10) is satisfied **or** $k = m_{max}$ **then**
    Fix k new zeros
    $k = 1, \quad \tilde{\ell} = \ell, \quad \boldsymbol{x}_{\tilde{\ell}} = \boldsymbol{x}_\ell, \quad \boldsymbol{r}_{\tilde{\ell}} = \boldsymbol{r}_\ell$
    goto start
  endif
endif
$k = k + 1$
goto start

Fig. 2. BC-GMRES($\leq m_{max}$) algorithm

is satisfied. At the same time, ε is set to ρ. Since the initial distribution of the approximate zeros has to be determined, the restart is forced to be performed in the 2nd iteration step. Step 2.2 means the restart is also performed if the convergence of residual norm does not stagnate. However, we notice that ε is not updated in this case. In a summary, the current ζ_ℓ is set to ε when the approximate zeros are scattered enough. If the next ζ_ℓ is greater than the current ζ_ℓ, to say ε, we consider that performing the restart is effective. It is because there are the results that the restart was performed when the current ζ_ℓ was computed and was set to ε.

We also show the new algorithm by means of Figure 2. The determination of the restart is carried out after the approximate solution and the residual vector are obtained. The new variant of GMRES(m) algorithm with the adaptive restarting process based on these two conditions is called BC-GMRES($\leq m_{max}$) algorithm. We notice that "BC" means "Bi-Condition."

4 Numerical Examples

All of the numerical examples are carried out using the MIMD parallel computer Compaq Beowulf. This system consists of 16 cells (Alpha 600MHz).

4.1 The Comparison of Three Algorithms

We compare BC-GMRES($\leq m_{max}$) algorithm with GMRES($\leq m_{max}$) algorithm and GMRES(m) algorithm for solving the linear systems (1). In this subsection, two numerical examples are presented.

Table 1. The numerical results in example 1 (time: computation time (sec), iter: iteration steps)

Algorithm	Dh							
	2^{-5}		2^{-4}		2^{-3}		2^{-2}	
	time	iter	time	iter	time	iter	time	iter
GMRES(10)	...	...	...	...	...	...	...	...
GMRES(20)	2162.0	41150	1365.0	26890	1232.0	24449	1069.0	21255
GMRES(30)	1546.0	21478	1088.0	15510	1490.0	20181	1014.0	14034
GMRES(40)	1297.0	14666	1316.0	14272	1112.0	12562	921.0	10460
GMRES(50)	2044.0	18521	1160.0	10335	1107.0	9866	1203.0	10213
GMRES($\leq$10)	549.0	16351	557.0	16455	414.0	12751	625.0	18267
GMRES($\leq$20)	669.0	12837	737.0	14158	719.0	13411	822.0	14675
GMRES($\leq$30)	690.0	9745	601.0	8609	663.0	9635	788.0	10921
GMRES($\leq$40)	581.0	7803	634.0	8553	731.0	9651	716.0	9453
GMRES($\leq$50)	630.0	7717	723.0	9056	757.0	9449	836.0	10288
BC-GMRES($\leq$10)	305.0	12345	271.0	11040	275.0	11088	271.0	11676
BC-GMRES($\leq$20)	232.0	8497	300.0	11254	231.0	8975	254.0	10626
BC-GMRES($\leq$30)	267.0	9843	302.0	11254	248.0	9764	223.0	9446
BC-GMRES($\leq$40)	229.0	7958	297.0	11254	254.0	9764	226.0	9446
BC-GMRES($\leq$50)	231.0	7958	298.0	11254	252.0	9764	227.0	9446

(...): The stopping criterion (11) could not be satisfied in one hour.

[Example 1.] We consider the boundary value problem of linear partial differential equation in the region $\Omega = [0,\ 1]^2$ as follows [4],

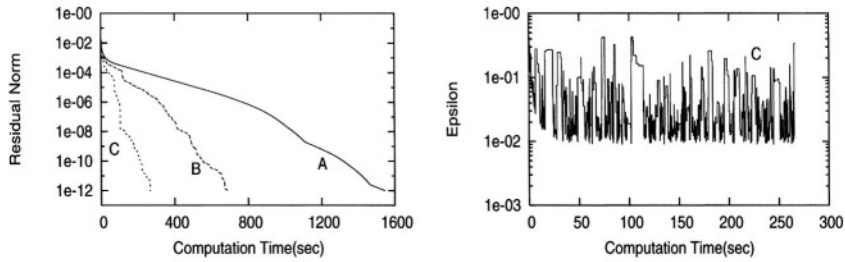
$$-u_{xx} - u_{yy} + D\{(y - 1/2)u_x + (x - 2/3)(x - 1/3)u_y\} = f \ on \ \Omega$$
$$u(x, y)|_{\partial\Omega} = 1 + xy$$

where f is determined so that the exact solution is $u = 1 + xy$. We use the five points differential scheme with 512 × 512 grid points to obtain the linear systems (1) whose dimension is 262,144. In Example 1, the initial approximate solution is determined as $\boldsymbol{x}_0 = 0$. The stopping criterion is

$$\|\boldsymbol{r}_\ell\|_2 / \|\boldsymbol{b}\|_2 < 1.0 \times 10^{-12} \tag{11}$$

We show the computation time and the number of iteration steps needed to satisfy the stopping criterion (11) in Table 1 as the numerical results, where $h = 1/513$. In all of the cases, BC-GMRES($\leq m_{max}$) algorithm performs better than the two other conventional algorithms. Roughly speaking, the computation time of BC-GMRES($\leq m_{max}$) algorithm requires only less than 50% of the GMRES($\leq m_{max}$) algorithm in most of the cases. On the other hand, the computation time of GMRES(m) algorithm requires more than 50% of the GMRES($\leq m_{max}$) algorithm in most of the cases. We also show the convergence behavior of residual norm and ε in Figure 3, where the case is $Dh = 2^{-5}$. The residual norm of BC-GMRES(30) algorithm converges about five times as fast as GMRES(30) algorithm in Figure 3 (a). Moreover, from Figure 3 (b), we can see that ε is varied automatically during the iteration steps. Therefore, ε does not have to be determined by users.

[Example 2.] We consider the boundary value problem of nonlinear partial differential equations in the region $\Omega = [0,\ 1]^3$ as follows [3],



(a) Residual norm vs. Computation time (b) ε vs. Computation time

Fig. 3. Example 1: The behaviors of the residual norm vs. computation time and ε vs. computation time ($Dh = 2^{-5}$), A: GMRES(30), B: GMRES(≤ 30), C:BC-GMRES(≤ 30)

$$\begin{cases} u_{xx} + u_{yy} + u_{zz} + D(uu_x + vu_y + wu_z) + u = f_1 \text{ on } \Omega \\ v_{xx} + v_{yy} + v_{zz} + D(uv_x + vv_y + wv_z) + v = f_2 \text{ on } \Omega \\ w_{xx} + w_{yy} + w_{zz} + D(uw_x + vw_y + ww_z) + w = f_3 \text{ on } \Omega \end{cases} \tag{12}$$

where functions $f_1,\ f_2,\ f_3$ and conditions of the boundary value are determined so that

$$u = \sin(\pi x)\cos(\pi y)\cos(\pi z),\ v = \cos(\pi x)\sin(\pi y)\cos(\pi z),$$
$$w = \cos(\pi x)\cos(\pi y)\sin(\pi z)$$

are satisfied. We use the seven points differential scheme with $80 \times 80 \times 80$ to obtain the nonlinear systems whose dimension is 1,536,000. We utilize Newton method to solve these nonlinear systems. Solving the linear systems (1) is required to compute the multiplication of the inverse of Jacobi matrix and the vector once per iteration step of Newton recurrence. The stopping criterion and the initial approximate solution of the linear systems (1) are just the same as in Example 1. The stopping criterion for Newton recurrence is

$$\|\boldsymbol{q}(\boldsymbol{s}_\ell)\|_2 / \|\boldsymbol{q}(\boldsymbol{s}_0)\|_2 < 1.0 \times 10^{-12} \tag{13}$$

where $\boldsymbol{s}_\ell$ and $\boldsymbol{q}(\boldsymbol{s}_\ell)$ are the ℓ-th approximate solution and the ℓ-th residual vector of Newton recurrence, respectively. We use the linear Lagrange polynomial running from $(0,\ jh,\ kh)$ to $(1,\ jh,\ kh)$ as for $\forall i,\ j$ to obtain the initial approximate solution of Newton recurrence $\boldsymbol{s}_0$. We show the computation time and iteration steps needed to satisfy the stopping criterion (13) in Table 2 as the numerical results. The computation time of Newton recurrence means the total computation time required for solving the linear systems (1). For example, if the number of iteration steps of Newton recurrence is three, the linear systems (1) is required to be solved three times. Therefore, in this case, the computation time of Newton recurrence means the time needed for solving the linear systems (1) three times. The computation time of BC-GMRES(≤ 10) algorithm in $Dh = 2^{-4}$ is more

Table 2. The numerical results in example 2 (time: computation time of Newton recurrence (sec), iter: iteration steps of Newton recurrence)

Algorithm	Dh							
	2^{-5}		2^{-4}		2^{-3}		2^{-2}	
	time	iter	time	iter	time	iter	time	iter
GMRES(10)	993.0	3	1365.0	4	1428.0	4	2056.0	5
GMRES(20)	893.0	3	1228.0	4	1277.0	4	1524.0	5
GMRES(30)	973.0	3	1306.0	4	1257.0	4	1638.0	5
GMRES(40)	920.0	3	1397.0	4	1405.0	4	1885.0	5
GMRES(50)	1148.0	3	1570.0	4	1546.0	4	2152.0	5
GMRES(≤ 10)	440.0	3	577.0	4	660.0	4	846.0	5
GMRES(≤ 20)	574.0	3	818.0	4	913.0	4	1229.0	5
GMRES(≤ 30)	654.0	3	915.0	4	996.0	4	1343.0	5
GMRES(≤ 40)	732.0	3	1046.0	4	1107.0	4	1437.0	5
GMRES(≤ 50)	745.0	3	1053.0	4	1069.0	4	1476.0	5
BC-GMRES(≤ 10)	440.0	3	625.0	4	439.0	4	617.0	5
BC-GMRES(≤ 20)	291.0	3	417.0	4	421.0	4	555.0	5
BC-GMRES(≤ 30)	306.0	3	419.0	4	429.0	4	570.0	5
BC-GMRES(≤ 40)	303.0	3	420.0	4	428.0	4	572.0	5
BC-GMRES(≤ 50)	297.0	3	420.0	4	438.0	4	570.0	5

costed about 10% than GMRES(≤ 10) algorithm. The computation time of BC-GMRES(≤ 10) in $Dh = 2^{-5}$ is just the same as GMRES(≤ 10) algorithm. Except for these two cases, however, the computation time of BC-GMRES($\leq m_{max}$) algorithm is shortened by less than 50% of GMRES($\leq m_{max}$) algorithm. Moreover, the computation time of BC-GMRES($\leq m_{max}$) algorithm is required only about 30% of GMRES(m) algorithm in most of the cases. Therefore, just like in Example 1, we can see that the computation time of BC-GMRES($\leq m_{max}$) algorithm is less costed than the two other conventional algorithms.

5 Conclusions

We have presented BC-GMRES($\leq m_{max}$) algorithm and have analyzed its performance for solving the linear systems (1). Its computation time was also compared with GMRES(m) algorithm and GMRES($\leq m_{max}$) algorithm. As for BC-GMRES($\leq m_{max}$) algorithm, we can see that its computation time is shorter than GMRES(m) algorithm and GMRES($\leq m_{max}$) algorithm in most of the cases in the two examples. Therefore, the major conclusion is that BC-GMRES($\leq m_{max}$) algorithm is more effective than these two conventional algorithms.

References

1. Arnoldi, W.: The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem, *Quart. Appl. Math.,* Vol. 9, pp. 17–29, (1951).
2. Saad, Y. and Schultz, M. K.: GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.,* No. 7, pp. 856–869, (1986).

3. Schönauer, W.: Scientific Computing on Vector Computers, North Holland, (1987).
4. Joubert, W.: Lanczos Methods for the Solution of Nonsymmetric Systems of Linear Equations, *SIAM J. Matrix. Anal. Appl.,* Vol. 13, No. 3, pp. 928–943, (1992).
5. N. Tsuno, T. Nodera.: The Speedup of the GMRES(m) Method Using the Early Restarting Procedure, (in Japanese), *J. of IPSJ,* Vol. 40, No. 4, pp. 1760–1773, (1999).
6. K. Moriya, T. Nodera.: The DEFLATED-GMRES(m, k) Method with Switching the Restart Frequency Dynamically, *Numer. Linear. Algebra. Appl.,* Vol. 7, pp. 569–584, (2000).

The Non-blocking Programming Paradigm in Large Scale Scientific Computations*

Philippas Tsigas and Yi Zhang

Department of Computing Science,
Chalmers University of Technology,
SE-412 60, Gothenburg, Sweden

Abstract. Non-blocking implementation of shared data objects is a new alternative approach to the problem of designing scalable shared data objects for multiprocessor systems. Non-blocking implementations allow multiple tasks to access a shared object at the same time, but without enforcing mutual exclusion to accomplish this. Since, in non-blocking implementations of shared data objects, one process is not allowed to block another process, non-blocking shared data objects have the following significant advantages over lock-based ones: 1) they avoid lock convoys and contention points (locks). 2) they provide high fault tolerance (processor failures will never corrupt shared data objects) and eliminates deadlock scenarios, where two or more tasks are waiting for locks held by the other. 3) they do not give priority inversion scenarios. As shown in [1, 2], non-blocking synchronisation has better performance in certain application than blocking synchronisation. In this paper, we try to provide an in depth understanding of the performance benefits of integrating non-blocking synchronisation in scientific computing applications.

1 Introduction

Shared memory multiprocessors are ideal systems for Large Scale Scientific Computations. Modern multiprocessors offer the shared memory programming paradigm together with low latency and high bandwidth interprocessor communication. Fast interprocessor communication gives to the programmers the possibility of exploring fine grain parallelism in their programs. Moreover, because processors communicate with each other by using conventional memory operations on shared memory, it is relatively easy to transfer sequential programs into parallel ones that run on top of shared memory multiprocessors.

A shared memory multiprocessor system consists of multiple processors, provides a single address space for programming, and supports communication between processors through operations on shared memory. Applications running on such systems may use more than one processor at the same time. Programs can improve their execution speed by exploiting the parallelism available on such systems. Single address space shared memory provides an easy programming model

* This work was partially supported by the Swedish Research Council (VR).

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1114–1124, 2004.
© Springer-Verlag Berlin Heidelberg 2004

to programmers. Shared memory operations can be implemented in hardware or software.

To programmers, programming for shared memory multiprocessors is similar to traditional sequential programming for uniprocessor systems. Communication between processors in shared memory multiprocessors is implicit and transparent via conventional memory access instructions, such as Read/Write, that are also used in sequential programming. Therefore, programmers do not have to consider details of low-level communication between processors and can focus mainly on the applications themselves. When an application is running on shared memory multiprocessors, all processes of the application share the same address space; traditional sequential programming also treats memory as a single address space. Such similarity in programming between shared memory multiprocessors and uniprocessors makes shared memory multiprocessors attractive.

Programming for shared memory multiprocessors introduces synchronisation problems that sequential programming does not need to address. Processes in shared memory multiprocessors communicate and coordinate with each other through reading from and writing to shared memory locations. Such Read/Write operations on memory can be executed simultaneously on several processors. The final results of these operations depend on their interleaving. To maintain consistency, synchronisation is used to guarantee that only desired interleaving of operations can happen. There are two ways to do synchronisation in shared memory: mutual exclusion and non-blocking synchronisation.

Mutual exclusion ensures that certain sections of code will not be executed by more than one process simultaneously. The standard solution to mutual exclusion at kernel level in uniprocessor systems is to momentarily disable interrupts to guarantee that the operation of a shared memory object will not be preempted before it completes. This solution is not feasible for uniprocessor systems at user level, where users do not have the privilege to disable interrupts. In multiprocessor systems, where processes execute on several processors, disabling interrupts at kernel level is too costly. In such cases, locks are used to guarantee that only one process can access a shared memory object: before a process accesses a shared memory object, it must get the lock associated with the object; after accessing the object, it will release the lock. Usually only one lock protects an object. The part of code that the process executes in order to access the object is called code in "critical section". If a process cannot get the lock of an object, then another process owns the lock and is working on the object in the critical section.

Non-blocking synchronisation is an alternative to mutual exclusion for implementing shared data objects. Shared data objects implemented with non-blocking synchronisation do not rely on mutual exclusion and do not require any communication with the kernel. Rather, they rely on hardware atomic primitives such as Compare-and-Swap or the pair Load-Link and Store-Conditional.

An implementation of a shared data object is called non-blocking if first it supports concurrency: several processes can perform operations on the shared data object concurrently; and moreover if it ensures that at any point of time

some/all of the non-fault concurrent processes will complete their operations on the object in a bounded time regardless of the speed or status of other processes. If an implementation guarantees progress of some non-fault processes, it is called lock-free; if it guarantees progress of all non-fault processes, it is called wait-free. This requirement rules out the use of locks for non-blocking synchronisation: if a process crashes while holding a lock, no process waiting for the lock can make any progress.

Compared to mutual exclusion, non-blocking synchronisation has the following significant advantages:

1. it avoids lock convoying effects [3]: if a process holding a lock is preempted or delayed, any other process waiting for the lock is unable to perform any useful work until the process holding the locks has finished its access to the shared object.
2. it provides high fault tolerance. By the definition of non-blocking synchronisation, failures of processes should never corrupt the shared data objects. When using mutual exclusion, a process which dies during modifying a shared object in its critical section might leave the shared object in an invalid state. Some kind of fault recovery technique must be used to recover the object then.
3. it eliminates deadlock scenarios, where two or more tasks are waiting for locks held by the other.
4. it does not give priority inversion scenarios.

Non-blocking programming paradigm is different from the lock-based programming paradigm. In this paper, we investigate how the performance of scientific computing applications is effected by adopting the non-blocking programming paradigm.

2 Previous and Current Work

Previously micro-benchmarks have been widely used to evaluation the performance of synchronisation mechanisms on small scale symmetric multiprocessors, as well as distributed memory machines [4,5,6,7,8] or simulators [6,9]. Although micro-benchmarks are useful since they may isolate performance issues, they do not represent the behaviours of real applications. The goal of designing efficient synchronisation mechanisms is to improve performance of real applications, which micro-benchmarks may not represent well.

For non-blocking synchronisation, many researchers proposed the use of non-blocking synchronisation, rather than blocking one, in the design of inter-process communication mechanisms for parallel and high performance computing. This advocacy is intuitive, but has not been investigated on top of real and well-understood applications; such an investigation could also reveal the effectiveness of non-blocking synchronisation on different applications. To address this need, in [1,2], Tsigas and Zhang showed how performance and speedup in parallel

applications would be affected by using non-blocking rather than blocking synchronisation. They performed a fair evaluation of non-blocking synchronisation and blocking based synchronisation in the context of well-established parallel benchmark applications.

In this paper, we try to provide an in depth understanding of the performance benefits of integrating non-blocking synchronisation in scientific computing applications.

3 Performance Impact of Non-blocking

As shown in [1,2], non-blocking synchronisation gives better performance in certain applications than the respective blocking synchronisation. The fact that non-blocking synchronisation avoids lock convoying effects is believed to be one of the main contributors to the performance improvement that comes with non-blocking synchronisation. Lock convoying effects are caused because of preemption of the processes running the applications. From our experience with non-blocking synchronisation, preemption of processes does contribute to performance degradation on applications with blocking based synchronisation. But it is not the only fact that effect the performance. We performed a set of experiments by running different scientific computing applications with exclusive use and without exclusive use on a cache coherent multiprocessor machine. The results we received with exclusive use are, of course, better than the results without exclusive use due to less frequent preemption. However, the performance gap between blocking and non-blocking synchronisation does not narrow much when changing from non-exclusive use to exclusive use. Avoiding the lock convoy effects only contributes a small part into the performance improvement that comes with non-blocking synchronisation.

On modern cache-coherent shared memory multiprocessors, the cache behaviour of an application also effect the performance of the application. The speed of improvement of processor speed exceeds the speed of improvement of memory accessing speed. This results to a bigger and bigger speed gap between processor speed and memory accessing speed. Cache, a small and fast memory located close to processors, is introduced to reduce the performance gap. However when the data required by a processor is not in the cache, a *cache miss,* takes place and operations on memory still need to be performed. Therefore *cache misses* are expensive for the performance of a program. Researchers of parallel applications are developing cache-conscious algorithms and applications to minimise cache misses during the execution of the application.

The performance difference between blocking and non-blocking synchronisation in applications on cache-coherent shared memory machines makes it interesting to investigate the cache behaviour of different synchronisation mechanisms.

