Parallel Integer Relation Detection: Techniques and Applications

David H. Bailey
David J. Broadhurst

Draft 1999-10-12

Abstract

Let $\{x_1, x_2, \cdots, x_n\}$ be a vector of real numbers. An *integer relation algorithm* is a computational scheme to find the n integers a_k, if they exist, such that $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0$. In the past few years, integer relation algorithms have been utilized to discover new results in mathematics and physics. Existing programs for this purpose require very large amounts of computer time, due in part to the requirement for multiprecision arithmetic, yet are poorly suited for parallel processing.

This paper presents a new integer relation algorithm designed for parallel computer systems, but as a bonus it also gives superior results on single processor systems. Single- and multi-level implementations of this algorithm are described, together with performance results on a parallel computer system. Several applications of these programs are discussed, including some new results in mathematical number theory, quantum field theory and chaos theory.

AMS MSC: Primary 11Y-16; secondary 11-04

D. H. Bailey: Lawrence Berkeley Laboratory, MS 50B-2239, Berkeley, CA 94720, USA, `dhbailey@lbl.gov`. This work was supported by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC03-76SF00098.

D. J. Broadhurst: Open University, Department of Physics, Milton Keynes MK76AA, United Kingdom, `D.Broadhurst@open.ac.uk`.

1. Introduction

Let $x = (x_1, x_2, \cdots, x_n)$ be a vector of real numbers. x is said to possess an integer relation if there exist integers a_i, not all zero, such that $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0$. By an *integer relation algorithm*, we mean a practical computational scheme that can recover (provided the computer implementation has sufficient numeric precision) the vector of integers a_i, if it exists, or can produce bounds within which no integer relation exists.

The problem of finding integer relations among a set of real numbers was first studied by Euclid, who gave an iterative scheme which, when applied to two real numbers, either terminates, yielding an exact relation, or produces an infinite sequence of approximate relations. The generalization of this problem for $n > 2$ was attempted by Euler, Jacobi, Poincaré, Minkowski, Perron, Brun, Bernstein, among others. The first integer relation algorithm with the required properties mentioned above was discovered in 1977 by Ferguson and Forcade [19]. Since then, a number of other integer relation algorithms have been discovered, including the "HJLS" algorithm [21] (which is based on the LLL algorithm), and the "PSLQ" algorithm.

2. The PSLQ Algorithm

The PSLQ integer relation algorithm features excellent numerical stability, and it is effective in recovering a relation when the input is known to only limited precision. It has been generalized to complex and even quaternion number systems. A detailed discussion of the PSLQ algorithm, together with a proof that the algorithm is guaranteed to recover a relation in a polynomially bounded number of iterations, is given in [18]. The name "PSLQ" derives from its usage of a partial sum of squares vector and a LQ (lower-diagonal-orthogonal) matrix factorization.

A simple statement of the PSLQ algorithm, which is entirely equivalent to the original formulation, is as follows: Let x be the n-long input real vector, and let nint denote the nearest integer function (for exact half-integer values, define nint to be the integer with greater absolute value). Select $\gamma > \sqrt{4/3}$. Then perform the following operations:

Initialize:

1. For $j := 1$ to n: for $i := 1$ to n: if $i = j$ then set $A_{ij} := 1$ and $B_{ij} := 1$ else set $A_{ij} := 0$ and $B_{ij} := 0$; endfor; endfor.

2. For $k := 1$ to n: set $s_k := \sqrt{\sum_{j=k}^{n} x_j^2}$; endfor. Set $t = 1/s_1$. For $k := 1$ to n: set $y_k := t x_k$; $s_k := t s_k$; endfor.

3. Initial H: For $j := 1$ to $n-1$: for $i := 1$ to $j-1$: set $H_{ij} := 0$; endfor; set $H_{jj} := s_{j+1}/s_j$; for $i := j+1$ to n: set $H_{ij} := -y_i y_j/(s_j s_{j+1})$; endfor; endfor.