Applications with blocking synchronisation usually use a lock to protect shared variables. When shared variables need to be updated, a lock must be acquired. Then variables can be computed and updated. After the variables

are updated, the lock must be released. A scenario of such operations from the Spark98 [10] is given in figure 1.

```
spark_setlock(lockid);
w[col][0] += A[Anext][0][0]*v[i][0] + ... ...;
w[col][1] += A[Anext][0][1]*v[i][0] + ... ...;
w[col][2] += A[Anext][0][2]*v[i][0] + ... ...;
spark_unsetlock(lockid);
```

Fig. 1. Lock-based operations in Spark98

when they update the shared variables, the operation that acquires the lock and the operations that update shared variables may cause cache misses. The lock usually become a memory bottleneck as all process want to access and modify it. The lock is usually not located at the same cache line with the shared variables which it protects.

Comparing it with blocking synchronisation, non-blocking synchronisation has better cache behaviour. The code for the same operation in the non-blocking programming paradigm is given in figure 2. Here, only the operations on the shared variables themselves may cause cache misses.

```
dfad(&w[col][0], A[Anext][0][0]*v[i][0] + ... ...);
dfad(&w[col][1], A[Anext][0][1]*v[i][0] + ... ...);
dfad(&w[col][2], A[Anext][0][2]*v[i][0] + ... ...);
```

Fig. 2. The non-blocking version of the previous operations in Spark98

Comparing the two programming paradigms, the non-blocking one requires low number of memory accesses and suffer less from cache misses. When the accessing pattern of shared memory is irregular, the cache system cannot predicate its pattern and the application has more chances to suffer from cache misses. To verify the above claim, we designed and performed the experiments described in next section.

4 Experiments

The purpose of these experiments is to compare the performance of applications that use blocking synchronisation and non-blocking synchronisation. We measured the time each application spend in different parts of the application; we also measure the number of cache misses generated by the application. All of our experiments were perform on a SGI Origin 2000 machine with 29 processors. A brief introduction of the system we used is given below.

4.1 SGI Origin 2000 Platform

The SGI Origin2000 [7] is a typical commercial cache coherent non-uniform memory access (ccNUMA) machine. It has an aggressive, scalable distributed shared memory (DSM) architecture. The ccNUMA architecture maintain a unified, global coherent memory and all resources are managed by a single copy of the operating system. The architecture is much more tightly integrated than in other recent commercial distributed shared memory (DSM) systems. A hardware-based directory cache coherency scheme ensures that data held in memory is consistent on a system-wide basis. Comparing with cache snooping, such a scheme keeps both absolute memory latency and the ratio of remote to local latency low, and provides remote memory bandwidth equal to local memory bandwidth (780MB/s each) [7].

In SGI Origin 2000, two processors form a node and share the same secondary cache. Directory based cache coherent protocol maintains coherence between nodes within one machine. The machine we use has twenty-nine 250MHz MIPS R10000 CPUs with 4MB L2 cache and 20GB main memory.

The SGI Origin 2000 provides two groups of transactional instructions that can be used to implement any other transactional synchronisation operations. The one used in this paper contains two simple operations, `load_linked` and `store_conditional`. The `load_linked` (or `LL`) loads a word from the memory to a register. The matching `store_conditional` (or `SC`) stores back possibly a new value into the memory word, unless the value at the memory word has been modified in the meantime by another process. If the word has not been modified, the store succeeds and a 1 is returned. Otherwise the, `store_conditional` fails, the memory is not modified, and a 0 is returned.

For more information on the SGI Origin 2000 the reader is referred to [7,11].

4.2 Experiments and Results

The first application that we used is the Spark98 kernel suite. Spark98 kernels is a collection of sparse matrix kernels for shared memory and message passing systems. Spark98 kernels have been developed to facilitate system builders with a set of example sparse matrix codes that are simple, realistic, and portable. Each kernel performs a sequence of sparse matrix vector product operations using matrices that are derived from a family of three dimensional finite element earthquake applications. The multiplication of a sparse matrix by a dense vector is central to many computer applications, including scheduling applications based on linear programming and applications that simulate physical systems. More information about Spark98 can be found in [10].

In [2], we showed that the non-blocking version of Spark98 performs better than the lock-based version and also better than the reduction-based version of Spark98. In this section, we examine the reason that the non-blocking version performs better than both blocking versions. More specifically, we want to identify the part of the application that has been improved. We measured for the lock-based version the execution time spend in critical section. For the

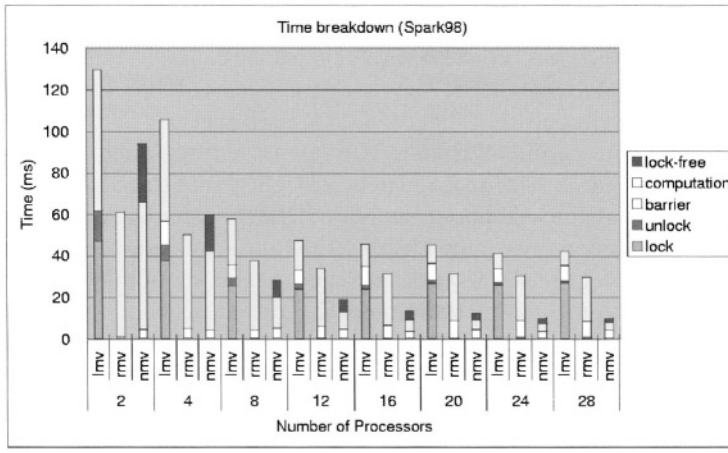


Fig. 3. Time breakdown of different Spark98 version

non-blocking version of the application, which has been improved by adopting the non-blocking synchronisation, we measured the time spend on the lock-free computing part. For all version, the time spend on barrier synchronisation is measured. All versions use barrier operations to synchronise process in different execution phase. Furthermore, the reduction-based version of Spark98 heavily rely on barriers to avoid lock operation. Figure 3 shows the results we have observed. When the number of processors is small, the reduction-based version of Spark98 performs the best: almost all the execution time is dedicated to computation. On the other side, the lock-based and non-blocking versions spend substantial time in synchronisation. On the other hand, when the number of processors becomes larger than 12, the speedup of the reduction-based version of Spark98 stops and the time spend on barrier synchronisation increases as the number of processors increases. The lock-based version can keep its speedup up to 24 processors. Although it is not the best one at the beginning, the non-blocking version performs the best when number of processors become larger than 8. The time spend in lock-free computing and the real computation keep the speedup nicely up to 28 processors. The time spend in barriers is almost constant for the non-blocking version; but for the lock-based and reduction-based ones, the time is almost zero for 2 processors and it is twice as much as non-blocking version when the number of processors reach 28. The larger the time spend in barriers, the more uneven the working load is distributed among processors. The non-blocking version seems to evenly distribute the working load among the processors.

The cache behaviour of these applications are shown in figure 4. As it was described at the beginning of this section, in the SGI Origin 2000 machine, two processors within one node share the same secondary cache. Therefore, only one secondary cache memory caches the main memory in the two-processor case.

When the number of processors is larger than 2, the cache coherent protocol becomes active in order to maintain coherence between several secondary caches. A memory access operation in one node may invalidate a secondary cache line in another node. This is why there is a large difference on the number of secondary cache misses between the 2-processor and the 4-processor experiments shown in figure 4.

From figure 4, the number of cache misses keeps increasing for the reduction-based version; but it keeps decreasing for the lock-based version and the non-blocking version after reaching 4 processors. The number of cache misses for the non-blocking version is always the smallest compared to both the other two versions. When the number of cache misses becomes stable, after 16 processors, it is less than one third of the respective number for the lock-based version which is also stable. The number of cache misses for the reduction-based version keeps increasing as the number of processors increases.

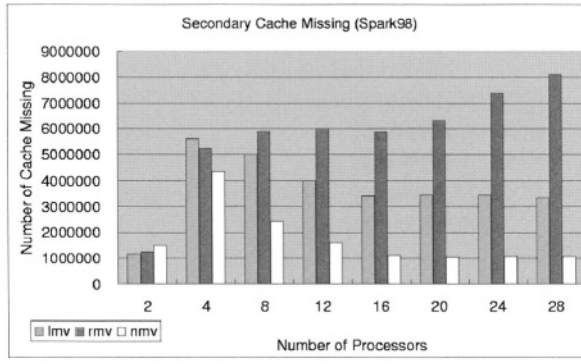


Fig. 4. Cache miss of different Spark98 version

Another application we investigated in this paper is Volrend. Volrend is an application from the SPLASH2 parallel application benchmark [12]. It renders three dimensional volume data into an image using a ray-casting method [13]. The volume data are read only. Its inherent data referencing pattern on data that are written (task queues and image data) is migratory, while its induced pattern at page granularity involves multiple producers with multiple consumers. Both the read accesses to the read only volume and the write accesses to task queues and image data are fine grained, so it suffers both fragmentation and false sharing.

As shown in [2], there is also a large performance difference between the lock-based version and the non-blocking version. We perform the same experiments as we did with Spark98 to investigate the time distribution and cache behaviour of both version.

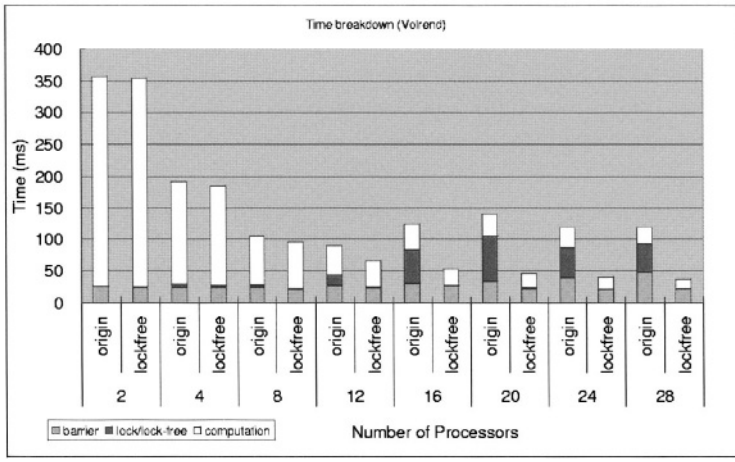


Fig. 5. Time breakdown of different Volrend version

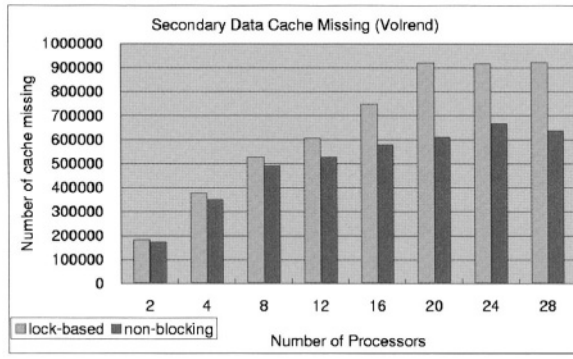


Fig. 6. Cache miss of different Volrend version

The time breakdown for Volrend is shown in figure 5. In the figure, the execution time stops to decrease for the lock-based version when there are more than 12 processors. The time spend in synchronisation increases dramatically when the number of processors reaches 16. But the non-blocking version continues to perform well up to 28 processors and the time spend in lock-free computing is negligible. The time spend in barriers is also almost constant for the non-blocking version of Volrend; but this time doubles for the lock-based version from 2 processors to 28 processors, which means that the non-blocking version offers more fair and balanced working load to processors. The even and balanced working load also contributes to the performance improvements.

The cache behaviour of the two versions of Volrend is shown in figure 6. There is also a large jump on the number of cache misses between 2-processor

and 4-processor as Spark98. In the figure, the non-blocking version also has smaller number of cache misses than the lock-based version in all cases. When all numbers become stable after we reach 20 processors, the number of cache misses for the non-blocking version is about two third of the respective number for the lock-based version.

5 Conclusion

In this paper, we investigate the reason that non-blocking synchronisation performs better than blocking synchronisation in scientific applications. We observed applications using non-blocking synchronisation generate less cache misses than the ones using lock-based synchronisation. Non-blocking synchronisation also balances better the work load among the processors when compared with lock-based synchronisation. Low number of cache misses and balanced work load are the two main reasons that give non-blocking synchronisation better performance. To help parallel programmers who are not experts on non-blocking synchronisation to use non-blocking synchronisation in their applications, a library that supports non-blocking synchronisation called NOBLE [14] has been developed at Chalmers University of Technology, Sweden. The library provides a collection of the most commonly used data types and protocols.

References

1. Tsigas, P., Zhang, Y.: Evaluating the performance of non-blocking synchronisation on shared-memory multiprocessors. In: Proceedings of the ACM SIGMETRICS 2001/Performance 2001, ACM press (2001) 320–321
2. Tsigas, P., Zhang, Y.: Integrating non-blocking synchronisation in parallel applications: Performance advantages and methodologies. In: Proceedings of the 3rd ACM Workshop on Software and Performance (WOSP'02), ACM press (2002) 55–67
3. Kopetz, H., Reisinge, J.: The non-blocking write protocol NBW: A solution to a real-time synchronisation problem. In: Proceedings of the Real-Time Systems Symposium, Raleigh-Durham, NC, IEEE Computer Society Press (1993) 131–137
4. Eichenberger, A., Abraham, S.: Impact of load imbalance on the design of software barriers. In: Proceedings of the 1995 International Conference on Parallel Processing. (1995) 63–72
5. Kumar, S., Jiang, D., Singh, J.P., Chandra, R.: Evaluating synchronization on shared address space multiprocessors: Methodology and performance. In: Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS-99). Volume 27,1., ACM Press (1999) 23–34
6. Kaumlgi, A., Burger, D., Goodman, J.R.: Efficient synchronization: Let them eat QOLB. In: 24th Annual International Symposium on Computer Architecture (24th ISCA'97), Computer Architecture News, ACM SIGARCH (1997) 170–180
7. Laudon, J., Lenoski, D.: The SGI origin: A ccNUMA highly scalable server. In: Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97). Volume 25,2 of Computer Architecture News., New York, ACM Press (1997) 241–251

8. Michael, M.M., Scott, M.L.: Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. Journal of Parallel and Distributed Computing **51** (1998) 1–26
9. Lim, B.H., Agarwal, A.: Reactive synchronization algorithms for multiprocessors. In: Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI), ACM press (1994) 25–35
10. O'Hallaron, D.R.: Spark98: Sparse matrix kernels for shared memory and message passing systems. Technical Report CMU-CS-97-178, CMU (1997)
11. Cortesi, D.: Origin 2000 and onyx2 performance tuning and optimization guide. http://techpubs.sgi.com/library/, SGI Inc. (1998)
12. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: Characteriation and methodological considerations. In: Proceedings of the 22nd Annual International Symposium on Computer Architecture, ACM Press (1995) 24–37
13. Nieh, J., Levoy, M.: Volume rendering on scalable shared memory mimd architectures. In: Proceeding of the 1992 Workshop on Volume Visualization. (1992) 17–24
14. Sundell, H., Tsigas, P.: Noble: A non-blocking inter-process communication library. In: Proceedings of the Sixth Workshop on Languages, Compilers and Run-time Systems for Scalable Computers. (2002)

Comprehensive Air Pollution Studies with the Unified Danish Eulerian Model

Zahari Zlatev

National Environmental Research Institute
Frederiksborgvej 399, P. O. Box 358, DK-4000 Roskilde, Denmark
zz@dmu.dk
http://www.dmu.dk/AtmosphericEnvironment/staff/zlatev.htm

Abstract. Air pollution, especially the reduction of the air pollution to some acceptable levels, is a highly relevant environmental problem, which is becoming more and more important. This problem can successfully be studied only when high-resolution comprehensive mathematical models are developed and used on a routinely basis. However, such models are very time-consuming, even when modern high-speed computers are available. The models need a great amount of input data (meteorological, chemical and emission data). Furthermore, the models are producing huge files of output data, which have to be stored for future uses (for visualization and animation of the results). Finally, huge sets of measurement data (normally taken at many stations located in different countries) have to be used in the efforts to validate the model results. The necessity to handle efficiently large-scale air pollution models in order to be able to resolves a series of important environmental tasks is discussed in this paper. The need for parallel runs is emphasized. The particular model used is the Unified Danish Eulerian Model (UNI-DEM), but most of the results can also be applied when other large-scale models are used. The use of UNI-DEM in some comprehensive air pollution studies is discussed in the end of the paper.

1 Need for Large-Scale Mathematical Models

The control of the pollution levels in different highly polluted regions of Europe and North America (as well as in other highly industrialized parts of the world) is an important task for the modern society. Its relevance has been steadily increasing during the last two-three decades. The need to establish reliable control strategies for the air pollution levels will become even more important in the future. Large-scale air pollution models can successfully be used to design reliable control strategies. Many different tasks have to be solved before starting to run operationally an air pollution model. The following tasks are most important: (a) describe in an adequate way all important physical and chemical processes, (b) apply fast and sufficiently accurate numerical methods in the different parts of the model, (c) ensure that the model runs efficiently on modern high-speed computers (and, first and foremost, on different types of parallel computers), (d)

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1125–1137, 2004.
© Springer-Verlag Berlin Heidelberg 2004

use high quality input data (both meteorological data and emission data) in the runs, (e) verify the model results by comparing them with reliable measurements taken in different parts of the space domain of the model, (f) carry out some sensitivity experiments to check the response of the model to changes of different key parameters and (g) visualize and animate the output results to make them easily understandable also for non-specialists. In this paper we shall concentrate our attention on the solution of the first three tasks. The air pollution model, which is actually used here, is the Unified Danish Eulerian Model (UNI-DEM). The principles are rather general, which means that most of the results are also valid for other air pollution models.

Five physical and chemical processes have to be described by mathematical terms in the beginning of the development of an air pollution model. These processes are: (1) horizontal transport (advection), (2) horizontal diffusion, (3) chemical transformations in the atmosphere combined with emissions from different sources, (4) deposition of pollutants to the surface and (5) vertical exchange (containing both vertical transport and vertical diffusion).

The description of the physical and chemical processes by mathematical terms leads to a system of partial differential equations (PDEs):

$$\begin{aligned}\frac{\partial c_s}{\partial t} &= -\frac{\partial(uc_s)}{\partial x} - \frac{\partial(vc_s)}{\partial y} - \frac{\partial(wc_s)}{\partial z} \\ &+\frac{\partial}{\partial x}\left(K_x\frac{\partial c_s}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y\frac{\partial c_s}{\partial y}\right) + \frac{\partial}{\partial z}\left(K_z\frac{\partial c_s}{\partial z}\right) \\ &+E_s - (\kappa_{1s}+\kappa_{2s})c_s + Q_s(c_1, c_2, \ldots, c_q), \quad s = 1, 2, \ldots, q,\end{aligned} \tag{1}$$

where (i) the concentrations of the chemical species are denoted by c_s, (ii) u, v and w are wind velocities, (iii) K_x, K_y and K_z are diffusion coefficients, (iv) the emission sources are described by E_s, (v) κ_{1s} and κ_{2s} are deposition coefficients and (vi) the chemical reactions are denoted by $Q_s(c_1, c_2, \ldots, c_q)$. Several chemical schemes can be chosen in UNI-DEM (the CBM IV scheme with $q = 35$, [12], is used in this paper).

The space domain of UNI-DEM is a 4800 km x 4800 km square, which contains the whole of Europe together with parts of Africa, Asia, the Arctic area and the Atlantic Ocean.

If initial conditions are available (for example from a previous run of the model), then these are read from the file where they are stored. If initial conditions are not available, then a five day start-up period is used to obtain initial conditions (i.e. the computations are started five days before the desired starting date with some background concentrations and the concentrations found at the end of the fifth day are actually used as starting concentrations).

The choice of lateral boundary conditions is in general very important. If the space domain is very large, then the choice of lateral boundary conditions becomes less important ([6]). The lateral boundary conditions are represented in the Danish Eulerian Model with typical background concentrations which are varied, both seasonally and diurnally.

It is difficult to treat the system of PDEs (1) directly. This is the reason for using different kinds of splitting ([9], [17], [19], [21] and [22]). The system of PDEs (1) is split into three sub-models in the newest version of UNI-DEM:

$$\frac{\partial c_s^{(1)}}{\partial t} = -\frac{\partial (w c_s^{(3)})}{\partial z} + \frac{\partial}{\partial z}\left(K_z \frac{\partial c_s^{(3)}}{\partial z}\right) \tag{2}$$

$$\frac{\partial c_s^{(2)}}{\partial t} = -\frac{\partial (u c_s^{(2)})}{\partial x} - \frac{\partial (v c_s^{(2)})}{\partial y} + \frac{\partial}{\partial x}\left(K_x \frac{\partial c_s^{(2)}}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y \frac{\partial c_s^{(2)}}{\partial y}\right) \tag{3}$$

$$\frac{dc_s^{(3)}}{dt} = E_s + Q_s(c_1^{(3)}, c_2^{(3)}, \ldots, c_q^{(3)}) - (\kappa_{1s} + \kappa_{2s}) c_s^{(3)} \tag{4}$$

The first of these sub-models, (2), describes the vertical exchange. The second sub-model, (3), describes the combined horizontal transport (the advection) and the horizontal diffusion. The last sub-model, (4), describes the chemical reactions together with emission sources and deposition terms.