4. Reduce H: For $i := 2$ to n: for $j := i-1$ to 1 step -1: set $t := \mathrm{nint}(H_{ij}/H_{jj})$; and $y_j := y_j + t y_i$; for $k := 1$ to j: set $H_{ik} := H_{ik} - t H_{jk}$; endfor; for $k := 1$ to n: set $A_{ik} := A_{ik} - t A_{jk}$ and $B_{kj} := B_{kj} + t B_{ki}$; endfor; endfor; endfor.

Iteration: Repeat the following steps until precision has been exhausted or a relation has been detected.

1. Select m such that $\gamma^i |H_{ii}|$ is maximal when $i = m$.

2. Exchange the entries of y indexed m and $m+1$, the corresponding rows of A and H, and the corresponding columns of B.

3. Remove corner on H diagonal: If $m \leq n-2$ then set $t_0 := \sqrt{H_{mm}^2 + H_{m,m+1}^2}$, $t_1 := H_{mm}/t_0$ and $t_2 := H_{m,m+1}/t_0$; for $i := m$ to n: set $t_3 := H_{im}$, $t_4 := H_{i,m+1}$, $H_{im} := t_1 t_3 + t_2 t_4$ and $H_{i,m+1} := -t_2 t_3 + t_1 t_4$; endfor; endif.

4. Reduce H: For $i := m+1$ to n: for $j := \min(i-1, m+1)$ to 1 step -1: set $t := \mathrm{nint}(H_{ij}/H_{jj})$ and $y_j := y_j + t y_i$; for $k := 1$ to j: set $H_{ik} := H_{ik} - t H_{jk}$; endfor; for $k := 1$ to n: set $A_{ik} := A_{ik} - t A_{jk}$ and $B_{kj} := B_{kj} + t B_{ki}$; endfor; endfor; endfor.

5. Norm bound: Compute $M := 1/\max_j |H_{jj}|$. Then there can exist no relation vector whose Euclidean norm is less than M.

6. Termination test: If the largest entry of A exceeds the level of numeric precision used, then precision is exhausted. If the smallest entry of the y vector is less than the detection threshold (see below), a relation has been detected and is given in the corresponding column of B.

We will now review a key result regarding the PSLQ algorithm. Let x be the n-long input real vector, and let H_x the original H matrix constructed in the initialization step above. Let $|\cdot|$ denote the Euclidean norm of a vector. Then Theorem 1 of [18] implies that after any number of iterations (presuming no relation has yet been found), any integer relation r of the vector x must satisfy

$$|r| \leq \frac{1}{\max_{1 \leq j \leq n-1} |H_{j,j}|}$$

This result is the basis for step 5 in the algorithm above.

The PSLQ algorithm, in effect, constructs a series of invertible integer matrices whose product is A (B is the inverse of A), and a series of orthogonal real matrices whose product is Q (Q is not explicitly computed above), such that $H = A H_x Q$ is lower trapezoidal. Theorem 1 of [18] actually states more than the bound result above. If A is *any* invertible integer matrix, and Q is the orthogonal matrix such that $A H_x Q$ is lower triangular (such a Q can always be produced by an LQ matrix factorization of $A H_x$) then this result on the norm of r still holds. We will take advantage of this fact in section 6.

It should be emphasized that for almost all applications of an integer relation algorithm such as PSLQ, very high precision arithmetic must be used. Only a very small class of relations can be recovered reliably with the 64-bit IEEE floating-point arithmetic that is available on current computer systems. In general, if one wishes to recover a relation of length n, with coefficients of maximum size d digits, then it follows by an information theory argument that the input vector x must be specified to at least nd digits, and one must employ floating-point arithmetic accurate to at least nd digits. Practical integer relation programs always require greater precision than this bound. In fact, the difference between the level of precision required for a given problem and the information theory bound is a key figure of merit for integer relation algorithms. PSLQ is very efficient in this regard — for most problems, PSLQ programs can reliably recover relations with only a few percent more digits of precision than the information theory bound.