Assume that the space domain is discretized by using a grid with $N_x \times N_y \times N_z$ grid-points, where N_x, N_y and N_z are the numbers of the grid-points along the grid-lines parallel to the Ox, Oy and Oz axes. Assume further that the number of chemical species involved in the model is $q = N_s$. Denote $N = N_x \times N_y \times N_z \times N_s$. Then the the sub-models (2)-(4) will be transformed into the following three systems of ODEs (ordinary differential equations):

$$\frac{dg^{(1)}}{dt} = f^{(1)}(t, g^{(1)}), \qquad \frac{dg^{(2)}}{dt} = f^{(2)}(t, g^{(2)}), \qquad \frac{dg^{(3)}}{dt} = f^{(3)}(t, g^{(3)}), \tag{5}$$

The components of functions $g^{(i)}(t) \in R^N$, $i = 1, 2, 3$, are the approximations of the concentrations (at time t) at all grid-squares and for all species. The components of functions $f^{(i)}(t, g) \in R^N$, $i = 1, 2, 3$, depend on the numerical method used in the discretization of the spatial derivatives.

A simple linear finite element method ([11], [26], [27]) is used to discretize the spatial derivatives in (2) and (3). The spatial derivatives can also be discretized by using other numerical methods (see [7], [20], [23] and [34]): (a) pseudospectral discretization, (b) semi-Lagrangian discretization (can be used only to discretize the first-order derivatives, i.e. the advection part should not be combined with the diffusion part when this method is to be applied) and (c) methods producing non-negative values of the concentrations (see, for example, [4]). There are no spatial derivatives in (4), which means that the third system of ODEs in (5) is trivially obtained by (4).

It is necessary to couple the three ODE systems. The coupling procedure is connected with the time-integration of these systems. Assume that the values of function g have been found for some $t = t_n$. The next time-step, time-step $n+1$ can be performed by integrating successively the three systems. The values of $g(t_n)$ are used as an initial condition in the solution of the first system in (5). The solution of of the first system in (5) is used as an initial condition of the

second system. Finally, the solution of of the second system is used as an initial condition of of the third system. The solution of the last system in (5) is used as an approximation to $g(t_{n+1})$. In this way, everything is prepared to start the calculations in the next time-step, step $n + 2$.

The first ODE system in (5) can be solved by using classical time-integration methods. The θ-method (see, for example, [18]) is currently used in UNI-DEM.

Predictor-corrector methods with several different correctors ([33]) are used in the solution of the second ODE system in (5). The correctors are carefully chosen so that the stability properties of the method are enhanced.

The solution of the third system in (5) is much more complicated, because this system is both time-consuming and stiff. The QSSA (quasi-steady-state approximation; see [15], [16] and [1]) is simple and relatively stable, but not very accurate (therefore it has to be run with a small time-stepsize). QSSA could be viewed as a very simplified use of differential-algebraic equations (such methods are discussed in [5], [10] and [28]). The classical numerical methods for stiff ODE systems (such as the Backward Euler Method, the Trapezoidal Rule and Runge-Kutta algorithms) lead to the solution of non-linear systems of algebraic equations and, therefore, they are more expensive ([29], [30]). On the other hand, these methods can be incorporated with an error control and perhaps with larger time-steps. The extrapolation methods ([8]) are also promising. It is easy to calculate an error estimation and to carry out the integration with large time-steps when these algorithms are used. However, it is difficult to implement such methods in an efficient way when all three systems in (5) are to be treated successively. Partitioning can also be used ([1]). Some convergence problems related to the implementation of partitioning are studied in [35]. The experiments with different integration methods for the chemical sub-model are continuing. The QSSA with some enhancements ([1]) will be used in this paper.

2 Need for High Performance Computing

The computers are becoming more and more powerful. Many tasks, which several years ago had to be handled on powerful supercomputers, can be handled at present on PCs or work-stations. However, there are still many tasks that can only be run on parallel computers. This is especially true for the large air pollution models. The numbers of equations in the ODE systems (5) for some 3-D options of UNI-DEM are given in Table 1 in order demonstrate the fact that high performance computing is needed when large air pollution models are to be treated. It should be reiterated here that the ODE systems (5) have to be handled at each time-step and the number of time-steps for a one-year run is about 250 000. Furthermore, many scenarios are to be run in comprehensive air pollution studies. It is not possible to solve all of the tasks listed in Table 1 at the computers available at present.

It should also be mentioned here that the computational tasks are reduced by a factor approximately equal to 10 when 2-D options of UNI-DEM are used

instead of the 3-D options, but also in this case the tasks are very large when fine resolution grids are specified.

It is very important to resolve in an efficient way two problems: (a) to improve the performance on one processor and (b) to achieve high speed ups when the job is run on several processors in parallel. The actions taken in the solution of these two problems are shortly discussed below (much more details can be found in [11], [24] and [25]).

The large two-dimensional arrays are divided into sections (chunks) in order to resolve the first problem. An attempt to use small amounts of data (which can be kept in cache) as long as possible is carried out in this way. In other words, the method is essentially based on the reorganization of the computations (and can be applied when different numerical methods are used). Good results can be obtained when the size of chunks, NSIZE, is not large (in the range from 16 to 96). On some computers the improvements are very considerable (this is demonstrated in [11], [24] and [25]).

It is important to solve the second problem by using standard parallelization tools. This is why only OpenMP ([31]) and MPI ([13]) are used in UNI-DEM. The implementation of these tools is described in [11], [24] and [25]. Some numerical results will be given in the next section.

Table 1. The number of equations per system of ODEs in (5).

Number of species	$(96 \times 96 \times 10)$	$(288 \times 288 \times 10)$	$(480 \times 480 \times 10)$
35	3225600	29030400	80640000
56	5160960	46448640	129024000
168	15482880	139345920	387072000

3 Numerical Results

Some results will be presented in this sections to demonstrate (i) the efficiency of the better utilization of the cache memory by using chunks and (ii) the good speed-ups (very often super-linear) that can be achieved when the code is run in parallel. We start by presenting short information about the initialization of UNI-DEM and about the computers used.

Initialization of UNI-DEM. UNI-DEM is a very flexible model in which many different option can be specified by using seven major parameters. These parameters are to be stored in a small input file. The list of the parameters and the allowed values are given in Table 2.

Description of the grid of Sun computers. Sun computers located at the Danish Centre for Scientific Computing (the Danish Technical University in Lyngby) were used in the runs. The computers and the their characteristics are shown in Table 3. All these computers were connected with a 1Gbit/s Switch.

Table 2. The seven major parameters used in the decision how to run UNI-DEM

No.	Parameter	Function	Allowed values	Desired features
1	NX	Grid points along Ox	96, 288 or 480	
2	NY	Grid points along Oy	NY = NX	Rectangular domains
3	NZ	Grid points along Oz	1 or 10	More layers
4	NSPECIES	Chemical species	35, 56 or 168	More chemical schemes
5	NSIZE	Size of chunks	Divisor of $NX \times NY$	
6	NREFINED	Resolution of emissions	0 (coarse) or 1 (fine)	
7	NYEAR	The year to be used	From 1989 to 1998	More years

Table 3. The computers available at the Sun grid.

Computer	Type	Power	RAM	Processors
Bohr	Sun Fire 6800	UltraSparc-III 750 MHrz	48 GB	24
Erlang	Sun Fire 6800	UltraSparc-III 750 MHrz	48 GB	24
Hald	Sun Fire 12k	UltraSparc-III 750 MHrz	144 GB	48
Euler	Sun Fire 6800	UltraSparc-III 750 MHrz	24 GB	24
Hilbert	Sun Fire 6800	UltraSparc-III 750 MHrz	36 GB	24
Newton	Sun Fire 15k	UltraSparc-IIIcu 900 MHrz	404 GB	72

The computers are united in a grid (consisting of 216 processors) so that a job sent without a special demand will be assigned on the computer on which there are sufficiently many free processors. The different computers have processors of different power (therefore, it is in principle possible to use the grid as a heterogeneous architecture, but this option is not available yet).

We have been allowed to use no more than 16 processors, and in the runs in this section we used only "newton" (i.e. we had always a requirement specifying the particular computer on which the job must be run)

More details about the high speed computers that are available at the Technical University of Denmark can be found in [32].

Running the MPI options of UNI-DEM. Four MPI options of UNI-DEM have been tested: (i) the 2-D model on a coarse grid, (ii) the 3-D option on a coarse grid, (iii) the 2-D option on a fine grid and (iv) the 3-D option on a fine grid.

The problems were run with three different sizes *NSIZE* of chunks: (a) the minimal size of the chunks, *NSIZE* = 1 for all cases, (b) a medium size of the chunks, *NSIZE* = 24 for all cases and (c) the maximal size of the chunks, which is *NSIZE* = 1152 for the coarse grid when 8 processors are used and *NSIZE* = 28800 for the fine grid (again when 8 processors are used).

Finally, in most of the cases both 1 processor and 8 processors were used. Some of the jobs were also run on 16 processors.

All runs of the options discretized on the coarse grid were run for the typical period of one year (in which case it is possible to study seasonal variations). The 2-D option of UNI-DEM discretized on the fine grid was run over a period of

one month. Finally, the 3-D option of UNI-DEM discretized on the fine grid was run over a time period of 42 hours. This is a rather short period, but it is still meaningful to a certain degree because several changes from day to night and from night to day occur in this period, which is important for the test of the photo-chemical reactions.

The computing times in all tables are given in seconds. The abbreviations used in the tables can be explained as follows: (a) ADV stands for the horizontal transport + diffusion process, (b) CHEM stands for the process uniting the chemical reactions, the treatment of the emissions and the deposition part, (c) COMM stands for the part needed to perform communications along the inner boundaries, (d) VERT stands for the vertical exchange processes and (e) TOTAL stands for the total computing time (including the sum of the times given in the same column above the last item + the computing times needed for performing input-output operations, pre-processing, post-processing, etc.).

The percentages of the computing times for the different processes related to the total computing times are given in the columns under "Part". The "Speed-up" is the ratio of the computing time on one processor and the computing time on p processors (where p is the number of processors that are used in the run under considerations; as mentioned above, eight processors were as a rule used in our experiments).

Running the 2-D MPI option discretized on the coarse grid. Results from the six runs with this code are shown in Table 4 (runs on one processor performed by using three values of *N SIZE*) and Table 5 (runs on 8 processors performed again with three values of *N SIZE*).

Table 4. Running UNI-DEM discretized on a $96 \times 96 \times 1$ grid on one processor.

	$NSIZE = 1$		$NSIZE = 24$		$NSIZE = 1152$	
Process	Time	Part	Time	Part	Time	Part
ADV	17617	28.2%	16035	32.6%	16742	26.8%
CHEM	37353	59.8%	26671	54.2%	38828	62.1%
COMM	2	0.0%	2	0.0%	2	0.0%
TOTAL	62443	100.0%	49239	100.0%	62510	100.0%

Running the 3-D MPI option discretized on the coarse grid. Results from the six runs with this code are shown in Table 6 (runs on one processor performed by using three values of *N SIZE*) and Table 7 (runs on 8 processors performed again with three values of *N SIZE*).

Running the 2-D MPI option discretized on the fine grid. Results from the six runs with this code are shown in Table 8 (runs on one processor performed by using three values of *N SIZE*) and Table 9 (runs on 8 processors performed again with three values of *N SIZE*).

Running the 3-D MPI option discretized on the fine grid. Results from the six runs with this code are shown in Table 10 (runs on one processor performed

Table 5. Running UNI-DEM discretized on a 96 × 96 × 1 grid on eight processors.

	$NSIZE = 1$			$NSIZE = 24$			$NSIZE = 1152$		
Process	Time	Part	Speed-up	Time	Part	Speed-up	Time	Part	Speed-up
ADV	851	11.1%	20.7	893	13.2%	18.0	860	11.4%	19.5
CHEM	4186	54.4%	8.9	2936	43.4%	6.8	4362	57.6%	8.9
COMM	791	10.4%	-	1110	16.4%	-	452	6.0%	-
TOTAL	7625	100.0%	8.2	6766	100.0%	7.3	7577	100.0%	8.2

by using three values of *N SIZE*) and Table 11 (runs on 8 processors performed again with three values of *N SIZE*).

Major conclusions from the runs. It is seen that the exploitation of the cache memory is always giving good results (compare the results for *NSIZE* = 24 with the results for *NSIZE* = 1 and *NSIZE* = 1152(28800). The speed-ups for the physical processes are super-linear (greater for ADV and VERT than for CHEM, which should be expected, because chunks are used in the chemical parts). The speed-ups for the total computing time are lower, but anyway at least close to linear.

Table 6. Running UNI-DEM discretized on a 96 × 96 × 10 grid on one processor.

	$NSIZE = 1$		$NSIZE = 24$		$NSIZE = 1152$	
Process	Time	Part	Time	Part	Time	Part
ADV	169776	31.5%	159450	37.8%	169865	30.9%
CHEM	337791	62.7%	233471	55.3%	348769	63.4%
VERT	23221	4.3%	21473	5.1%	23014	4.2%
COMM	2	0.0%	2	0.0%	2	0.0%
TOTAL	538953	100.0%	421763	100.0%	549835	100.0%

Table 7. Running UNI-DEM discretized on a 96 × 96 × 10 grid on eight processors.

	$NSIZE = 1$			$NSIZE = 24$			$NSIZE = 1152$		
Process	Time	Part	Speed-up	Time	Part	Speed-up	Time	Part	Speed-up
ADV	18968	27.4%	9.0	18498	33.3%	8.6	18641	26.3%	9.1
CHEM	41334	59.6%	8.2	29189	52.3%	8.0	43291	61.3%	8.1
VERT	1213	1.7%	19.1	1200	2.2%	17.9	1240	1.8%	18.6
COMM	911	1.3%	-	878	1.6%	-	973	1.4%	-
TOTAL	69325	100.0%	7.8	55723	100.0%	7.6	70653	100.0%	7.8

Major conclusions from the runs. It is seen that the exploitation of the cache memory is always giving good results (compare the results for *NSIZE* = 24 with the results for *NSIZE* = 1 and *NSIZE* = 1152(28800). The speed-ups for the physical processes are super-linear (greater for ADV and VERT than

Table 8. Running UNI-DEM discretized on a 480 × 480 × 1 grid on one processor.

	$NSIZE = 1$		$NSIZE = 24$		$NSIZE = 28800$	
Process	Time	Part	Time	Part	Time	Part
ADV	485062	63.9%	484923	70.3%	491704	41.7%
CHEM	224804	29.1%	143923	20.9%	611502	51.8%
COMM	1	0.0%	1	0.0%	2	0.0%
TOTAL	771261	100.0%	690027	100.0%	1179518	100.0%

Table 9. Running UNI-DEM discretized on a 480 × 480 × 1 grid on eight processors.

	$NSIZE = 1$			$NSIZE = 24$			$NSIZE = 28800$		
Process	Time	Part	Speed-up	Time	Part	Speed-up	Time	Part	Speed-up
ADV	34499	45.5%	14.1	34567	48.9%	14.0	33589	26.8%	14.6
CHEM	27159	35.8%	8.3	18816	26.6%	7.6	69168	55.2%	8.4
COMM	5937	7.8%	-	8128	11.5%	-	14474	11.6%	-
TOTAL	75854	100.0%	10.2	70856	100.0%	9.7	125246	100.0%	9.4

Table 10. Running UNI-DEM discretized on a 480 × 480 × 10 grid on one processor.

	$NSIZE = 1$		$NSIZE = 24$		$NSIZE = 28800$	
Process	Time	Part	Time	Part	Time	Part
ADV	261631	67.0%	271419	72.9%	268337	49.8%
CHEM	86317	22.1%	56797	15.3%	228216	42.3%
VERT	40721	10.4%	42320	11.4%	41223	7.6%
COMM	1	0.0%	1	0.0%	1	0.0%
TOTAL	390209	100.0%	372173	100.0%	539319	100.0%

for CHEM, which should be expected, because chunks are used in the chemical parts). The speed-ups for the total computing time are lower, but anyway at least close to linear.

Scaling results for the MPI options. It has been shown in the previous section that the computing times are reduced by a factor close to 8 (and in many cases by a factor greater than 8) when the number of the processors used is increased from 1 to 8. It is desirable that the same tendency holds when

Table 11. Running UNI-DEM discretized on a 480 × 480 × 10 grid on eight processors.

	$NSIZE = 1$			$NSIZE = 24$			$NSIZE = 28800$		
Process	Time	Part	Speed-up	Time	Part	Speed-up	Time	Part	Speed-up
ADV	13606	46.2%	19.2	13515	52.7%	20.1	13374	28.9%	20.1
CHEM	10398	35.3%	8.3	6681	26.0%	8.5	25888	56.0%	8.8
VERT	2830	9.6%	14.4	2802	10.9%	15.1	2709	5.9%	15.2
COMM	2316	7.9%	-	2340	9.1%	-	3925	8.5%	-
TOTAL	29449	100.0%	13.3	25654	100.0%	14.5	46210	100.0%	11.7

Table 12. Running UNI-DEM discretized on a $96 \times 96 \times 10$ grid on 16 processors with $NSIZE = 24$. The Speed-up-8 factors are calculated as ratios of the computing times obtained when 8 processors are used (which are given in Table 5) and the computing times when 16 processors are used. The Speed-up-1 factors are calculated as ratios of the computing times obtained when 1 processor is used (which are given in Table 4) and the computing times when 16 processors are used.

Process	Time	Part	Speed-up-8	Speed-up-1
ADV	8044	27.4%	2.3	19.8
CHEM	14261	48.5%	2.1	16.4
VERT	388	1.3%	3.1	55.3
COMM	4203	14.3%	-	-
TOTAL	29389	100.0%	1.9	14.6

Table 13. Running UNI-DEM discretized on a $480 \times 480 \times 1$ grid on 8 processors by using the MPI option and the OpenMP option. The time period for these two runs was one year.

Process	MPI option	OpenMP option
ADV	822291	1663812
CHEM	393158	596920
COMM	255785	-
TOTAL	1782752	2614983

the number of processors is greater than 8 (i.e. it is desirable that increasing the number of processors used by a factor of k will results in decreasing the computing times by a factor approximately equal to k). It is often said that the parallel algorithm scales well when such a trend can be obtained.

Some runs were performed on 16 processors and the results were compared with those obtained on 8 processors. Some results, which are obtained when the 3-D option of UNI-DEM are run, are given in Table 12 for the coarse grid option. The results indicate that the parallel algorithms applied in UNI-DEM scale very well. Super-linear speed-ups were registered for the main physical processes, while nearly linear speed-ups were found for the total computing times.

Comparing MPI options with OpenMP options. The Sun computers, which were used to calculate the results are shared memory machines. Therefore, one should expect the OpenMP options of the code to be more efficient than the MPI options. In fact, the MPI options are more efficient. In the previous section it was explained why this should be expected. Some results are given in Table 13 in order to illustrate the fact that the leading dimension of of arrays is reduced when the MPI options are used results also in reduction of the computing times.

Plans for further improvements of the performance. The improvement of the fine resolution options of UNI-DEM, especially the 3-D fine resolution option, is an important task which must be resolved in the near future. It is necessary both to improve the performance of the different options of the model and to have access to more processors (and/or to more powerful computers) in order to be able to run operationally fine resolution options of UNI-DEM.

4 Comprehensive Studies Performed by Using UNI-DEM

UNI-DEM has been used to study (i) long-term variations of air pollution levels in Europe, (ii) impact of future climate changes on air pollution levels, (iii) influence of natural emissions on high ozone levels. Results obtained in these studies are reported in [2], [3], [14], [36], [37], [38], [39], [40] and [41].

Acknowledgements. A grant (CPU-1101-17) from the Danish Centre for Scientific Computing (DCSC) gave us access to the Sun computers at the Technical University of Denmark. The members of the staff of DCSC helped us to resolve some difficult problems related to the efficient exploitation of the grid of Sun computers.