The software products Maple and Mathematica include multiple precision arithmetic facilities. One may also use any of several freeware multiprecision software packages, such as the MPFUN package (Fortran-77 and Fortran-90 versions are available), which was developed by the first author [1, 2], and the C/C++ version of MPFUN, which was recently developed by Sid Chatterjee and Hermann Harjono of the University of North Carolina [16]. The two MPFUN packages permit one to write a program in conventional Fortran-77/90 or C/C++, respectively, identifying some or all of the variables to be multiple precision (integer, real or complex). Then in expressions where these variables appear, the appropriate multiple precision routines are automatically referenced, thus saving considerable programming effort.

In the course of the operation of the PSLQ algorithm on a real computer system, the entries of the y vector gradually decrease in size, with the largest and smallest entries usually differing by no more than two or three orders of magnitude. When a relation is detected by the algorithm, the smallest entry of the y vector abruptly decreases to roughly the multiprecision "epsilon" (i.e. 10^{-p}, where $p > nd$ is the precision level in digits). The detection threshold in the termination test (iteration step 6) above is typically set to be a few orders of magnitude greater than the epsilon value, in order to allow for reliable relation detection in the presence of some numerical round-off error. The ratio between the smallest and the largest y entry when a relation is detected can be taken as a "confidence level" that the relation is a true relation and not an artifact of insufficient numeric precision. Very small ratios at detection, such as 10^{-100}, almost certainly denote a true relation, although of course such results do not constitute a rigorous proof.

As shown in [18], the PSLQ algorithm is guaranteed to find relations in a bounded number of iterations. However, this result is based on the assumption of perfect, infinite-precision arithmetic. In an implementation on a real computer system, one can never rule out hardware, software and programming errors, although the chances of these errors can be minimized by independent computations. Also, PSLQ programs utilize multiprecision software with finite working precision, and they make decisions based on numerical tolerances. Thus it is possible that numerical anomalies can result, although these anomalies can generally be remedied by using higher precision.

3. Some Applications of the PSLQ Algorithm

One application of PSLQ in the field of mathematical number theory is to determine whether or not a given constant α, whose value can be computed to high precision, is algebraic of some degree n or less. This can be done by first computing the vector $x = (1, \alpha, \alpha^2, \cdots, \alpha^n)$ to high precision and then applying an integer relation algorithm. If a relation is found for x, then this relation vector is precisely the set of integer coefficients of a polynomial satisfied by α. If a relation is not found, the maximum bound determined by PSLQ means that α cannot be the root of an polynomial of degree less than or equal to n, with integer coefficients whose size (Euclidean norm) is less than the established bound. For example, it is well known [8] that

$$\begin{aligned}
\zeta(2) &= 3\sum_{k=1}^{\infty}\frac{1}{k^2\binom{2k}{k}} \\
\zeta(3) &= \frac{5}{2}\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k^3\binom{2k}{k}} \\
\zeta(4) &= \frac{36}{17}\sum_{k=1}^{\infty}\frac{1}{k^4\binom{2k}{k}}
\end{aligned}$$

These results have led some to suggest that

$$Z_5 = \zeta(5)/\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k^5\binom{2k}{k}}$$

might also be a simple rational or algebraic number. Computations using the PSLQ algorithm [1] have established that if Z_5 satisfies a polynomial of degree 25 or less, then the Euclidean norm of the coefficients must exceed 2×10^{37}. Results such as this strongly suggest that the constants $\zeta(n)$ for $n > 4$ are not given by simple one-term formulas as above. Indeed, this "negative" result was fruitful in that it led to the discovery of multi-term identities for such sums [11]. An example will be given in section 8.

One of the first "positive" results of this sort was the identification of the constant $B_3 = 3.54409035955\cdots$ [1]. B_3 is the third bifurcation point of the logistic map $x_{k+1} = rx_k(1-x_k)$, which exhibits period doubling shortly before the onset of chaos. To be precise, B_3 is the smallest value of the parameter r such that successive iterates x_k exhibit eight-way periodicity instead of four-way periodicity. Computations using a predecessor algorithm to PSLQ found that B_3 is a root the polynomial $4913+2108t^2-604t^3-977t^4+8t^5+44t^6+392t^7-193t^8-40t^9+48t^{10}-12t^{11}+t^{12}$. A result for B_4 will be given in section 8.

A large number of results were recently found using PSLQ in the course of research on multiple sums, such as those evaluated in Table 1. After computing the numerical values of many of these constants, a PSLQ program was used to determine if a given constant satisfied an identity of a conjectured form. These efforts produced numerous empirical evaluations and suggested general results [3]. Eventually, elegant proofs were found for many of these specific and general results ([6] and [7]). Three examples of identities that are now proven are given in Table 1. In the table, $\zeta(t) = \sum_{j=1}^{\infty} j^{-t}$ is the Riemann zeta function, and $\mathrm{Li}_n(x) = \sum_{j=1}^{\infty} x^j j^{-n}$ denotes the polylogarithm function.

It has been found that there is an intimate connection between such multiple sums and the constants resulting from evaluation of Feynman diagrams in quantum field theory [12, 13]. In particular, the renormalization procedure (which removes infinities from the perturbation expansion) entails *multiple zeta values* defined by [9]

$$\zeta(s_1, s_2, \cdots, s_r) = \sum_{k_1>k_2>\cdots>k_r>0}\frac{1}{k_1^{s_1}\,k_2^{s_2}\cdots k_r^{s_r}}$$

The ζ notation is used in analogy with Riemann's zeta function. The PSLQ algorithm was used to find formulas and identities involving these constants. Again, a fruitful theory emerged, including a large number of both specific and general results [9, 10].

$$
\begin{aligned}
\sum_{k=1}^{\infty}\left(1+\tfrac{1}{2}+\cdots+\tfrac{1}{k}\right)^2 (k+1)^{-4} &= \tfrac{37}{22680}\pi^6-\zeta^2(3)\\
\sum_{k=1}^{\infty}\left(1+\tfrac{1}{2}+\cdots+\tfrac{1}{k}\right)^3 (k+1)^{-6} &= \zeta^3(3)+\tfrac{197}{24}\zeta(9)+\tfrac{1}{2}\pi^2\zeta(7)\\
&\quad -\tfrac{11}{120}\pi^4\zeta(5)-\tfrac{37}{7560}\pi^6\zeta(3)\\
\sum_{k=1}^{\infty}\left(1-\tfrac{1}{2}+\cdots+(-1)^{k+1}\tfrac{1}{k}\right)^2 (k+1)^{-3} &= 4\,\mathrm{Li}_5(\tfrac{1}{2})-\tfrac{1}{30}\ln^5(2)-\tfrac{17}{32}\zeta(5)\\
&\quad -\tfrac{11}{720}\pi^4\ln(2)+\tfrac{7}{4}\zeta(3)\ln^2(2)+\tfrac{1}{18}\pi^2\ln^3(2)-\tfrac{1}{8}\pi^2\zeta(3)
\end{aligned}
$$

Table 1: Specimen evaluations, found with PSLQ and now proven

Some recent quantum field theory results using PSLQ are even more remarkable. For example, it has now been shown [14] that in each of ten cases with unit or zero mass, the finite part the scalar 3-loop tetrahedral vacuum Feynman diagram reduces to 4-letter "words" that represent iterated integrals in an alphabet of 7 "letters" comprising the one-forms $\Omega := dx/x$ and $\omega_k := dx/(\lambda^{-k}-x)$, where $\lambda := (1+\sqrt{-3})/2$ is the primitive sixth root of unity, and k runs from 0 to 5. A 4-letter word is a 4-dimensional iterated integral, such as

$$U \;:=\; \zeta(\Omega^2\omega_3\omega_0) = \int_0^1 \frac{dx_1}{x_1}\int_0^{x_1}\frac{dx_2}{x_2}\int_0^{x_2}\frac{dx_3}{(-1-x_3)}\int_0^{x_3}\frac{dx_4}{(1-x_4)} = \sum_{j>k>0}\frac{(-1)^{j+k}}{j^3k}$$