References

1. V. ALEXANDROV, A. SAMEH, Y. SIDDIQUE AND Z. ZLATEV, *Numerical integration of chemical ODE problems arising in air pollution models,* Environmental Modelling and Assessment, Vol. 2 (1997), 365–377.
2. C. AMBELAS SKJØTH, A. BASTRUP-BIRK, J. BRANDT AND Z. ZLATEV, *Studying variations of pollution levels in a given region of Europe during a long time-period,* Systems Analysis Modelling Simulation, Vol. 37 (2000), 297-311.
3. A. BASTRUP-BIRK, J. BRANDT, I. URIA AND Z. ZLATEV, *Studying cumulative ozone exposures in Europe during a seven-year period,* Journal of Geophysical Research, Vol. 102 (1997), 23917-23935.
4. A. BOTT, *A positive definite advection scheme obtained by non-linear renormalization of the advective fluxes,* Monthly Weather Review, Vol. 117 (1989), 1006-1015.
5. K. BRENAN, S. CAMPBELL AND L. PETZOLD, *Numerical solution of initial value problems in differential-algebraic equations,* SIAM, Philadelphia, 1996.
6. R. A. BROST, *The sensitivity to input parameters of atmospheric concentrations simulated by a regional chemical model,* Journal of Geophysical Research, Vol. 93 (1988), 2371-2387.
7. W. P. CROWLEY, *Numerical advection experiments,* Monthly Weather Review, Vol. 96 (1968), 1–11.
8. P. DEUFLHARD, (1985). *Recent progress in extrapolation methods for ordinary differential equations.* SIAM Review, Vol. 27 (1985), 505-535.
9. I. DIMOV, I. FARAGO, A. HAVASI AND Z. ZLATEV, *L-Commutativity of the operators in splitting methods for air pollution models,* Annales Univ. Sci. Budapest, Vol. 44, (2001), 129-150.
10. R. DJOUAD AND B. SPORTISSE, *Solving reduced chemical models in air pollution modelling,* Applied Numerical Mathematics, Vol. 40 (2003), 49-61.
11. K. GEORGIEV AND Z. ZLATEV, *Parallel Sparse Matrix Algorithms for Air Pollution Models,* Parallel and Distributed Computing Practices, Vol. 2 (1999), 429-442.
12. M. W. GERY, G. Z. WHITTEN, J. P. KILLUS AND M. C. DODGE, *A photochemical kinetics mechanism for urban and regional computer modeling,* Journal of Geophysical Research, Vol. 94 (1989), 12925–12956.
13. W. GROPP, E. LUSK AND A. SKJELLUM, *Using MPI: Portable programming with the message passing interface,* MIT Press, Cambridge, Massachusetts (1994).

14. A. HAVASI AND Z. ZLATEV, *Trends of Hungarian air pollution levels on a long time-scale,* Atmospheric Environment, Vol 36 (2002), 4145-4156.
15. E. HESSTVEDT, Ø. HOV AND I. A. ISAKSEN, *Quasi-steady-state approximations in air pollution modelling: comparison of two numerical schemes for oxidant prediction,* International Journal of Chemical Kinetics, Vol. 10 (1978), 971–994.
16. Ø. HOV, Z. ZLATEV, R. BERKOWICZ, A. ELIASSEN AND L. P. PRAHM, *Comparison of numerical techniques for use in air pollution models with non-linear chemical reactions,* Atmospheric Environment, Vol. 23 (1988), 967–983.
17. W. HUNSDORFER, B. KOREN, M. VAN LOON AND J. G. VERWER, *A positive finite difference advection scheme,* J. Comput. Phys., Vol. 117 (1995), 35-46.
18. J. D. LAMBERT, *Numerical methods for ordinary differential equations.* Wiley, New York (1991).
19. D. LANCER AND J. G. VERWER, *Analysis of operators splitting for advection-diffusion-reaction problems in air pollution modelling,* J. Comput.Appl. Math., Vol. 111 (1999), 201-216.
20. M. VAN LOON, *Testing interpolation and filtering techniques in connection with a semi-Lagrangian method,* Atmospheric Environment, Vol. 27A (1993), 2351-2364.
21. G. I. MARCHUK, *Mathematical modeling for the problem of the environment,* Studies in Mathematics and Applications, No. 16, North-Holland, Amsterdam (1985).
22. G. J. MCRAE, W. R. GOODIN AND J. H. SEINFELD, *Numerical solution of the atmospheric diffusion equations for chemically reacting flows,* Journal of Computational Physics, Vol. 45 (1984), 1–42.
23. C. R. MOLENKAMPF, *Accuracy of finite-difference methods applied to the advection equation,* Journal of Applied Meteorology, Vol. 7 (1968), 160–167.
24. W. OWCARZ AND Z. ZLATEV, *Running a large air pollution model on an IBM SMP computer,* International Journal of Computer Research, Vol. 10, No. 4 (2001), 321-330.
25. W. OWCZARZ AND Z. ZLATEV, *Parallel matrix computations in air pollution modelling,* Parallel Computing, Vol. 28 (2002), 355-368.
26. D. W. PEPPER AND A. J. BAKER, *A simple one-dimensional finite element algorithm with multidimensional capabilities,* Numerical Heath Transfer, Vol. 3 (1979), 81-95.
27. D. W. PEPPER, C. D. KERN AND P. E. LONG, JR., *Modelling the dispersion of atmospheric pollution using cubic splines and chapeau functions,* Atmospheric Environment, Vol. 13 (1979), 223–237.
28. L. F. SHAMPINE, M. W. REICHELT AND J. A. KIERZENKA, *Solving Index-1 DAEs in MATLAB and Simulink.* SIAM Rev., Vol. 41 (1999), 538-552.
29. J. G. VERWER AND M. VAN LOON, *An evaluation of explicit pseudo-steady state approximation for stiff ODE systems from chemical kinetics,* J. Comp. Phys., Vol. 113 (1996), 347-352.
30. J. G. VERWER AND D. SIMPSON, *Explicit methods for stiff ODE's from atmospheric chemistry,* Appl. Numer. Math., Vol. 18 (1995), 413-430.
31. WEB-SITE FOR OPEN MP TOOLS, http://www.openmp.org, 1999.
32. WEB-SITE OF THE DANISH CENTRE FOR SCIENTIFIC COMPUTING AT THE TECHNICAL UNIVERSITY OF DENMARK, *Sun High Performance Computing Systems,* http://www.hpc.dtu.dk, 2002.
33. Z. ZLATEV, *Application of predictor-corrector schemes with several correctors in solving air pollution problems,* BIT, Vol. 24 (1984), 700–715.
34. Z. ZLATEV, *Computer treatment of large air pollution models,* Kluwer Academic Publishers, Dordrecht-Boston-London (1995).

35. Z. ZLATEV, *Partitioning ODE systems with an application to air pollution models,* Computers and Mathematics with Applications, Vol. 42 (2001), 817-832.
36. Z. ZLATEV, *Massive data set issues in air pollution modelling,* In: Handbook on Massive Data Sets (J. Abello, P. M. Pardalos and M. G. C. Resende, eds.), pp. 1169-1220, Kluwer Academic Publishers, Dordrecht-Boston-London (2002).
37. Z. ZLATEV, J. CHRISTENSEN AND A. ELIASSEN, *Studying high ozone concentrations by using the Danish Eulerian Model,* Atmospheric Environment, Vol. 27A (1993), 845-865.
38. Z. ZLATEV, J. CHRISTENSEN AND Ø. HOV, *An Eulerian air pollution model for Europe with nonlinear chemistry,* Journal of Atmospheric Chemistry, Vol. 15 (1992), 1-37.
39. Z. ZLATEV, I. DIMOV AND K. GEORGIEV, *Studying long-range transport of air pollutants,* Computational Science and Engineering, Vol. 1, No. 3 (1994), 45-52.
40. Z. ZLATEV, I. DIMOV, TZ. OSTROMSKY, G. GEERNAERT, I. TZVETANOV AND A. BASTRUP-BIRK, *Calculating losses of crops in Denmark caused by high ozone levels,* Environmental Modelling and Assessment, Vol. 6 (2001), 35-55.
41. Z. ZLATEV, J. FENGER AND L. MORTENSEN, *Relationships between emission sources and excess ozone concentrations,* Computers and Mathematics with Applications, Vol. 32, No. 11 (1996), 101-123.

Parallel Algorithms for Evolutionary History Reconstruction

Jacek Błażewicz[1,2], Piotr Formanowicz[1,2], Paweł Kędziora[1], and Paweł Wojciechowski[1]

[1] Institute of Computing Science, Poznań University of Technology, Piotrowo 3A, 60-965 Poznań, Poland.
blazewic@put.poznan.pl,
{piotr, Pawel.Kedziora, Pawel.Wojciechowski}@cs.put.poznan.pl
[2] Institute of Bioorganic Chemistry, Polish Academy of Sciences, Noskowskiego 12/14, 61-704 Poznań, Poland.

Abstract. A phylogenetic tree construction is one of the most important problems in computational biology. From computational point of view it is also one of the most difficult problem because of its intrinsic intractability. Efficient algorithms are known only for some special cases of the problem which are unrealistic from biological point of view. Many algorithms are known for the problem, but since the problem is hard, they are usually heuristics. In this paper we present three exact parallel algorithms for the problem. They have been tested in computational experiment ran on SUN Fire computer.

1 Introduction

One of the most important aspects of computational biology is a reconstruction of evolutionary history of a given group of organisms. Such a history is usually shown as a *phylogenetic tree* [5,4,2]. A root in the tree represents a hypothetical ancestor of all species represented by other nodes of the tree. Leaves correspond to species living in the present while internal nodes correspond to hypothetical organisms which appeared during the process of evolution. A phylogenetic tree is only a hypothesis - it shows a possible way of species evolution (the one which is the most probable from some point of view). Building phylogenetic tree for even small group of species is usually difficult task. There are at least two sources of this difficulty. First, not all laws of evolution are known. Hence, algorithms for phylogenetic tree construction are based on some simplified models of evolution. Second, for a construction of such a tree one has only data concerning species living in the present, i.e. those which correspond to leaves in the tree. Hence, the evolutionary history is reconstructed on the basis of the similarity among data representing species living in the present.

There are two main classes of phylogenetic trees:
1) trees constructed on the basis of the similarities among features of given species,
2) trees built on the basis of evolutionary distances between species.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1138–1145, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The features used in the construction of trees of the first class are usually sequences of nucleic acids or proteins. In this case a feature or a *character* is a position in the sequence. A nucleotide or an aminoacid present at this position is a *value* or *state* of the character. Similarities between sequences from the considered set of species are used as a base for tree construction. The tree is constructed in such a way that the number of state changes on a path from the root to any leaf is minimal. In the case of a construction of a tree from the second class one tries to estimate the time which passed from a divergence of given species. In such a tree internal nodes also represent hypothetical ancestors but in addition they are labeled with a time when two given species diverged or with a time which passed since the divergence took place. In the former case values of the labels on a path from root to any leaf should increase. In the latter, they should decrease.

There are known many algorithms for the phylogenetic tree construction. But, since the problem of construction of such a tree is generally computationally hard, efficient exact algorithms are known only for some special cases, e.g. for problems where the number of characters or states is limited to a very small value. Such algorithms are useful in a very limited number of cases. Hence, also heuristic methods are proposed which are able to construct trees based on more general models. Another way to manage the intrinsic intractability of the problem is to design parallel algorithms. The last approach is especially promising, since parallel computers and huge clusters of workstations are becoming more commonly available.

The organization of the paper is as follows. In Section 2 a sequential version of the algorithm is described, while in Section 3 parallel versions are presented. In Section 4 results of computational experiments are shown. The paper ends with conclusions in Section 5.

2 The Method

The sequential version of the algorithms is based on the approach implemented in DNAPENNY - a part of PHYLIP package [3]. The algorithm is of branch and bound type and it searches a space of all phylogenetic trees for a given group of taxa. The space is searched in the following way. At the beginning four organisms are selected from the taxa list. For these organisms all unrooted trees, in this case three, are constructed. The trees are starting points for the construction of trees containing all considered taxa. For these trees parsimony is calculated according to Fitch algorithm [1]. The most parsimonious tree obtained at this stage of the algorithm will be extended first. The next, fifth taxon is tried to be added to the tree in all possible ways, i.e. in every branch of the tree. As a result 15 trees for five taxa is constructed. For all of them again parsimony is calculated and the most parsimonious tree is selected for further extension (the remaining trees also will be extended in further steps of the algorithm). In this way the algorithm finally constructs a tree for all taxa from the considered set. Parsimony score of this tree becomes an upper bound. As one can notice, levels

in the search tree corresponds to the numbers of taxa for which the phylogenetic tree is constructed at a given stage of the method. In the next step the algorithm returns to level $n-1$ of the search tree (where n is the number of considered taxa). At this stage it selects the most parsimonious tree for $n-1$ taxa among those which have not been extended yet. Lower bound at a given stage of tree construction is a minimal number of characters' states changes necessary for extending a current subtree to full phylogenetic tree for n taxa.

3 Parallel Algorithms

In this section parallel versions of the method previously described will be presented.

3.1 Version 1

The algorithm has master-slave structure consisting of p processes. At the beginning process master creates a list of slaves and labels them as being idle. Next, master selects r slaves from the list, where r is a number of possible phylogenetic trees for four species. Then, master sends to each of the selected slaves one of the trees for four taxa. Each of these slaves extends the obtained tree by fifth taxon and returns all but one the resulting trees to master. The tree which has not been sent to master is further extended by the slave. Master sends the received tree and a list of organisms which have not been considered at this stage of the method to idle slaves. Again, each of the slaves extends the obtained tree by sixth organism and sends all but one resulting trees to master. Slaves work in this way until level L_2 in the search tree is reached. At this moment each slave has its own tree for q taxa. The remaining trees for q taxa are stored by master. Each process, i.e. all slaves and master, creates all trees which can be constructed for a given q-taxon subtree. If a complete tree for n taxa with parsimony score smaller than current upper bound is found it is sent to master as a current best solution. Master updates the upper bound and sends it to all slaves. When all trees being extensions of a given q-taxon subtree are checked slave sends a request to master. Master sends to this slave the maximum parsimonious q-taxon tree from the set of unextended subtrees.

3.2 Version 2

This version of the algorithm also has the master-slave structure. At the beginning process master searches an upper part of the search tree, i.e. a part between level 0 (root) and some level L_1. The searching strategy for this part is depth first search (DFS) like in the sequential version. When master reaches level L_1 it extends a current node of the search tree according to breadth first search strategy (BFS), i.e. it constructs all subtrees for taxa corresponding to level L_1. Then, the subtrees are put into a queue and master continues searching the upper part of the tree. Simultaneously, slaves, which are idle at the beginning

of the algorithm, send request for master. As an answer to the request master sends a subtree from the queue. When received by slave the subtree is extended by the remaining taxa until all of them are in the tree or the upper bound is reached. If the final tree for n taxa has value of parsimony score better than the current upper bound, it is send to master as a new solution and master sends this value to all slaves as a new upper bound. When the whole upper part of the search tree is checked by master it behaves similarly to slaves and extends the subtrees from the queue.

3.3 Version 3

The third version is similar to the second one. The difference is that in the third one master does not stop at level L_1 and continues searching the tree according to DFS strategy until it receives a request from a slave. At this moment it sends to this slave its current node of the search tree and continues searching the tree without the subtree rooted in the node already sent. When slave receives a node it extends a corresponding phylogenetic subtree similarly like in version 2 of the method. There are two distinguished levels in the search tree: L_1 and L_2. Slaves' requests are not accepted by master above level L_1 and below L_2. If they would be accepted above L_1 a granularity of task were to small and slaves obtained to big tasks which would lead to poor load balancing. On the other hand, if the requests would be accepted below L_2 the granularity were to big, the tasks were to small which could result in very intensive communication.

Table 1. Computation times in seconds for Version 1 of the algorithm and $L_2 = 5$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	414.11	412.86	381.48	413.89	32.79	0.43
3	138.88	138.71	129.73	138.86	11.01	0.65
5	83.89	83.68	77.98	84.47	7.76	0.87
7	82.70	82.59	75.71	82.69	5.91	1.09
10	56.60	57.23	51.52	57.59	6.01	–
13	58.34	58.32	50.47	–	6.28	1.75
16	33.90	34.22	32.48	35.76	6.29	2.01

4 Computational Experiments

The parallel algorithms described in the previous section have been implemented in C++ language using MPI library and tested on SUN Fire 6800 machine in Poznań Supercomputing and Networking Center.

The algorithms have been tested on six instances *A, B, C, D, E,* and *F.* All of them consist of 10 sequences of length 100 nucleotides. Instances *A* and *B* have been generated using pseudo-random number generator. Nine sequences

Table 2. Computation times in seconds for Version 1 of the algorithm and $L_2 = 6$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	414.47	413.04	378.50	413.18	31.99	0.44
3	142.29	142.28	129.96	142.07	15.10	0.65
5	86.51	86.01	80.09	86.26	12.23	0.87
7	63.03	62.46	58.33	62.72	12.13	1.10
10	46.27	45.89	42.88	47.42	12.42	1.42
13	38.70	38.19	35.36	38.38	13.02	1.77
16	34.34	34.11	32.57	34.63	14.26	2.08

Table 3. Computation times in seconds for Version 1 of the algorithm and $L_2 = 7$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	413.44	412.90	374.10	413.23	32.36	0.45
3	163.83	163.95	153.66	162.36	28.15	0.66
5	111.30	109.80	109.31	110.20	28.30	0.89
7	101.56	98.12	106.49	99.51	28.54	1.11
10	100.09	101.38	107.22	99.56	28.85	1.43
13	102.56	99.54	107.57	99.00	28.86	1.77
16	103.31	99.71	105.36	101.02	36.83	2.07

Table 4. Computation times in seconds for Version 2 of the algorithm and $L_1 = 5$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	425.41	423.29	396.12	429.61	35.42	0.51
3	147.28	146.43	137.37	146.90	14.83	0.72
5	89.08	90.35	83.69	89.51	11.56	0.95
7	65.35	65.81	61.38	64.18	10.46	1.16
10	47.87	48.15	45.33	47.68	10.78	1.50
13	39.55	38.46	36.99	39.32	11.18	1.83
16	34.69	35.39	33.17	35.03	11.51	2.16

of instance *C* have been generated by pseudo-randomly shuffling the tenth sequence. Instance *D* have been generated in a similar way. Instance *E* consists of hundred-nucleotide fragments of some RNA viruses (taken from the beginnings of their genomes), and instance *F* consists of such fragments of Hepatitis C Virus.

In Tables 1, 2 and 3 there are shown computation times for the first version of the algorithm and values of L_2 equal to 5, 6 and 7, respectively.

As one can notice, for the three tested values of L_2 the algorithm is most effective for $L_2 = 5$. From the nature of the considered problem it follows that the number of phylogenetic trees which can be constructed at a given level l of the search tree grows very fast. Hence, there is a big difference between a

Table 5. Computation times in seconds for Version 2 of the algorithm and $L_1 = 6$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	425.43	424.73	398.25	426.39	35.64	1.99
3	167.06	165.99	158.76	167.34	26.53	2.21
5	107.14	104.99	105.75	106.49	26.14	2.42
7	92.54	92.24	94.95	93.78	26.38	2.67
10	94.21	92.95	95.29	92.56	26.58	2.98
13	94.95	92.45	94.89	94.34	26.97	3.32
16	95.14	93.78	95.82	94.68	27.49	3.62

Table 6. Computation times in seconds for Version 2 of the algorithm and $L_1 = 7$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	433.55	431.55	402.29	431.77	45.37	5.81
3	431.64	431.68	403.44	431.33	45.60	5.97
5	436.39	439.57	407.60	434.96	45.79	6.10
7	439.17	432.89	408.82	433.11	46.46	6.46
10	432.18	433.67	404.46	434.19	47.01	6.79
13	433.69	436.39	407.17	433.68	47.42	7.12
16	440.49	436.39	407.09	441.30	47.39	7.44

Table 7. Computation times in seconds for Version 3 of the algorithm and $L_1 = 3, L_2 = 5$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	414.29	413.79	386.89	413.91	34.33	0.43
3	189.79	189.21	171.63	189.30	18.10	0.65
5	118.59	118.44	109.83	107.15	12.32	0.87
7	102.72	102.59	94.07	102.99	9.17	1.09
10	71.39	71.08	63.63	71.57	10.36	1.42
13	50.55	58.39	56.14	64.39	8.55	1.75
16	43.50	41.95	43.96	42.61	6.45	2.08

number of tasks sent to slaves for values of L_2 differing by 1. For the bigger values of L_2 the number of tasks processed by slaves grows rapidly and also the number of messages exchanges by master and slaves increases very fast. This is probably the reason of the weaker results of the first version of the algorithm for values of L_2 equal to 6 and 7 in comparison to $L_2 = 5$. Moreover, for instance *F,* which appeared to be rather easy for the algorithm, computation times increase when the number of processors increases. This, rather surprising phenomena can be also explained as a results of relatively expensive communication between processes. In this case, since the instance is easy and one processor is able to solve

Table 8. Computation times in seconds for Version 3 of the algorithm and $L_1 = 4, L_2 = 6$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	414.16	413.73	386.13	413.90	34.32	0.43
3	163.67	163.34	162.62	163.05	21.57	0.65
5	107.19	100.29	110.82	100.96	21.93	0.87
7	89.36	87.86	86.46	76.36	–	1.08
10	70.25	52.81	70.37	66.13	22.84	1.42
13	–	55.65	46.64	59.40	22.94	1.75
16	55.93	52.52	50.36	44.32	20.36	2.07

Table 9. Computation times in seconds for Version 3 of the algorithm and $L_1 = 5, L_2 = 7$

No of processors	Instances					
	A	*B*	*C*	*D*	*E*	*F*
1	414.36	413.79	386.26	414.14	34.31	0.43
3	216.67	217.86	217.70	213.74	29.66	0.65
5	150.28	141.05	187.21	147.57	29.76	0.87
7	129.56	127.36	189.02	136.10	30.62	1.09
10	138.33	139.55	180.69	137.02	–	1.42
13	145.94	137.38	–	134.71	31.09	1.75
16	132.93	–	187.87	139.63	31.27	2.08

it in a very short time the cost of communication in a multiprocessor version exceeds the profit of the parallelization.