There are 7^4 four-letter words. Only two of these are primitive terms occurring in the 3-loop Feynman diagrams: U, above, and

$$V \;:=\; \mathrm{Real}[\zeta(\Omega^2\omega_3\omega_1)] = \sum_{j>k>0}\frac{(-1)^j\cos(2\pi k/3)}{j^3k}.$$

The remaining terms in the diagrams reduce to products of constants found in Feynman diagrams with fewer loops. These ten cases as shown in Figure 1. In these diagrams, dots indicate particles with nonzero rest mass. The formulas that have been found for the corresponding constants are given in Table 2. The constant $C = \sum_{k>0}\sin(\pi k/3)/k^2$.

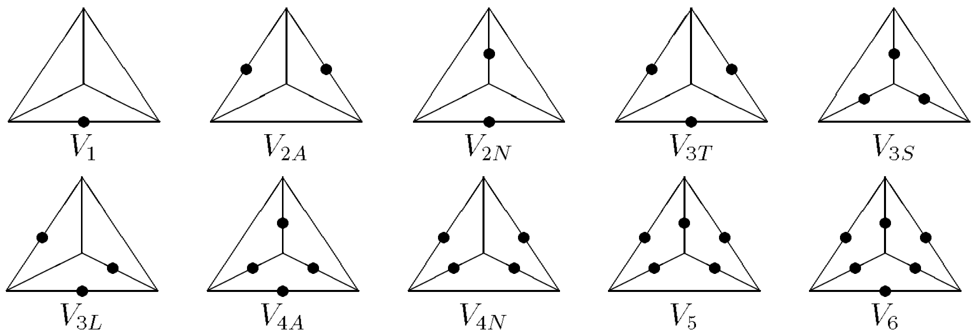

Figure 1: The ten tetrahedral cases

4. A New Formula for Pi

Through the centuries mathematicians have assumed that there is no shortcut to computing just the n-th digit of π. Thus, it came as no small surprise when such an

V_1	$=$	$6\zeta(3)+3\zeta(4)$
V_{2A}	$=$	$6\zeta(3)-5\zeta(4)$
V_{2N}	$=$	$6\zeta(3)-\frac{13}{2}\zeta(4)-8U$
V_{3T}	$=$	$6\zeta(3)-9\zeta(4)$
V_{3S}	$=$	$6\zeta(3)-\frac{11}{2}\zeta(4)-4C^2$
V_{3L}	$=$	$6\zeta(3)-\frac{15}{4}\zeta(4)-6C^2$
V_{4A}	$=$	$6\zeta(3)-\frac{77}{12}\zeta(4)-6C^2$
V_{4N}	$=$	$6\zeta(3)-14\zeta(4)-16U$
V_5	$=$	$6\zeta(3)-\frac{469}{27}\zeta(4)+\frac{8}{3}C^2-16V$
V_6	$=$	$6\zeta(3)-13\zeta(4)-8U-4C^2$

Table 2: Evaluations of the 10 constants corresponding to the 10 cases in Figure 1

algorithm was recently discovered [4]. In particular, this simple scheme allows one to compute the n-th hexadecimal (or binary) digit of π without computing any of the first $n-1$ digits, without using multiple-precision arithmetic software, and at the expense of very little computer memory. The one millionth hex digit of π can be computed in this manner on a current-generation personal computer in only about 60 seconds run time.

This scheme is based on the following new formula, which was discovered using PSLQ:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left[\frac{4}{8k+1}-\frac{2}{8k+4}-\frac{1}{8k+5}-\frac{1}{8k+6}\right]$$

It is likely the first instance in history of a significant new formula for π discovered by computer. Further base-2 results are given in [4, 15]. In [14] base-3 results were obtained, including

$$\begin{aligned}\pi^2 = \frac{2}{27}\sum_{k=0}^{\infty}\frac{1}{729^k}&\left[\frac{243}{(12k+1)^2}-\frac{405}{(12k+2)^2}-\frac{81}{(12k+4)^2}-\frac{27}{(12k+5)^2}\right.\\ &\left.-\frac{72}{(12k+6)^2}-\frac{9}{(12k+7)^2}-\frac{9}{(12k+8)^2}-\frac{5}{(12k+10)^2}+\frac{1}{(12k+11)^2}\right]\end{aligned}$$