Computation times for the second version of the algorithm are shown in tables 4, 5 and 6.

It is easy to see that here the influence of the value of L_1 on the effectiveness of the algorithms is similar like for version 1.

Tables 7, 8 and 9 show computation times for version 3 of the algorithm. This version appears to be more effective than version 2.

The computational experiments showed that communication in the machine used for testing the algorithms can have surprisingly big influence on the effectiveness of the algorithms. One can expected that version 1 of the algorithm could be the slowest one because of the greatest amount of communications. But it appeared to be the most effective one. This phenomenon follows probably from the way of message passing used in the implementation of the algorithms (in some cases it happened that master process performed big amount of computations before it could sent some tasks to idle master processes). That is probably also the reason of the fact that an efficiency of the algorithms is getting worse for bigger number of processors used in the experiment.

Moreover, from the nature of the considered problem it follows that subproblems represented by nodes being on some levels l and $l+1$ in the search tree are

of very different sizes. So, it is very important which nodes, i.e. being on which levels are send by master to slave processes.

5 Conclusions

In the paper three parallel versions of branch and bound algorithm for phylogenetic tree construction have been presented. All of them have master-slave structure. The parallelization has been done on a level of a strategy of traversing the search tree. In the tree each node corresponds to some phylogenetic tree for a subset of a set of taxa being an instance of the problem. To each of the processes some nodes of the search tree are assigned and they solve phylogenetic tree construction problem for some small set of organisms. Such an approach reduces a number of messages sent by slave processes since they must inform master process only about the obtained tree in the case if it is the best one at a given stage of computation (versions 2 and 3) and also about some threes represented by leaves being successors of the node received from master (version 1).

To summarize, the parallel algorithms for phylogenetic tree construction are the area of algorithmic studies not very deeply explored. Parallel algorithms are especially important for evolutionary studies, since computation times for the real biological problems considered in this are are often very long and the algorithms used (the sequential ones) are usually heuristic. Since parallel or distributed systems have became more available recently, good parallel algorithms could accelerate research in this area of biological sciences. The algorithms presented in the paper seem to be good starting point for further research on exact parallel algorithms for evolutionary history reconstruction and they can have practical importance when run on a machine with great number of processors. The important feature of the algorithms is also they independence on a specific machine (it should be possible to run them on other parallel machines with MPI library without serious changes in a source code).

References

1. Fitch, W. M.: Toward defining the course of evolution: minimum change for a specified tree topology. Systematic Zoology **20** (1971) 406–416
2. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
3. PHYLIP Home Page: http://evolution.genetics.washington.edu/phylip.html
4. Setubal, J., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)
5. Waterman, M. S.: Introduction to Computational Biology. Maps, Sequences and Genomes. Chapman & Hall, London (1995)

A Hierarchical Model of Parallel Genetic Programming Applied to Bioinformatic Problems

Julien Frey, Robin Gras, Patricia Hernandez, and Ron Appel

Swiss Institute of Bioinformatics, CMU, 1 rue Michel Servet,
CH-1211 Geneva 4, Switzerland
julien.frey@isb-sib.ch

Abstract. Genetic Programming (GP), an evolutionary method, can be used to solve difficult problems in various applications. However, three important problems in GP are its tendency to find non-parsimonious solutions (bloat), to converge prematurely and to use a tremendous amount of computing time. In this paper, we present an efficient model of distributed GP to limit these general GP drawbacks. This model uses a multi-objective optimization and a hierarchical communication topology.

1 Introduction

Genetic programming (GP) [1] is a technique for automatic discovery of programs to solve difficult problems. In GP, programs are expressed as parse trees, which are composed of functions and terminals (programmatic elements) appropriate to the given problem. GP provides a way to search the space of all possible programs to find a solution of the problem. For this task, GP applies evolutionary algorithms (EA) [2] to a population of programs. EA are stochastic search methods inspired by the natural evolution. Given a description of the problem to be solved (a fitness function) and a set of programmatic elements, GP starts with a population of random programs. Then this population evolves by applying iteratively a step of evaluation and a step of reproduction. The step of evaluation consists in executing each programs of the population in order to compute its fitness value. Then the step of reproduction uses the fitness information to recombine the programs using mechanisms borrowed from genetic crossover, mutation and natural selection. The iterative process continues until a certain termination criterion has been satisfied. One drawback of GP is the considerable computing time that can be required to produce a solution. For most practical problems, the step of evaluation requires a substantial amount of processing time. For example, when GP is used to solve a problem by learning, the fitness cases (training environments) are very important to improve the robustness of the solution generated. The more numerous and various fitness cases are and the more robust solution is but the more time consuming the evaluation is. The chance of success in applying GP also depends on the adequacy of the parameters (the population size, the iteration number, and the fitness cases) in relation

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1146–1153, 2004.
© Springer-Verlag Berlin Heidelberg 2004

to the difficulty of the problem. Thus, to improve the efficiency of GP, more computing power is required. However a parallel algorithm approach can be applied to the classical GP for reducing processing time. Parallel Genetic Programming (PGP) can be classified into three different models: **(1) Master-slaves PGP.** This model uses a single global population and the step of evaluation is done on different processors. The master performs the standard sequential GP process. During the step of evaluation the slaves collect an equal number of individuals from the master, evaluate them, and return the fitness values to the master. Furthermore, the step of reproduction may also be done in parallel. This model involves a high number of exchanges between processes and is very suitable for a shared memory computer. **(2) Fine-grained PGP** [3]. In this model, the population is divided into a large number of small subpopulations. Each subpopulation is assigned to an independent process. Subpopulations are spatially distributed on a low-dimensional grid and individuals interact locally within a small neighborhood. This model is suitable for machines consisting in a great number of processors connected with a specific high-speed topology (massively parallel architectures). **(3) Coarse-grained PGP** [4]. The population is divided into several subpopulations. Each subpopulation applies the classical GP process independently with its own parameters. When a certain criterion is satisfied, each subpopulation sent a fixed number of individuals to one ore more subpopulation(s). In return, an equal number of individuals are received from one or more subpopulation(s). This migration can occur either asynchronously or after the synchronization of all processes. This model, so called *island model,* limits the premature convergence (finding the local optimum instead of the global optimum).

Two other important problems in GP are bloat and the general EA problem of premature convergence. Bloat [5,6] is when the trees considered during a GP run grow in size and become larger than what is necessary to represent good solutions. Bloat slows down the search by increasing evaluation time and decreasing the probability that genetic operation will change the operational part of the tree. It also increases the over fitting during the learning process. The premature convergence [6] is the rapid loss of diversity in the population leading to the stagnation in a local optimum. In this paper we present a more efficient coarse-grained PGP, called *pyramidal model,* which allows limiting the drawbacks of the classical GP process.

2 Method

The *pyramidal model* (PyM) uses a more efficient communication topology between subpopulations of coarse-grained PGP. In PyM, subpopulations are distributed in several superimposed layers. The number of subpopulations contained in each layer increases according to the depth. The communication occurs between subpopulations of different layers and in a unidirectional way. Thus, each subpopulation sends programs to a subpopulation of the upper layer and receives programs from subpopulations of the lower layer. This communication

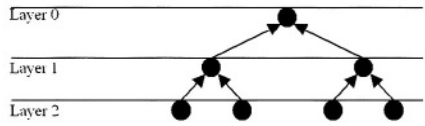


Fig. 1. Communication topology of *pyramidal model* coarse-grained PGP

topology can be viewed as a pyramidal communication between subpopulations (Fig. 1 shows the communication topology used in the *pyramidal model*). The main parameters of each subpopulation are adapted according to the depth of its layer. When the depth of a subpopulation increases, its size increases, the time required by its step of evaluation decreases, and its number of iterations decreases. In that way, the lower subpopulations explore permanently the search space and send promising solutions to the upper subpopulations, which exploit this information. Thus, this communication topology allows separating the step of exploration from the step of exploitation. During the step of exploration, the evaluation of a program is made on some fitness cases and requires little computing time. Thus, the exploration isolates very rapidly promising area of the search space. The best solutions found go up layers in layers where they are combined and where their fitness values are computed on numerous and various fitness cases. We use a Multi-objective Optimization (MO) [6], to deal with the problem of bloat. The MO is the search for multiple solutions, which satisfy the different objectives to different degrees. We use two objectives, a functionality objective (the fitness value or *obj* 1) and a size objective (the number of nodes in the program or *obj2*). An important notion in MO is the dominance. We use a strong non-dominance: program *A* dominates program *B* if $A_{obj1} \geq B_{obj1}$ and $A_{obj2} \leq B_{obj2}$. During the GP process of each subpopulation, each new solution evaluated is integrated into a stack of non-dominated programs. When the maximum number of iterations is reached, all programs contained in the stack are sent and a new GP process begins. The reception of programs is check at the end of each step of evaluation and the received programs are placed in a queue. During each step of reproduction, some programs are withdrawn from the queue and integrated into the population without alteration. The communication between subpopulations is asynchronous. The PyM process is stopped when all subpopulations of the layer 0 have reached their maximum number of iterations.

3 Application

These past few years, the massive sequencing of genomes has provided huge quantities of new proteins. However it is difficult to determine the function of these proteins for which only the primary sequence is known. Indeed the characterization of a protein in laboratory can require several years of research. Sequence homology is traditionally used to predict the function of an unknown

protein. The primary sequence of the unknown protein is aligned with proteins of known function by using programs like CLUSTALW [7] or MoDEL [8], and the function of the most homologous known protein is assigned to the unknown protein. This homology based method gives good results when the sequence homology is high (> 60%) but fails otherwise. An alternative approach to function prediction consists in use only the primary sequence of the unknown protein [9]. This method is based on the fact that a number of sequence features (or sequence attributes) are depending on the function of the protein. A function prediction method based on such attributes can be applied to all proteins when primary sequence is known. These attributes include features such as length, isoelectric point and composition of the polypeptide chain. Our method described here integrates 21 individual attributes calculated using the primary sequence of proteins. Our method predicts whether a protein of known primary sequence is likely to be involved in the transcription mechanism. We have applied our predictive approach to specifically identify transcription factors but the same scheme can be used to predict any other functional classes like hormones, receptors or enzymes. The discriminative impact of only one attribute is not sufficient to predict the function class of a protein. However, if we use an adapted combination of attributes, the discriminative impact could be increased. We have applied the genetic programming to the search of such combinations.

3.1 Terminal Set, Function Set, and Architecture

For our problem, a program is a transcription factor predictor. It is composed of arithmetic operations and conditional operations. The set of terminals, named *T*, consists of the 21 attributes and constants, named *C,* generated between -10^6 and $+10^6$. The set of functions, named *F,* is composed of standard arithmetic operators (addition, subtraction, division and multiplication) and one conditional operator. The figure 2 gives an example of a valid program generated with the sets *T* and *F*. For a protein sequence, the values of its attributes are given as input to the program and a numeric value (bold in the figure 3) is given as output. Then, this value is used to decide whether the protein is likely to function as transcription factor. Indeed, this value is compared with a threshold (bold and italic in the figure 2), which is specific to the program, and a prediction is returned.

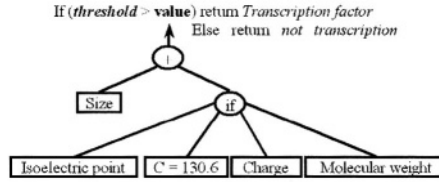


Fig. 2. An example of program generated by using the set *T* and the set *F*.

3.2 Fitness Measure and Training Data

The fitness function is the driving force of the evolution in GP. It measures the adequacy of a program to the given problem. For our problem, the fitness indicates how a program is able to predict correctly the function of proteins. The fitness of a program is computed by comparing the values returned by the program for the positive proteins and the values returned for the negative proteins. The positive proteins are the proteins of the training set which are known to function as transcription factor. The negative proteins are the proteins of the training set which are known to not function as transcription factor. We have used the Kolmogorov-Smirnov test (KS-test) to carry out this comparison. The KS-test tries to determine if two datasets differ significantly. The KS-test has the advantage of making no assumption about the distribution of data, it is non-parametric and distribution free. The KS-test uses the maximum vertical deviation between two cumulative distributions as the statistic *D*. For our problem, we want to obtain the program with the highest value *D*. Indeed, higher the value *D* is, more separated the two distributions are, more discriminating the value returned by the program is, and more occurred the prediction given by the program is. The value *D* lies between 0 and 1.The fitness value of a program is computed in the following several steps. First, the value of each positive protein is computed using the program. Second, these values are sorted in ascending order. The same steps are applied for the negative proteins. The KS-test is applied on the two datasets previously obtained. Thus, the statistic *D* is used as the fitness of the program and the value corresponding of the statistic *D* is used as the threshold of the program. An example is given in figure 3. For testing our method, we have used a learning set composed of proteins extracted from the SWISS-PROT database [10]. We have used 8024 negative human proteins and 920 positive human proteins. These data are divided into two sets: A training set used during the GP process to evaluate the programs and a control set used after the GP process to test the programs. The two sets are composed of 4012 negative proteins and 460 positive proteins.

4 Results and Discussion

4.1 Multi-objective Test

We have tested the efficiency of the multi-objective optimization. For that, we have compared the quality of the solutions generated by two algorithms differing only by their methods of selection. The first algorithm, named *Tournament Selection* (TS), selects the individuals of the current population by using the tournament selection of size 2 [11]. This algorithm aims at maximizing the fitness value of programs during generations. The second algorithm, named *Multi-Objective* (MO), uses two criteria of selection. We have choice the Multi-Objective Genetic Algorithm (MOGA) proposed by Fonseca and Fleming [12] as selection scheme. It aims at maximizing the performance of the programs (fitness value) and at minimizing their size (number of nodes) during generations. We have used the

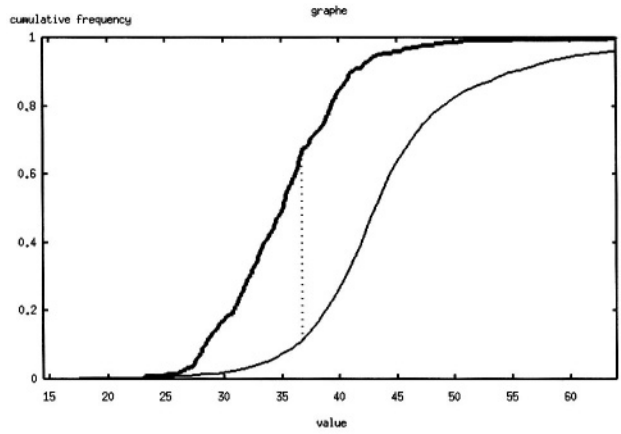


Fig. 3. Cumulative distribution of the 460 positive protein values (bold curve) and cumulative distribution of the 4012 negative protein values (simple curve) obtained with a program. The dashed line represents the statistic D (threshold value = 36.8)

computational time as criterion of termination for the GP process. Figure 4 shows the evolution of the quality of the solutions generated by the two algorithms according to the computational time for populations of size 400 (similar results not shown here have been obtained for other population sizes). These results are averaged over 100 runs. The two algorithms generate solutions of equivalent quality with a little advantage for the MO algorithm. However the average size of the solutions obtained by the MO algorithm is lower than those obtained by the TS algorithm. Indeed, at the end of the GP process and for a population size of 400 individuals, the average size of the solutions generated by the MO algorithm is of 87 nodes against 801 nodes for the TS algorithm. Thus at equal time, the MO algorithm has found solutions more parsimonious and of higher quality than those generated by the TS algorithm.

4.2 Coarse Grained PGP Method

We have tested the efficiency of the coarse grained PGP method described in this paper. For that, we have compared the quality of the solutions generated by a serial algorithm and a parallel algorithm. The termination criterion of the process is the number of evaluated proteins. For the serial algorithm, we have used a population of 400 individuals and a training set of 4472 proteins. For the parallel algorithm, we have used 4 subpopulations connected with the hierarchical communication topology described previously. We have used one subpopulation in the layer 0. This population of size 200 uses a training set of 4472 proteins. The three other subpopulations are in the layer 1. Each of them is composed of 500 individuals and uses a training set of 500 proteins. They send the non-dominated solutions at the population of the layer 0 when they

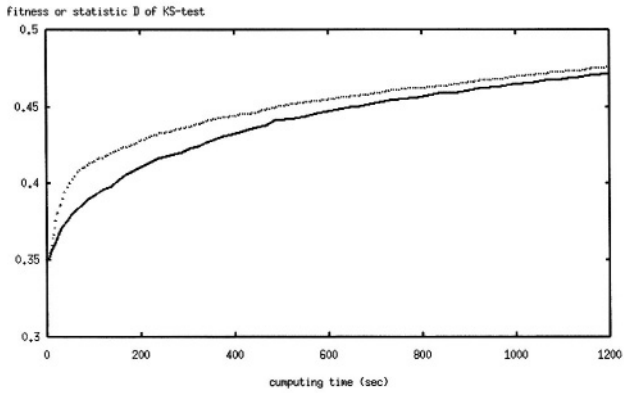


Fig. 4. Fitness of the solutions generated by the MO (*dashed curve*) and TS (*solid curve*) algorithms according to the computational time for a population of 400 individuals. Results averaged over 100 runs.

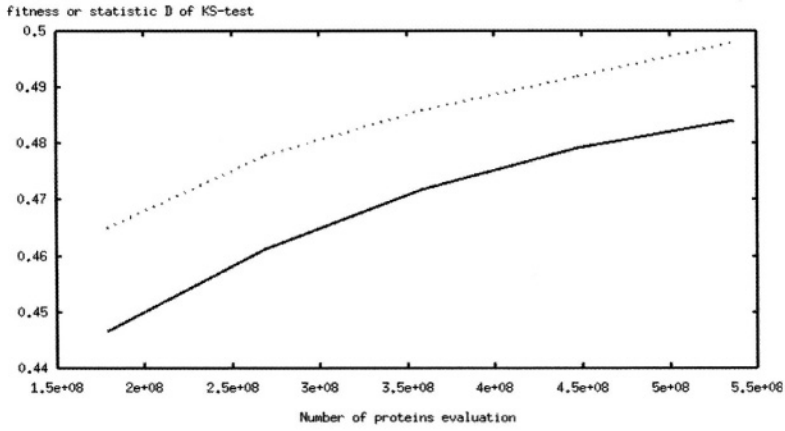


Fig. 5. Fitness of the solutions generated by the serial algorithms (*solid curve*) and the parallel algorithm (*dashed curve*) according to the number of proteins evaluation. Results averaged over 100 runs

reach 20 generations. The figure 5 shows the results averaged over 100 runs. We can see that the parallel algorithm gives better results than the serial algorithm for a given number of fitness evaluations allowing a supra-linear acceleration of the parallel model. However it is difficult to compare the two methods in ideal conditions because the adaptation of each parameter is long and tiresome. In further research, we want to test the island model versus the PyM model.

5 Conclusion

This paper presents a multi-objective coarse-grained PGP method as a general approach to avoid bloat, to promote diversity and to separate the step of exploration from the step of exploitation in GP. The hierarchical topology of this method allows the use of the most efficient parameters for the different steps of the GP process. This approach appears to be promising for difficult combinatorial problems that require a supervised learning like the discovery of a robust function predictor.

References

1. Koza, J.: Genetic Programming: on the programming of computers by means of natural selection. The MIT Press. (1992)
2. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)
3. Manderick, B., Spiessens, P.: Fine-grained parallel genetic algorithms. In: The Third International Conference on Genetic Algorithms. (1989) 428–433
4. Cantu-Paz, E.: Topologies, migration rates, and multi-population parallel genetic algorithms. In: Proceeding of GECCO. (1999) 91–98
5. Soule, T., Foster, J.: Effect of code growth and parsimony pressure on populations in genetic programming. Evolutionary Computation **6(4)** (1999) 293–309
6. De Jong, E., Waston, R., Pollack, J.: Reducing bloat and promoting diversity using multi-objective methods. In: Proceeding of the Genetic and Evolutionary Computation Conference. (2001)
7. Thompson, J., Higgins, D., Gibson, T.: CLUSTAL W: improving the sensibility of progressive multiple sequence alignment through sequence weighting, positions-specific gap and weight matrix choise. Nucleic Acids Research **22** (1994) 4673–4680
8. Hernandez, D., Gras, R., Lisacek, F., Appel, R.: MoDEL: Inference de motifs avec un algorithme evolutionniste. In: Proceeding of JOBIM. (2002) 265–267
9. Jensen, L., Gupta, R., Blom, N., Devos, D., Tamames, J., Kesmir, C., Nielsen, H., Staerfeldt, H., Rapacki, K., Workman, C., Anderson, C., Knudsen, S., Krogh, A., Valencia, A., Brunak, S.: Prediction of human protein function from post-translational modifications and localization features. J. Mol. Biol **319** (2002) 1257–1265
10. Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M., Estreicher, A., Gasteiger, E., Martin, M., Michoud, K., O'Donovan, C., Phan, I., Pilbout, S., Schneider, M.: The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. Nucleic Acids Res. **31** (2003) 365–370
11. Blickle, T., Thiele, T.: A comparison of selection schemes used in genetic algorithms. Technical Report 11, TIK (1995)
12. Coello Coello, C., Van Veldhuizen, D., Lamont, G.: Evolutionary Algorithms for Solving Multi-objective Problems: Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (2002)

A Fault-Tolerant Protocol for Resource Allocation in a Grid Dedicated to Genomic Applications*

Michel Hurfin[1], Jean-Pierre Le Narzul[1,2], Julien Pley[1], and Philippe Raïpin Parvédy[1]

[1] IRISA – Campus de Beaulieu, 35042 Rennes, France
{hurfin, jlenarzu, jpley, praipinp}@irisa.fr
[2] GET/ENST Bretagne – Campus de Rennes
35512 Cesson-Sévigné, France

Abstract. The major aim of a Grid is to federate several powerful distributed resources within a single virtual entity which can be accessed transparently and efficiently by external users. As a Grid is a *distributed* and *unreliable* system involving heterogeneous resources located in different geographical domains, fault-tolerant resource allocation services have to be provided. In particular, when crashes occur, tasks have to be reallocated quickly and automatically, in a completely transparent way from the users' point of view.
This paper presents PARADIS, a system based on a Consensus building block that has been designed and implemented in a Grid dedicated to genomic applications. These time-consuming applications can be split up into a huge number of independent tasks which can be allocated independently on different domains.

1 Introduction

The major aim of a Grid is to federate several powerful distributed resources (computers but also data storage facilities) within a single virtual entity which can be accessed transparently and efficiently by external users. In this new business model, the administrators of the Grid have now the responsibility of ensuring that sufficient resources are deployed to meet the growing workload demands of the users. Therefore distributed resource management softwares have to be developed to provide acceptable response time and to address efficiently, among other issues, *resource allocation* and *fault tolerance.* As the Grid is shared by several users, load balancing strategies have to be implemented to distribute the tasks among the resources. When crashes occur, tasks have to be reallocated quickly and automatically, in a transparent way from the users' point of view.

A Grid is a *distributed* system involving heterogeneous resources located in different geographical domains that are potentially managed by different organizations. Therefore, most of the difficulties encountered when designing Grid

* This work was supported by the French Ministry of Research - Project GénoGRID of the program ACI "Globalization of computer resources and data".

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1154–1161, 2004.
© Springer-Verlag Berlin Heidelberg 2004

software are related to well known problems in distributed computing. Some of these problems (observation of the global state of a distributed system, crash failure detection, ...) are hard to solve when the system is asynchronous. An asynchronous system is characterized by the lack of a global synchronized clock, and puts no bound on the transmission delay of messages[1]. In a large scale Grid, the only reasonable assumption is to consider the system as being asynchronous even if some subparts are composed of computers connected through local area networks and thus can be considered as "synchronous island in an asynchronous sea". A first contribution of this work is to define clearly this two levels architecture. Within a synchronous subnetwork, called herein a *domain,* the problems are quite easy to solve. Interactions between domains are much more complex to manage. The second contribution of this paper is to address the two major issues previously mentioned, namely resource allocation and fault-tolerance, in an homogeneous way. We propose to solve both problems using a slightly modified group concept [11]. More precisely, all distant interactions between domains are managed by a small group of registered processors (exactly one per domain). Each member of the group acts as a *proxy* for its own domain and interacts with the other members of the group to build consistent observations of both workloads in each domain and composition of the group.

Several powerful systems have been developed to provide a Grid infrastructure. Globus [7] and UNICORE [1] are probably the most famous ones. The facilities provided by these two well-established systems address several issues not discussed in this paper. Our goal is to study the resource allocation problem in an asynchronous and unreliable setting. Until now, failures are addressed within the existing systems only through the definition of a centralized failure manager that is in charge of detecting and notifying the observed crashes. We claim that this approach is not the best solution because it creates a single point of failure. Whereas these systems have been developed to cope with general applications, our proposal copes with a particular class of application and exhibit a particular agreement component which can be used to handle all the interactions between domains and to implement, in a modular way, a fault-tolerant resource allocation protocol.

In a general setting, tasks cannot be allocated independently because they may interact together during their executions. The proposed solution does not cover this particular problem and focuses on a specific class of applications. We consider applications which can be decomposed into several independent tasks. This criteria is satisfied by many biological applications. In this particular context, we contribute actively to the design and development of an experimental Grid for genomic applications (project called " ACI GénoGRID" [9] founded by the French Ministry of Research). This project aims to conceive a Grid devoted to time-consuming genomic computations. The computing and data resources used to create the Grid belong to genomic or bioinformatics centers dispatched in the western part of France. With the help of biologists, our purpose is to val-

[1] Such a bound may exist but either this value is unknown or the known value is so high that it cannot be used to define reasonable timeouts.

idate our approach with at least three genomic applications (intensive sequence comparison, protein threading [10], and detection of repeated sequences). The execution of one of these applications can split up into a huge number of independent tasks. Consequently, the allocation of the tasks to the resources can also be done independently. In our approach, all the interactions between domains aim exclusively to reach agreements.

This paper focuses on the fault tolerant protocol proposed to solve the resource allocation problem. The mechanisms developed to offer a secure and interactive access to the Grid (through a standard WEB site) are not discussed. Also, we do not describe the applications used to validate our implementation. Instead we precise a few programming rules that have to be satisfied by any application conceived to be executed on our experimental Grid.

This paper is organized as follows. Section 3.1 presents the consensus-based mechanisms that manage the Grid membership and the ressource allocation. Section 2 discusses the two levels architecture. Section 3.2 describes the programming rules that have to be respected. Section 3 presents the operating system called PARADIS and focuses mainly on the resource allocation protocol. Finally, Section 4 concludes this paper.

2 Architecture

The Grid is deployed over the Internet. Even if this network is globally asynchronous, it is composed of synchronous subnetworks called *domains* (in practice, these domains correspond to LANs). To improve the fault tolerance and the efficiency of computations on the Grid, we try to benefit as much as possible from the synchronous properties of communications within a domain and to avoid as much as we can the (asynchronous) communications between domains. In order to provide an easy access to the Grid from anywhere, the applications can be launched through web portals.

2.1 The Domain Level

A domain is a set of heterogeneous nodes which communicate in a synchronous way. A node can be either a resource of the Grid or a machine devoted to control activities. The management of the domain is organized according to the master-slave model: in each domain, a single node named the *master* is selected to manage all the other nodes (named the *slaves*). In particular, the master has to schedule all the tasks carried out in its domain. At any time, the master can check the loads of its slaves. This information is used to compute an appropriate local scheduling of tasks. The composition of the domain is dynamic: nodes can join or leave the domain by requesting to the master.

Nodes fail only by crashing. As a domain is synchronous, all the crashes can be detected in a reliable way. When the crash of a resource is detected by the master, the master distributes again the tasks (previously allocated to the faulty node) among the remaining resources. The crash of the master has also to be

tolerated. Some nodes (the *heirs*) are preselected to replace the master when it disappears. Thanks to a leader election protocol, a single heir is allowed to replace the previous master. If no node can replace the master, all the domain becomes unavailable. Of course, during the computation, the heirs have to keep track of the whole knowledge of their master. As the role of these backups is just to ensure that there is not a single point of failure per domain, we will no more discuss about them in the remaining sections.

2.2 The Grid Level

The Grid is an asynchronous network connecting different domains. To avoid a flood of the Grid, only one node per domain is allowed to communicate with the other domains, this node is called the *proxy.* All the proxies of the Grid constitute a group. In practice, a single node per domain acts both as the proxy and the master. Like the composition of a domain, the composition of the network of domains is also dynamic. Through invocations of the *join* and *leave* operations, the administrator of a domain can decide (independently from the other administrators) to add or remove his own domain from the Grid whenever he wants (maintenance and repair, alternating periods of private and public use of the local resources, ...). A domain is unavailable if no node of this domain can act as a proxy/master (occurrence of crash failures) or if the domain has been disconnected from the Grid (occurrence of communication failures). On one hand, join and leave operations are intentional and broadcast to all the members. On the other hand, evolutions caused by occurrences of failure are unpredictable and are not necessarily observed by all the members of the group. In the proposed solution, each proxy is coupled with a failure detector module which maintains a list of domains that it currently suspects to be unavailable. A *Group Membership* service will ensure that all the proxies, that are currently members of the group, are consistent with the past history of the group, namely, the *join* and *leave* operations already executed and the failures suspected to have occurred.

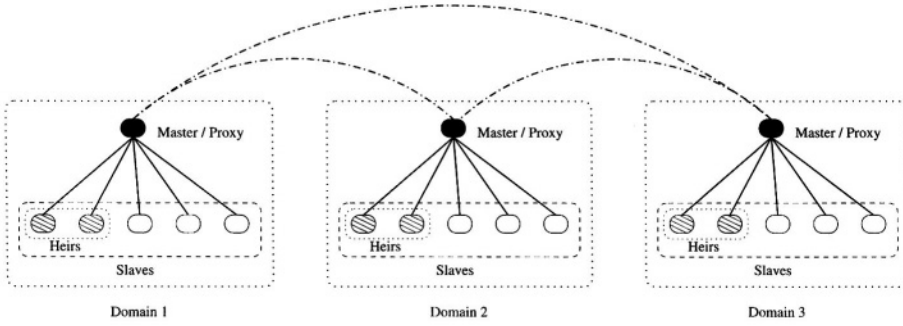


Fig. 1. Grid Hierarchy

3 The Operating System PARADIS

3.1 A Consensus-Based Approach

In the proposed solution, all the proxies communicate together only to reach agreements. Each agreement allows them to share a common view of the evolution of the Grid: decisions are required to solve, despite failures, the group membership problem [8] and the resource allocation problem. Agreement problems can be solved using a consensus based approach. Unfortunately, the consensus problem [3] is actually impossible to solve in a deterministic way in asynchronous distributed systems when even a single process may crash [6]. To circumvent this impossibility result, several approaches have been investigated. One of them is based on the concept of unreliable failure detectors [3] that will observe the availability of remote domains. In an asynchronous system, it is impossible to implement perfect failure detectors. An available domain can be mistaken for a faulty one by some proxy [3]. Failure detectors can be classified according to the properties (completeness and accuracy) they satisfy. A class of failure detectors denoted $\diamond\mathcal{S}$ is of particular interest because this class has been proved to be the weakest one that allows to solve the consensus problem. The proposed solution assumes an underlying failure detector of the class $\diamond\mathcal{S}$. This class is defined by the following completeness and accuracy properties: Any faulty domain is eventually suspected (strong completeness) and there is a time after which there is a correct domain that is no longer suspected (eventual weak accuracy). Relying on $\diamond\mathcal{S}$ failure detectors to solve agreement problems assumes that a majority of domains within the group remains available.

In a recent past, several $\diamond\mathcal{S}$-based consensus protocols have been designed. Obviously, rather than selecting a particular consensus protocol, it is more efficient to provide a generic and adaptative solution that can be customized to cope with the characteristics of the environment as well as the characteristics of the reliable distributed abstractions that have to be provided [5]. This idea has led us to design and implement ADAM, a component-based library of agreement abstractions [4] based on a generic agreement component which can be configured to solve efficiently various agreement problems. In the current implementation, all the interactions between proxies are handled by a group-based toolkit (called EDEN) which has been developed using ADAM. ADAM is based on EVA [2], an event-based framework for developing distributed abstractions and high-level communication protocols. Using a generic agreement service (*i.e.* the library ADAM) is at the core of the proposed solution. As indicated previously, a consensus building block allows to solve the membership problem. It is also a key service to find a correct assignation of the tasks submitted by the users to the resources of the Grid.

3.2 Design of Applications

The Grid is devoted to genomic applications which consume a lot of CPU time. To improve their efficiency, these applications are parallelized. As indicated in

Protocol 1 PARADIS

1: application A is submitted on the Grid by user U via a portal
2: each proxy bids for main task T_A. The lowest bid, given by proxy P, carries T_A off
3: T_A runs on proxy P and generates some elementary tasks
4: **while** T_A generates elementary tasks **do**
5: every generated task is submitted to the Grid
6: **for** each elementary task t **do**
7: every proxy bids. Proxy P' which has given the lowest bid carries t off
8: P' places t is on a slave S of its domain
9: slave S executes t
10: t ends. S sends its result to P'
11: proxy P fetches the result of t from P'
12: **end for**
13: **end while**
14: T_A ends
15: U is notified via a portal that the results of A are available at a given place
16: U fetches the results

the Introduction, we focus mainly on the applications whose executions could lead to launch many independent tasks in parallel. Some programming rules have been defined to cope with this particular aspect. More precisely, the code corresponding to an application is divided into at least two different codes. A unique task (called the main task) is in charge of submitting sequentially the execution of elementary tasks. These elementary tasks are independent and thus can be executed in parallel in any domains. The main task as well as the elementary tasks have to be registered before. Thus the submission process (implemented by two primitives called *submit* and *wait*) requires only to provide a script file (use to start the execution of the elementary task) and the input files (used as input parameters within the script). We assume that each elementary task produces a single result file. At any time, the main task can ask if the result of a particular task is locally available. It can also ask to kill one of the submitted tasks. We provide an application programming interface (API) to design the main task.

3.3 PARADIS

Executing System: In each domain and for each task that has to be allocated[2], the proxy considers all the resources located in its domain, first, to test if a resource is able to execute the task (presence of the data and software, ...) and, second, to estimate (based on the power and current load of a compatible resource) the time required to complete the execution of the task. Actually, these estimations are provided by a bid mechanism which can be customized to implement various load balancing strategies: one can try to minimize either

[2] In fact, tasks are not allocated one by one but in bunches. This allows to tune the granularity of the allocation process when the duration of a task is too short compared to the cost of the allocation protocol.

the response time or the maximum completion time or the total idle time(In the following, we consider a particular bid mechanism that minimize the response time). After having selected the most adequate resource and determined a corresponding bid, the proxy will participate to an agreement protocol to fix unanimously the identity of the domain which seems to be the most appropriate to execute the task. When failures occur, tasks are reallocated automatically.

When a user U launches an application A on the Grid via a portal (1), his request is broadcast to every proxy. Each of them has to estimate whether it is, or not, the most appropriate one to run the main task T_A corresponding to this application. The bid given by a proxy corresponds to the waiting time before the request can be processed. The lowest bid *wins* and gets the main task T_A (2). To ensure that there is one and only one winner (denoted P in our algorithm), we use the consensus mechanism provided by the middleware EDEN.

During its execution T_A generates some streams of tasks(3). For each task t, the same protocol is respected: as it has been done for the main task T_A, each elementary task t is submitted on the Grid (5) and the proxies bid on it. However, now, one bid does not concern the waiting time before execution on the proxy, but the waiting time before the end of execution on its domain *i. e.* on the first resource of its domain that should complete the execution of t, including the waiting time before the execution starts. Once all the bids have been collected, proxy P' that has proposed the lowest one gets t (7-8). When the node S that had been selected by P' to execute t is available, it starts executing t (9). When S has completed the execution of t, it sends the result to P' (10) which notifies proxy P that t has been executed. Then, P fetches the result from P' (11).

This same scheme is repeated for each stream of tasks generated by T_A until the execution of T_A ends (14). Then, proxy P notifies the user U that the result of A is available (15). Finally, U fetches the result of A (16).

Fault Tolerance System: To ensure the fault tolerance of an application, many points are controlled by the system.

- A master knows the tasks of its slaves, if a slave crashes all its tasks are allocated to other slaves. If no slave is available, the concerned tasks are re-submitted to the Grid.
- Like their master, all the heirs know both running applications and running tasks in their domain. If the master crashes, a heir takes its place and hence becomes the master. All the tasks which were managed by the crashed master are now managed by the new one. All the main tasks which were executed by the previous master are re-executed by the new master. Before the new master re-executes one of these main tasks, all the corresponding elementary tasks generated by the previous master are killed by the system.
- All the proxies know all the running applications and all the main and elementary tasks running on the Grid. If a domain is unavailable (is disconnected), all the main and elementary tasks previously allocated to this domain are re-submitted on the Grid.

Optimizations: EDEN enables our protocol to be fault-tolerant, but this has a cost. The huge number of elementary tasks that are generated by some applications make PARADIS inefficient. To circumvent this problem, we aggregate several tasks in one bunch of tasks that will be handled as a single meta-task at the Grid level and then decomposed into several tasks at the domain level. As a single call to EDEN allows to affect several tasks to a domain , the cost of an agreement is divided by the number of allocated tasks. Depending on their durations and their characteristics (known off-line), the number of tasks that compose one bunch can be adapted dynamically during the execution of the main task. Finally, one can note that several strategies of load balancing can be tested, just by replacing the function corresponding to the bid computation.

4 Conclusion

This paper provides an overview of PARADIS. This system is a contribution to the definition of fault-tolerant resource allocation services which can be used in a Grid.

References

1. J. Almond and M. Romberg, The unicore project: Uniform access to supercomputing over the web. *Proceedings of 40th Cray User Group Meeting,* 1998.
2. F. Brasileiro, F. Greve, M. Hurfin, J.-P. Le Narzul, and F. Tronel, Eva: an Event-Based Framework for Developing Specialised Communication Protocols. *Proc. of the 1st IEEE Int. Symp. on Network Computing and Applications,* p. 108-119, 2002.
3. T. Chandra and S. Toueg, Unreliable Failure Detectors for Reliable Distributed Systems. *JACM,* 43(2):225-267, 1996.
4. F. Greve, Réponses efficaces au besoin d'accord dans un groupe. *Ph.D. Thesis,* University of Rennes, november 2002.
5. M. Hurfin, R. Macêdo, M. Raynal, and F. Tronel, A General Framework to Solve Agreement Problems. *Proc. of the 18th IEEE Int. Symposium on Reliable Distributed Systems (SRDS'99),* pages 56-65, 1999.
6. Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *JACM,* 32(2):374-382, 1985.
7. I. Foster and C. Kesselman, "The Globus Project": A Status Report. *Proc. of the 7th IEEE Heterogeneous Computing Workshop,* pp. 4–19, 1998.
8. F. Greve, M. Hurfin, M. Raynal, and F. Tronel, Primary Component Asynchronous Group Membership as an Instance of Generic Agreement Framework. *Proc. of the 5th Int. Symposium on Autonomous Decentralized Systems* pages 93-100, 2001.
9. D. Lavenier, H. Leroy, M. Hurfin, R. Andonov, L. Mouchard, and F. Guinand, Le projet GénoGRID: une grille expérimentale pour la génomique. *Actes des 3èmes Journées Ouvertes Biologie Informatique Mathématiques,* pp. 27-31, France, 2002.
10. J. Pley, R. Andonov, J.-F. Gibrat, A. Marin, and V. Poirriez Parallélisations d'une méthode de reconnaissance de repliements de protéines (FROST). *Proc. of the 3th Journées Ouvertes de Biologie, Informatique et Mathématiques,* pp. 287-288, 2002.
11. D. Powell, Special Issue on Group Communication. *CACM,* 39(4), 1996.

Parallel Stochastic Search for Protein Secondary Structure Prediction

Victor Robles[1], María S. Pérez[1], Vanessa Herves[1], José M. Peña[1], and Pedro Larrañaga[2]

[1] Department of Computer Architecture and Technology, Technical University of Madrid, Madrid, Spain
[2] Department of Computer Science and Artificial Intelligence, University of the Basque Country, San Sebastián, Spain

Abstract. Prediction of the secondary structure of a protein from its aminoacid sequence remains an important and difficult task. Up to this moment, three generations of Protein Secondary Structure Algorithms have been defined: The first generation is based on statistical information over single aminoacids, the second generation is based on windows of aminoacids –typically 11-21 aminoacids– and the third generation is based on the usage of evolutionary information. In this paper we propose the usage of naïve Bayes and Interval Estimation Naïve Bayes (IENB) –a new semi naïve Bayes approach– as suitable third generation methods for Protein Secondary Structure Prediction (PSSP). One of the main stages of IENB is based on a heuristic optimization, carried out by estimation of distribution algorithms (EDAs). EDAs are non-deterministic, stochastic and heuristic search strategies that belong to the evolutionary computation approaches. These algorithms under complex problems, like Protein Secondary Structure Prediction, require intensive calculation. This paper also introduces a parallel variant of IENB called PIENB (Parallel Interval Estimation Naïve Bayes).

1 Introduction and Related Work

Stochastic search algorithms are founded on the idea of selective and heuristic exploration over the complete space of possible solutions. These algorithms evaluate only a sample of this space and, using some heuristics, select future candidates in terms of their possibilities to improve current solutions. This is a very important issue for the cases in which the evaluation of each candidate is expensive in terms of computation. Although only a (relatively) small set of candidates is evaluated, the number of evaluations for a very complex problem could be very high. There are different efforts to make this kind of techniques to perform faster. The parallel nature of these algorithms sets a clear strategy to deal with this problem.

One of the best known stochastic algorithms are Genetic Algorithms (GAs) [8]. GAs have also been designed as parallel algorithms in three different ways [1,2,15]: (i) as master-slave problem with a single population, the master node computes all the genetic operators and the evaluation of the fitness of the individuals is calculated by slave processors, (ii) multiple-population algorithms, independent problems are executed with its own population, these populations exchange best individual according to some

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 1162–1169, 2004.
© Springer-Verlag Berlin Heidelberg 2004

migration rules (this model has been called *island model* [26,17]) and (iii) fine-grain parallel GAs, consistent in a spatially-structure population with a single individual per node and neighborhood restrictions for genetic crossover.

The most interesting, both in terms of practical application and theoretical contribution, is the island model. The performance gained using this approach comes twofold. First, the global population is split into smaller sub-populations and the offspring of new individuals is also divided by the number of nodes of the computation. Although the computation performance is probably better, as the size of the population decreases the quality of the solution could also be reduced due to the lack of diversity in each of the subpopulations. This is solved by the migration of individuals between populations. Second, there are researchers who claim the possibility to reach superlineal speedups in this kind of algorithms, achieving better result with less number of total individual evaluated. Although there are many controversial discussions [21] some studies about the increment of the selection pressure [2] provide an appropriate answer.

Our contribution deals with the extension of the ideas already developed for parallel GAs towards another stochastic paradigm (EDAs [14]) and apply them to the optimization of the Interval Estimation Naïve Bayes performance. Afterwards IENB will be used to deal with the PSSP problem.

The outline of this paper is as follows. Section 2 is an introduction to the semi naïve Bayes approach IENB. Section 3 describes our parallel version of this approach. Section 4 analyzes naïve Bayes and IENB as suitable methods for PSSP. Section 5 shows the results of the evaluation of these methods in PSSP. Finally, section 6 enumerates the conclusions and outlines further future work.

2 Interval Estimation Naïve Bayes

The naïve Bayes classifier [5,7] is a probabilistic method for classification. It can be used to determine the probability that an example belongs to a class given the values of the predictor variables. The naïve Bayes classifier guarantees optimal induction given a set of explicit assumptions [4]. However, it is known that some of these assumptions are not compliant in many induction scenarios, for instance, the condition of variable independence respecting to the class variable. Improvements of accuracy has been demonstrated by a number of approaches, collectively named semi naïve Bayes classifiers, which try to adjust the naïve Bayes to deal with a-priori unattended assumptions.

Previous semi naïve Bayes classifiers may be divided into three groups, depending on different pre/post-processing issues: (i) to manipulate the variables to be employed prior to application of naïve Bayes induction [11,13,18], (ii) to select subsets of the training examples prior to the application of naïve Bayes classification [10,12] and (iii) to correct the probabilities produced by the standard naïve Bayes [25,6].

In this work, to deal with the problem of Protein Secondary Structure Prediction, we have used a new semi naïve Bayes approach named *Interval Estimation Naïve Bayes (IENB)* [22] that belongs to approaches that correct the probabilities produced by the standard naïve Bayes. In this approach, instead of calculating the point estimation of the conditional probabilities from data, as simple naïve Bayes does, confidence intervals are calculated. After that, by searching for the best combination of values into these intervals, it is aimed to break the assumption of independence among variables the simple naïve Bayes does. This search is carried out by a heuristic search algorithm and is guided by the accuracy of the classifiers.

To deal with the heuristic search EDAs –estimation of distribution algorithms– have been selected. EDAs [14] are non-deterministic, stochastic and heuristic search strategies that belong to the evolutionary computation approaches. In EDAs, a number of solutions or individuals are created every generation, evolving once and again until a satisfactory solution is achieved. In brief, the characteristic that most differentiates EDAs from other evolutionary search strategies, such as GAs, is that the evolution from a generation to the next one is done by estimating the probability distribution of the fittest individuals, and afterwards, by sampling the induced model. This avoids the use of crossing or mutation operators, and, therefore, the number of parameters that EDAs requires is reduced considerably.

While IENB improves naïve Bayes accuracy, its biggest problem is the running time. This problem is worst in the case of the protein dataset due to its size (about 70000 instances). Thus, we have decided the development of a parallel version of this algorithm in order to improve its performance. This parallelization is described in the next section.

3 Parallel IENB

With the aim of increasing the performance and accuracy of IENB, we have developed a parallel version of IENB, named PIENB. This approach is based on the simultaneous execution of the IENB code on different nodes of a cluster, exchanging the best individuals achieved in the nodes each N generations. PIENB uses the island model, described in the first section. The algorithm takes into account the following aspects:

1. Every node generates and improves an independent population, but each N generations, the best M individuals of this population are migrated in a round-robin fashion. The algorithm checks if a concrete individual has been already sent to the target node. Nodes only send individuals that are not included in the destination. This migration implies a faster convergence to the solution, because of the feedback process between the nodes. N and M are configuration parameters, which depends on the population size and the number of nodes. The migrated individuals replace the worst individuals in the destination population.
2. PIENB takes advantage of the higher processor capacity of a cluster of several nodes. Therefore, PIENB may achieve better results in a shorter time. Typically, for a cluster of n nodes, the speedup is near to n.

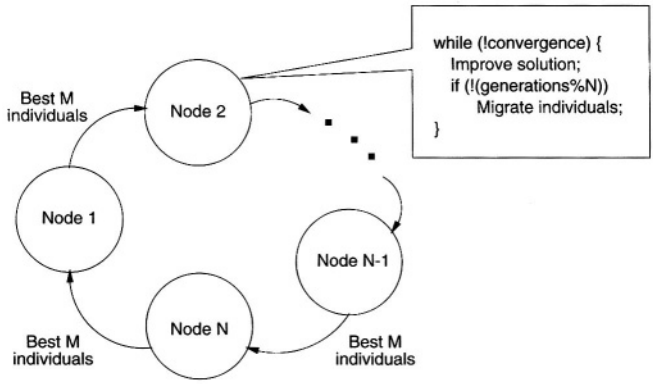


Fig. 1. PIENB flow control

Figure 1 shows the PIENB flow control. The pseudocode implemented in every node is also shown. The arrows represent the messages sent and received by every node, describing the relationship between the nodes. Nevertheless, it is possible to use different network topologies.

When one node has converged, it does not finish, because it has links with other nodes. In this case, this node takes the role of "bridge", receiving and sending messages from and to the corresponding nodes in the topology. Only when all the nodes have converged, the application finishes, belonging the solution to the last node.

To implement PIENB, MPI [16] has been used, mainly because of the following reasons:

1. It is an standard message-passing interface, which allows different processes to communicate among them through the usage of messages.
2. It is widely used in cluster of workstations.
3. It enhances the solution performance, because of its capacity for parallel programming.
4. It provides primitives for changing the network topology.

MPI is used as communication framework in the migration and bridge process.

4 Protein Secondary Structure Prediction with IENB

Prediction of a secondary structure of a protein from its aminoacid sequence remains an important and difficult task. Successful predictions provide a starting point for direct tertiary structure modelling, and also can significantly improve sequence analysis and sequence-structure threading for aiding in structure and function determination [24].

Since early attempts to predict secondary structure, most effort have focused on development of mappings from a local window of residues in the sequence to the structural state of the central residue in the window, and a large number of methods for estimating such mappings have been developed.

Methods predicting protein secondary structure have improved substantially in the 90's through the use of machine learning methods and evolutionary information [23]. At the alignment level, the increasingly size of databases and the ability to produce profiles that include remote homologs using PSI-BLAST have also contributed to performance improvement [9,19,20].

In this section we present a novel approach to protein secondary structure prediction (PSSP) based on the usage of naïve Bayes, IENB and its parallel version (PIENB). Most of the state-of-the-art PSSP methods are based on a three layer fashion: a first layer that maps from sequence to structure, a second layer from structure to structure and a third layer that corrects the obtained structure [23,9]. In this case, we have developed only the first layer with really promising results (see next section).

In order to make the predictions, we have used a window of 13 aminoacids. To be able to use the evolutionary information (profiles) in naïve Bayes, IENB and PIENB we have adjusted the naïve Bayes formula:
Example of protein: A,R,N,S,T,V, ...
Example of protein profile: A80 S20, R50 S45 T5, N75 D5 C5 Q10, ...

Naïve Bayes classification formula (window of n aminoacids):

$$P(C = c|X_1 = x_1, \ldots, X_n = x_n) \propto P(C = c) \prod_{k=1}^{n} P(X_k = x_k|C = c) \quad (1)$$

Naïve Bayes classification formula for proteins profiles (window of n aminoacids):

$$P(C = c|X_1 = x_1, \ldots, X_n = x_n) \propto P(C = c) \prod_{k=1}^{n} \left(\sum_{j=1}^{20} pr_j P(X_k = x_j|C = c) \right) \quad (2)$$

where pr_j is the probability that the aminoacid in position k would be mutated into value x_j.

5 Experimental Results

For the experimentation with PSSP the datasets CB513 [3] has been used. For all the proteins in the dataset the evolutionary information has been included using the program PSI-BLAST from the database PIR-NREF. This database has been filtered to take out low complexity, coiled-coil and transmembrane regions. To generate the learning cases

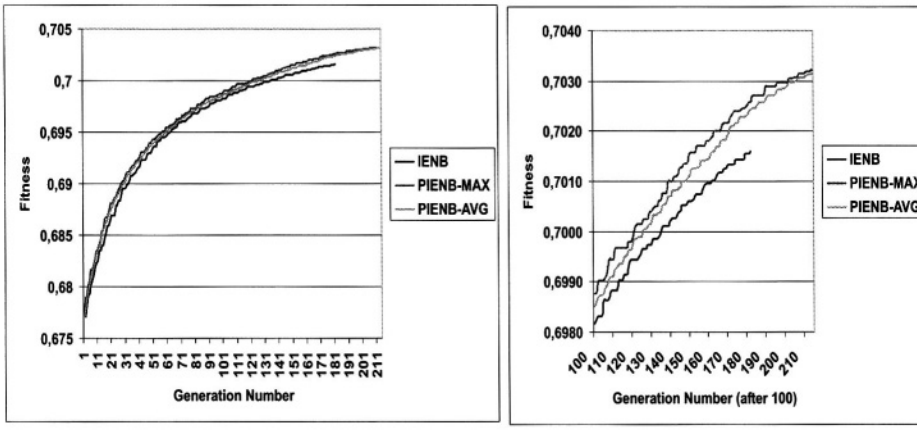


Fig. 2. Fitness value for IENB and PIENB depending on the number of generations

we used a window of 13 aminoacids, obtaining a total of approximately 70000 instances. For obtaining the accuracy prediction a *leave-one-out* validation is performed.

The experimentation has been done with a 8 nodes cluster with Intel Xeon 2MHz, 1GB of RAM and connected by a Gygaethernet.

Several classification mechanism have been performed with this dataset. Table 1 shows the results of all of these executions. First, Naïve Bayes algorithm with no evolutionary information and, second, using this information. An important improvement is achieved as well as an increment in the execution time. This increment is due to (i) the larger number of attributes the algorithm has to estimate, (ii) the more expensive training and evaluation calculation and (iii) the bigger size of the input data (with vs. without profile information).

Table 1. Experimental Results for Protein Secondary Structure Prediction

Algorithm	Accuracy	Time
Naïve Bayes without evolutionary information	61.22	3 seconds
Naïve Bayes	67.58	80 seconds
IENB	70.16	40 days
PIENB	70.33	5 days

The last two rows of the table retrieve the results for both the sequential and parallel versions of the Interval Estimation Naïve Bayes (this last execution has been done 5 times, the showed value is the average). As it is shown a better classification accuracy is achieved but with a difference in execution time of several orders of magnitude. A further analysis of these two cases follows.

The parameters used to performs these experiments have been:

1. **IENB**:
   - Population size: 1000 individuals per generation
   - Offspring: 2000
   - Other options: elitism
2. **PIENB**:
   - Population size: 1000 individuals per generation (125 for each of the subpopulations)
   - Offspring: 2000
   - Migration rate: 10 individuals every 5 generations
   - Migration topology: Unidirectional ring (round-robin)
   - Other options: elitism
   - Migration replacement: Best migrated individuals replace worst

The better performance reached by the parallel version can be possible because of two reasons, first the speedup factor is close to 8 because of the ratio between communication and processing is very low. Second, the exploration of solutions using quasi-independent populations provided by the island model improves the quality of the solution and skips sub-optimal maximums. In order to analyze this bias a representation of the best fitness (in the case of the sequential version) and the best and averaged fitness of each of the subpopulations (for the parallel one) is pictured in graph 2.

6 Conclusions and Further Work

On this contribution a new parallel semi-Naïve Bayes classifier has been presented. This new algorithm is based on stochastic search of the best combination of conditional probabilities. This approach has been designed as a very complex optimization problem, thus a parallel version of the algorithm has been implemented. This parallel version both reduces the execution time and improves the overall fitness of the algorithm. Our method is a single-layer classification approach that is very competitive with state-of-the-art classifiers [9]. And our future interests are addressed to design a second/third layer to perform structure-structure prediction.

The parallel algorithm presented here is a first experiment in the application of multi-population schemas for EDAs algorithms, different topologies [2], different policies and a combination of migration parameters are open to continue researching here.

References

1. T.C. Belding. The distributed genetic algorithm revisited. In *Proceedings of the Sixth International Conference on Genetic Algorithms,* pages 114–121, 1995.
2. E. Cantú-Paz. *Efficient and accurate parallel genetic algorithms.* Kluwer Academic Publishers, 2001.
3. J. A. Cuff and G. J. Barton. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins: Struct. Funct. Genet.,* pages 508–519, 1999.

4. P. Domingos and M. Pazzani. Beyond independence: conditions for the optimality of the simple Bayesian classifier. In *Proceedings ofthe 13th International Conference on Machine Learning,* pages 105–112, 1996.
5. R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.
6. J.T.A.S. Ferreira, D.G.T. Denison, and D.J. Hand. Weighted naive Bayes modelling for data mining. Technical report, Deparment of mathematics, Imperial College, May 2001.
7. D.J. Hand and K. Yu. Idiot's Bayes – not so stupid after all? *International Statistical Review,* 69(3):385–398, 2001.
8. J.H. Holland. Genetic algorithms and the optimal allocation of trials. *Journal on Computing,* 2(2):88–105, 1973.
9. D.T. Jones. Protein secondary structure prediction based on decision-specific scoring matrices. *Journal of Molecular Biology,* 292:195–202, 1999.
10. R. Kohavi. Scaling up the accuracy of naïve-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining,* pages 202–207, 1996.
11 I. Kononenko. Semi-naive Bayesian classifier. In *Sixth European Working Session on Learning,* pages 206–219, 1991.
12. P. Langley. Induction of recursive Bayesian classifiers. In *European Conference on Machine Learning. Berlin: Springer-Verlag,* pages 153–164, 1993.
13. P. Langley and S. Sage. Induction of selective Bayesian classifiers, pages 399–406, 1994.
14. P. Larrañaga and J.A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publisher, 2001.
15. David Levine. *A Parallel Genetic Algorithm for the Set Partitioning Problem.* PhD thesis, Illinois Institute of Technology, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.
16. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard,* May 1994.
17. G. Michaelson and N. Scaife. Parallel functional island model genetic algorithms through nested skeletons. In *Proceedings of 12th International Workshop on the Implementation of Functional Languages,* pages 307–313, September 2000.
18. M. Pazzani. Searching for dependencies in Bayesian classifiers. In *Proceedings ofthe Fifth International Workshop on Artificial Intelligence and Statistics,* pages 239–248, 1996.
19. G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary strucure in three and eight classes using recurrent neural networks and profiles. *Proteins,* 47:228–235, 2002.
20. D. Przybylski and B. Rost. Alignments grow, secondary structure prediction improves. *Proteins,* Submitted, 2001.
21. W.F. Punch. How effective are multiple populations in genetic programming. In *Genetic Programming, Proceedings of the Third Annual Conference,* 1998.
22. V. Robles, P. Larañaga, J.M. Peña, O. Marbán, J. Crespo, and M.S. Pérez. Collaborative filtering using interval estimation naïve bayes. *Lecture Notes in Artificial Intelligence (Advances in Web Intelligence),* (2663):46–53, May 2003.
23. B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology,* 232:584–599, 1993.
24. S.C. Schmidler, J.S. Liu, and D.L. Brutlag. Bayesian segmentation of protein secondary structure. *Journal of Computational Biology,* 7(l/2):233–248, 2000.
25. G.I. Webb and M.J. Pazzani. Adjusted probability naïve Bayesian induction. In *Australian Joint Conference on Artificial Intelligence,* pages 285–295, 1998.
26. Darrell Whitley, Soraya B. Rana, and Robert B. Heckendorn. Island model genetic algorithms and linearly separable problems. In *volutionary Computing, AISB Workshop,* pages 109–125, 1997.

This page intentionally left blank

Author Index

Abascal, Policarpo 963
Abdalhaq, Baker 1067
Abramson, David 586, 730
Abrich, Daniel 389
Abu Salem, Fatima 217
Adamska, Katarzyna 536
Ahn, Chang Wook 544, 552, 560
Ahn, Hyunju 682
Alcalde, Bernardo Frederes Krämer 292
Alevizos, Panagiotis D. 225
Alonso, Pedro 963, 969
Amamiya, Makoto 383
Aouad, Lamine M. 938
Appel, Ron 1146
Astaloš, Ján 831
Atłasiewicz, Krzysztof 746

Badía, José M. 969
Bae, Hae-Young 688
Bahi, Jacques 77
Bała, Piotr 481
Baliś, Bartosz 331,779
Balogh, Zoltan 694, 700
Banaś, Krzysztof 431
Baraglia, Ranieri 1075
Beaumont, Olivier 880
Belevantsev, Andrey 278
Beltrán, Marta 145, 888
Bielecki, Andrzej 439
Bilski, Jarosław 601
Blaheta, Radim 153
Błażewicz, Jacek 1138
Blythe, James 821
Borkowski, Janusz 269, 338
Bosque, Jose L. 145, 888
Bouvry, Pascal 447
Bożejko, Wojciech 566
Brezany, Peter 788, 796
Brzezinski, Jerzy 1,9
Brzezniak, Maciej 161
Bubak, Marian 331, 344, 352, 779, 804, 812, 872
Bujanda, Blanca 1034
Burczynski, Tadeusz 572

Caramico D'Auria, Alvaro 475
Carpentieri, Bruno 977
Castaings, William 521
Chernov, Alexander 278
Choi, Eunmi 83
Chung, Sang-Hwa 682
Cierniak, Robert 609
Cohen, Johanne 896
Cortés, Ana 1067
Cortina, Raquel 963
Couturier, Raphaël 77
Cpałka, Krzysztof 618
Cuenca, Javier 169
Cummings, John 419
Czarnas, Piotr 233
Czarnul, Pawel 360
Czech, Zbigniew J. 233

Dębski, Lech 455
Deelman, Ewa 821
Denneulin, Yves 405,1099
Dereniowski, Dariusz 985
Díaz, Irene 963
Digas, Boris 497
Diverio, Tiarajú Asmuz 292
Dobrucký, Miroslav 831
Dorta, Isabel 286
Drozdowski, Maciej 175
Duff, Iain S. 977
Dziewierz, Mariusz 694, 700
Dzwinel, Witold 513

Esposito, Filippo 475

Fernández, Daniel 505
Flaherty, Joseph E. 1083
Flasiński, Mariusz 746
Forenc, Jaroslaw 528
Formanowicz, Piotr 1138
Frey, Julien 1146
Froehlich, Dave 521
Funika, Włodzimierz 331, 344, 352

Gabryel, Marcin 626
García, Luis-Pedro 169
Gawęda, Adam 626,659

Gawiejnowicz, Stanisław 89
Gepner, Pawel 17
Gil, Yolanda 821
Giménez, Domingo 169
Giraud, Luc 977
Glendinning, Ian 461
Gocyła, Przemyslaw 233
Goldberg, David E. 544
González, José 169
Gorawski, Marcin 676
Górka, Kamil 804
Goscinski, Andrzej 27
Gras, Robin 1146
Grochowski, Marek 97
Gubała, Tomasz 804
Guinet, Christian 1099
Gursoy, Attila 241
Guzmán, Antonio 888

Habala, Ondrej 831
Haglauer, Monika 475
Hajto, Paweł 119
Hayashi, Yoichi 641
Hernandez, Patricia 1146
Herves, Vanessa 1162
Hluchý, Ladislav 521, 694, 700, 831
Hobbs, Michael 27
Hölbig, Carlos Amaral 292
Huedo, Eduardo 840
Hurfin, Michel 1154
Hwang, Sung-Ho 1028

Iio, Kentaro 383
Inutin, Sergey A. 993

Jakl, Ondřej 153
Janciak, Ivan 796
Jansen, Klaus 105
Jeannot, Emmanuel 896
Jeon, Il-Soo 998
Jones, Rhys 730
Jordan, Andrzej 528
Jorge, Juan Carlos 1034
Jung, Sung-Won 682
Jurek, Janusz 746

Kaczmarek, Paweł L. 306
Kalaev, Vladimir 469
Kalinov, Alexey 907
Kamieniarz, Grzegorz 475, 1091
Kang, Jaeho 682
Kaplita, Grzegorz 331
Karczewski, Konrad 189
Kechadi, Tahar 954
Kędziora, Paweł 1138
Kesselman, Carl 821
Kiliański, Zbigniew 344,352
Kim, Hyun-Sung 998, 1028
Kim, Jae-Hong 688
Kim, Ki Pyo 552
Kim, Sung-Hee 688
Kitowski, Jacek 181, 700
Kiyamura, Susumu 298
Klimov, Sergey 907
Kobler, Rene 368
Kokosiński, Zbigniew 249,255
Kopanski, Damian 338
Kopsky, John 497
Koranda, Scott 821
Korošec, Peter 113
Korytkowski, Marcin 626
Kranzlmüller, Dieter 368
Krawczyk, Henryk 306
Krawczyk, Krzysztof 694, 700
Kryazhimskiy, Arkady 754
Krzyżanowski, Piotr 1006
Kubale, Marek 985
Kuczewski, Bartosz 762
Kuczynski, Tomasz 375
Kunszt, Peter 848
Kurc, Wiesław 89
Kurniawan, Donny 730
Kus, Waclaw 572
Kusakabe, Shigeru 383
Kwedlo, Wojciech 580
Kwiatkowski, Jan 389
Kwon, Hyuk-Chul 682

Laclavik, Michal 694, 700
Laforenza, Domenico 1075
Lambert, Simon 700
Larrañaga, Pedro 1162
Laskowski, Eryk 314
Lastovetsky, Alexey 921
Laure, Erwin 848
Lazzarini, Albert 821
Lebre, Adrien 405, 1099
Lee, Inseon 397
Lee, Jae-Dong 688
Lee, Jin-Ho 998

Lee, Sung-Woon 1014
Legrand, Arnaud 930
Leon, Coromoto 286
Leszczynski, Jacek S. 738, 706
Lewis, Andrew 586, 730
Llorente, Ignacio M. 840
Lombard, Pierre 405, 1099
Lukac, Rastislav 595, 714
Lukanin, Denis 469
Luque, Emilio 505, 1067

Macías, Elsa M. 946
Maghraoui, Kaoutar El 1083
Maheshwari, Piyush 915
Majewska, Marta 700
Maksimov, Vyacheslav 754
Malawski, Maciej 804, 812, 872
Malczok, Rafal 676
Malikov, Oleg 278
Margalef, Tomàs 1067
Martín, María J. 323
Masko, Lukasz 59
Maslennikow, Oleg 36
Matysiak, Ryszard 475,1091
Mehta, Gaurang 821
Metkowski, Rafał 481
Meyer, Norbert 161
Min, Dugki 83
Moe, Randi 1020
Montero, Rubén S. 840
Morandi Júnior, Paulo Sérgio 292
Moriya, Kentaro 1105
Musiał, Grzegorz 455

Nagórka, Arkadiusz 722
Napolitano, Jim 419
Narzul, Jean-Pierre Le 1154
Nikolow, Darin 181
Nodera, Takashi 1105
Nowak, Dominika 439
Nowicki, Robert 632, 641

Ömer, Bernhard 461
Olas, Tomasz 189
Oliveira, Arlindo L. 864
Orłowski, Tomasz 489
Osborne, James 856

Padoy, Nicolas 896
Pankowska, Lidia 89
Papa, Maria Alessandra 821
Paprzycki, Marcin 497
Park, Hee-Joo 1028
Park, Hyoungwoo 397
Park, Taesoon 397
Patan, Maciej 762, 770
Peachey, Tom 586, 730
Peña, José M. 1162
Pérez, María S. 1162
Petcu, Dana 261
Petiton, Serge G. 938
Plataniotis, Konstantinos N. 595, 714
Pley, Julien 1154
Pluta, Sebastian 738
Pokropińska, Agata 641
Portero, Laura 1034
Purcz, Pavol 201

Raïpin Parvédy, Philippe 1154
Ramakrishna, R.S. 544, 552, 560
Ramet, Pierre 880
Ranilla, José 963
Reddy, Ravi 921
Renard, Hélène 930
Reroń, Elżbieta 746
Rivera, Francisco F. 323
Robert, Yves 930
Robič, Borut 113
Robles, Victor 1162
Rodriguez, Casiano 286
Rogiers, Jos 455
Roman, Jean 880
Rutkowska, Danuta 650
Ryu, Kwang Ryel 682
Rząsa, Wojciech 779

Sampels, Michael 42
Sánchez, David 946
Saukh, Sergey 1042
Savvas, Ilias 954
Schaefer, Robert 97
Scherer, Rafał 659
Sczygiol, Norbert 722
Seki, Hiroyuki 298
Seredyński, Franciszek 447
Sergyienko, Anatoli 36
Sevastjanow, Pawel 668
Shevtshenko, Juri 36
Šilc, Jurij 113
Silcock, Jackie 27

Šimeček, Ivan 207
Šimo, Branislav 831
Singh, David E. 323
Skrzypek, Marcin 119
Sławińska, Magdalena 413
Sloot, Peter 872
Słota, Renata 181,694,700
Smętek, Marcin 344,352
Smołąg, Jacek 601
Smolka, Bogdan 595, 714
Smyk, Adam 50
Sobaniec, Cezary 1
Solis-Oba, Roberto 105
Starczewski, Janusz 659
Starý, Jiří 153
Šterk, Marjan 1052
Stockinger, Heinz 848
Stockinger, Kurt 848
Stpiczyński, Przemysław 1059
Studzinski, Grzegorz 375
Suárez, Álvaro 946
Suppi, Rerno 505
Swierniak, Andrzej 595, 714
Sylvand, Guillaume 977
Szepieniec, Tomasz 779
Szychowiak, Michal 9
Szymanski, Boleslaw K. 419,1083

Takata, Yoshiaki 298
Taniguchi, Hideo 383
Tasoulis, Dimitris K. 225
Tchernykh, Andrei 131
Teresco, James D. 1083
Thai, Nguyen Due 125
Tjoa, A. Min 788, 796
Tomas, Adam 189
Tonellotto, Nicola 1075
Topa, Pawel 513
Tran, Viet D. 521, 831
Trezentos, Paulo 864
Trobec, Roman 1052
Trystram, Denis 131
Tsigas, Philippas 1114
Tudruj, Marek 50, 59, 338
Tvrdík, Pavel 207

Uciński, Dariusz 762,770
Uhruski, Piotr 97

Vahi, Karan 821
Valentin, Olivier 405, 1099
Varela, Carlos 419, 1083
Varma, Pradeep 69
Venetsanopoulos, Anastasios N. 595, 714
Vernier, Flavien 77
Vidal, Antonio M. 169, 969
Vivien, Frederic 930
Volkert, Jens 368
Vrahatis, Michael N. 225

Wackowski, Kazimierz 17
Walendziuk, Wojciech 528
Wanek, Helmut 788
Wawrzyniak, Dariusz 1
Wielebski, Łukasz 175
Wismüller, Roland 331, 344, 352, 779
Wiszniewski, Bogdan 489
Wodecki, Mieczysław 566
Wöhrer, Alexander 788
Wojciechowski, Paweł 1138
Wójtowicz, Piotr 746
Wright, Helen 856
Wyrzykowski, Roman 189, 375, 738

Yeom, Heon Y. 397
Yoo, Kee-Young 1014

Zaharie, Daniela 261
Zając, Katarzyna 804, 812, 872
Zhang, Yi 1114
Zhmakin, Alexander 469
Zlatev, Zahari 1125
Zomaya, Albert Y. 447
Żurada, Jacek 601

This page intentionally left blank

This page intentionally left blank

Lecture Notes in Computer Science

For information about Vols. 1–2901

please contact your bookseller or Springer-Verlag

Vol. 3027: C. Cachin, J. Camenisch (Eds.), Advances in Cryptology - EUROCRYPT 2004. XI, 628 pages. 2004.

Vol. 3025: G.A. Vouros, T. Panayiotopoulos (Eds.), Methods and Applications of Artificial Intelligence. XV, 546 pages. 2004. (Subseries LNAI).

Vol. 3019: R. Wyrzykowski, J. Dongarra, M. Paprzycki, J. Waśniewski (Eds.), Parallel Processing and Applied Mathematics. XIX, 1174 pages. 2004.

Vol. 3015: C. Barakat, I. Pratt (Eds.), Passive and Active Network Measurement. XI, 300 pages. 2004.

Vol. 3012: K. Kurumatani, S.-H. Chen, A. Ohuchi (Eds.), Multi-Agnets for Mass User Support. X, 217 pages. 2004. (Subseries LNAI).

Vol. 3011: J.-C. Régin, M. Rueher (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. XI, 415 pages. 2004.

Vol. 3010: K.R. Apt, F. Fages, F. Rossi, P. Szeredi, J. Váncza (Eds.), Recent Advances in Constraints. VIII, 285 pages. 2004. (Subseries LNAI).

Vol. 3009: F. Bomarius, H. Iida (Eds.), Product Focused Software Process Improvement. XIV, 584 pages. 2004.

Vol. 3007: J.X. Yu, X. Lin, H. Lu, Y. Zhang (Eds.), Advanced Web Technologies and Applications. XXII, 936 pages. 2004.

Vol. 3006: M. Matsui, R. Zuccherato (Eds.), Selected Areas in Cryptography. XI, 361 pages. 2004.

Vol. 3005: G.R. Raidl, S. Cagnoni, J. Branke, D.W. Corne, R. Drechsler, Y. Jin, C.G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G.D. Smith, G. Squillero (Eds.), Applications of Evolutionary Computing. XVII, 562 pages. 2004.

Vol. 3004: J. Gottlieb, G.R. Raidl (Eds.), Evolutionary Computation in Combinatorial Optimization. X, 241 pages. 2004.

Vol. 3003: M. Keijzer, U.-M. O'Reilly, S.M. Lucas, E. Costa, T. Soule (Eds.), Genetic Programming. XI, 410 pages. 2004.

Vol. 3001: A. Ferscha, F. Mattern (Eds.), Pervasive Computing. XVII, 358 pages. 2004.

Vol. 2999: E.A. Boiten, J. Derrick, G. Smith (Eds.), Integrated Formal Methods. XI, 541 pages. 2004.

Vol. 2998: Y. Kameyama, P.J. Stuckey (Eds.), Functional and Logic Programming. X, 307 pages. 2004.

Vol. 2997: S. McDonald, J. Tait (Eds.), Advances in Information Retrieval. XIII, 427 pages. 2004.

Vol. 2996: V. Diekert, M. Habib (Eds.), STACS 2004. XVI, 658 pages. 2004.

Vol. 2995: C. Jensen, S. Poslad, T. Dimitrakos (Eds.), Trust Management. XIII, 377 pages. 2004.

Vol. 2994: E. Rahm (Ed.), Data Integration in the Life Sciences. X, 221 pages. 2004. (Subseries LNBI).

Vol. 2993: R. Alur, G.J. Pappas (Eds.), Hybrid Systems: Computation and Control. XII, 674 pages. 2004.

Vol. 2992: E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, E. Ferrari (Eds.), Advances in Database Technology - EDBT 2004. XVIII, 877 pages. 2004.

Vol. 2991: R. Alt, A. Frommer, R.B. Kearfott, W. Luther (Eds.), Numerical Software with Result Verification. X, 315 pages. 2004.

Vol. 2989: S. Graf, L. Mounier (Eds.), Model Checking Software. X, 309 pages. 2004.

Vol. 2988: K. Jensen, A. Podelski (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XIV, 608 pages. 2004.

Vol. 2987: I. Walukiewicz (Ed.), Foundations of Software Science and Computation Structures. XIII, 529 pages. 2004.

Vol. 2986: D. Schmidt (Ed.), Programming Languages and Systems. XII, 417 pages. 2004.

Vol. 2985: E. Duesterwald (Ed.), Compiler Construction. X, 313 pages. 2004.

Vol. 2984: M. Wermelinger, T. Margaria-Steffen (Eds.), Fundamental Approaches to Software Engineering. XII, 389 pages. 2004.

Vol. 2983: S. Istrail, M.S. Waterman, A. Clark (Eds.), Computational Methods for SNPs and Haplotype Inference. IX, 153 pages. 2004. (Subseries LNBI).

Vol. 2982: N. Wakamiya, M. Solarski, J. Sterbenz (Eds.), Active Networks. XI, 308 pages. 2004.

Vol. 2981: C. Müller-Schloer, T. Ungerer, B. Bauer (Eds.), Organic and Pervasive Computing –ARCS 2004. XI, 339 pages. 2004.

Vol. 2980: A. Blackwell, K. Marriott, A. Shimojima(Eds.), Diagrammatic Representation and Inference. XV, 448 pages. 2004. (Subseries LNAI).

Vol. 2978: R. Groz, R.M. Hierons (Eds.), Testing of Communicating Systems. XII, 225 pages. 2004.

Vol. 2977: G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, F. Zambonelli (Eds.), Engineering Self-Organising Systems. X, 299 pages. 2004. (Subseries LNAI).

Vol. 2976: M. Farach-Colton (Ed.), LATIN 2004: Theoretical Informatics. XV, 626 pages. 2004.

Vol. 2973: Y. Lee, J. Li, K.-Y. Whang, D. Lee (Eds.), Database Systems for Advanced Applications. XXIV, 925 pages. 2004.

Vol. 2972: R. Monroy, G. Arroyo-Figueroa, L.E. Sucar, H. Sossa (Eds.), MICAI2004: Advances in Artificial Intelligence. XVII, 923 pages. 2004. (Subseries LNAI).

Vol. 2971: J.I. Lira, D.H. Lee (Eds.), Information Security and Cryptology -ICISC 2003. XI, 458 pages. 2004.

Vol. 2970: F. Fernández Rivera, M. Bubak, A. Gómez Tato, R. Doallo (Eds.), Grid Computing. XI, 328 pages. 2004.

Vol. 2966: F.B. Sachse, Computational Cardiology. XVIII, 322 pages. 2004.

Vol. 2965: M.C. Calzarossa, E. Gelenbe, Performance Tools and Applications to Networked Systems. VIII, 385 pages. 2004.

Vol. 2964: T. Okamoto (Ed.), Topics in Cryptology - CT-RSA 2004. XI, 387 pages. 2004.

Vol. 2963: R. Sharp, Higher Level Hardware Synthesis. XVI, 195 pages. 2004.

Vol. 2962: S. Bistarelli, Semirings for Soft Constraint Solving and Programming. XII, 279 pages. 2004.

Vol. 2961: P. Eklund (Ed.), Concept Lattices. IX, 411 pages. 2004. (Subseries LNAI).

Vol. 2960: P.D. Mosses (Ed.), CASL Reference Manual. XVII, 528 pages. 2004.

Vol. 2958: L. Rauchwerger (Ed.), Languages and Compilers for Parallel Computing. XI, 556 pages. 2004.

Vol. 2957: P. Langendoerfer, M. Liu, I. Malta, V. Tsaoussidis (Eds.), Wired/Wireless Internet Communications. XI, 307 pages. 2004.

Vol. 2956: A. Dengel, M. Junker, A. Weisbecker (Eds.), Reading and Learning. XII, 355 pages. 2004.

Vol. 2954: F. Crestani, M. Dunlop, S. Mizzaro (Eds.), Mobile and Ubiquitous Information Access. X, 299 pages. 2004.

Vol. 2953: K. Konrad, Model Generation for Natural Language Interpretation and Analysis. XIII, 166 pages. 2004. (Subseries LNAI).

Vol. 2952: N. Guelfi, E. Astesiano, G. Reggio (Eds.), Scientific Engineering of Distributed Java Applications. X, 157 pages. 2004.

Vol. 2951: M. Naor (Ed.), Theory of Cryptography. XI, 523 pages. 2004.

Vol. 2949: R. De Nicola, G. Ferrari, G. Meredith (Eds.), Coordination Models and Languages. X, 323 pages. 2004.

Vol. 2948: G.L. Mullen, A. Poli, H. Stichtenoth (Eds.), Finite Fields and Applications. VIII, 263 pages. 2004.

Vol. 2947: F. Bao, R. Deng, J. Zhou (Eds.), Public Key Cryptography - PKC 2004. XI, 455 pages. 2004.

Vol. 2946: R. Focardi, R. Gorrieri (Eds.), Foundations of Security Analysis and Design II. VII, 267 pages. 2004.

Vol. 2943: J. Chen, J. Reif (Eds.), DNA Computing. X, 225 pages. 2004.

Vol. 2941: M. Wirsing, A. Knapp, S. Balsamo (Eds.), Radical Innovations of Software and Systems Engineering in the Future. X, 359 pages. 2004.

Vol. 2940: C. Lucena, A. Garcia, A. Romanovsky, J. Castro, P.S. Alencar (Eds.), Software Engineering for Multi-Agent Systems II. XII, 279 pages. 2004.

Vol. 2939: T. Kalker, I.J. Cox, Y.M. Ro (Eds.), Digital Watermarking. XII, 602 pages. 2004.

Vol. 2937: B. Steffen, G. Levi (Eds.), Verification, Model Checking, and Abstract Interpretation. XI, 325 pages. 2004.

Vol. 2936: P. Liardet, P. Collet, C. Fonlupt, E. Lutton, M. Schoenauer (Eds.), Artificial Evolution. XIV, 410 pages. 2004.

Vol. 2934: G. Lindemann, D. Moldt, M. Paolucci (Eds.), Regulated Agent-Based Social Systems. X, 301 pages. 2004. (Subseries LNAI).

Vol. 2930: F. Winkler (Ed.), Automated Deduction in Geometry. VII, 231 pages. 2004. (Subseries LNAI).

Vol. 2929: H. de Swart, E. Orlowska, G. Schmidt, M. Roubens (Eds.), Theory and Applications of Relational Structures as Knowledge Instruments. VII, 273 pages. 2003.

Vol. 2926: L. van Elst, V. Dignum, A. Abecker (Eds.), Agent-Mediated Knowledge Management. XI, 428 pages. 2004. (Subseries LNAI).

Vol. 2923: V. Lifschitz, I. Niemelä (Eds.), Logic Programming and Nonmonotonic Reasoning. IX, 365 pages. 2004. (Subseries LNAI).

Vol. 2919: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing. XI, 530 pages. 2004.

Vol. 2917: E. Quintarelli, Model-Checking Based Data Retrieval. XVI, 134 pages. 2004.

Vol. 2916: C. Palamidessi (Ed.), Logic Programming, XII, 520 pages. 2003.

Vol. 2915: A. Camurri, G. Volpe (Eds.), Gesture-Based Communication in Human-Computer Interaction. XIII, 558 pages. 2004. (Subseries LNAI).

Vol. 2914: P.K. Pandya, J. Radhakrishnan (Eds.), FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science. XIII, 446 pages. 2003.

Vol. 2913: T.M. Pinkston, V.K. Prasanna (Eds.), High Performance Computing - HiPC 2003. XX, 512 pages. 2003. (Subseries LNAI).

Vol. 2911: T.M.T. Sembok, H.B. Zaman, H. Chen, S.R. Urs, S.H. Myaeng (Eds.), Digital Libraries: Technology and Management of Indigenous Knowledge for Global Access. XX, 703 pages. 2003.

Vol. 2910: M.E. Orlowska, S. Weerawarana, M.M.P. Papazoglou, J. Yang (Eds.), Service-Oriented Computing - ICSOC 2003. XIV, 576 pages. 2003.

Vol. 2909: R. Solis-Oba, K. Jansen (Eds.), Approximation and Online Algorithms. VIII, 269 pages. 2004.

Vol. 2908: K. Chae, M. Yung (Eds.), Information Security Applications. XII, 506 pages. 2004.

Vol. 2907: I. Lirkov, S. Margenov, J. Waśniewski, P. Yalamov (Eds.), Large-Scale Scientific Computing. XI, 490 pages. 2004.

Vol. 2906: T. Ibaraki, N. Katoh, H. Ono (Eds.), Algorithms and Computation. XVII, 748 pages. 2003.

Vol. 2905: A. Sanfeliu, J. Ruiz-Shulcloper (Eds.), Progress in Pattern Recognition, Speech and Image Analysis. XVII, 693 pages. 2003.

Vol. 2904: T. Johansson, S. Maitra (Eds.), Progress in Cryptology - INDOCRYPT 2003. XI, 431 pages. 2003.

Vol. 2903: T.D. Gedeon, L.C.C. Fung (Eds.), AI 2003: Advances in Artificial Intelligence. XVI, 1075 pages. 2003. (Subseries LNAI).

Vol. 2902: F.M. Pires, S.P. Abreu (Eds.), Progress in Artificial Intelligence. XV, 504 pages. 2003. (Subseries LNAI).