5. Multi-Level Implementations of PSLQ

In spite of the relative efficiency of PSLQ compared to the other algorithms in the literature, computer run times of programs that straightforwardly implement the PSLQ algorithm are typically quite long. Even modest-sized problems can require many hours for solution on a current personal computer or workstation. This is mainly due to the cost of using high precision arithmetic software for nearly every operation in the algorithm.

As it turns out, it is possible to perform most, if not all, of the PSLQ iterations using ordinary 64-bit computer arithmetic, with only occasional recourse to multiprecision arithmetic. In this way, run times can be dramatically reduced. Here is a sketch of this scheme, which will be referred to as a "two-level" implementation of the PSLQ algorithm.

In the following, "double precision" means the 64-bit IEEE hardware arithmetic available on most current computer systems, and $\bar{y}$, $\bar{A}$, $\bar{B}$ and $\bar{H}$ denotes double precision counterparts to the arrays y, A, B and H in the PSLQ algorithm.

First, perform the multiprecision initialization steps of PSLQ as given in section 2 above. Then perform a double precision "re-initialization" step: set $\bar{A}$ and $\bar{B}$ to the $n \times n$ identity matrix; set $\bar{y}$ to the best double precision approximation of the current y vector, multiplied by a scale factor so that its largest entry is unity; and set $\bar{H}$ to the best double precision approximation of the current H matrix. For some extremely large problems it may be necessary to scale the $\bar{H}$ matrix to avoid numeric overflow. Then perform an LQ (lower-diagonal-orthogonal) matrix factorization on $\bar{H}$, and replace $\bar{H}$ by the lower diagonal portion of the result (the upper right portion is zeroed). The subroutine DQRDC of the Linpack library [17] may be employed for this factorization, provided both the input and output matrices are transposed.

Next, perform PSLQ iterations using the double precision arrays. In the course of these iterations, the entries of $\bar{A}$ and $\bar{B}$ (which contain integer values, although stored as IEEE double precision data), steadily increase in size. Monitor the entries of these matrices as they are updated, and when any entry reaches a certain threshold (the authors use 10^{13}), or when the smallest $\bar{y}$ entry becomes smaller than a certain threshold (the authors use 10^{-14}), then update the multiprecision arrays by means of matrix multiplication operations, as follows:

$$\begin{aligned} y &:= y \cdot \bar{B} \\ B &:= B \cdot \bar{B} \\ A &:= \bar{A} \cdot A \\ H &:= \bar{A} \cdot H \end{aligned}$$

After these updates are performed, the entries of the A matrix and the y vector are checked, as in the termination test (iteration step 6) of PSLQ, and a norm bound is computed. If neither of the termination conditions holds, then the double precision arrays are re-initialized again as mentioned above, another set of double precision iterations are performed, and the process continues.

This general scheme works well for many problems, but there are several difficulties that must be dealt with in a fully robust implementation. One difficulty is that at some point in the computation (typically at the very beginning), the y vector may have a dynamic range that exceeds the range (11 or 12 orders of magnitude) that can be safely handled using double precision iterations. Another difficulty is that occasionally an entry is produced in the $\bar{A}$ or $\bar{B}$ matrix that exceeds the largest whole number ($2^{53} = 9.007 \cdots \times 10^{15}$) that can be exactly represented as 64-bit IEEE data. A straightforward solution when such a condition occurs is to abandon the current iteration, restore a previous iteration's values of $\bar{y}$, $\bar{A}$, $\bar{B}$ and $\bar{H}$, update the multiprecision arrays as above, perform an LQ matrix factorization on the H matrix, and then perform iterations using full multiprecision arithmetic until these special conditions no longer hold.

A more efficient solution for large problems that require very high precision is to employ "intermediate precision", in other words a fixed level of precision (the authors use 125 digits) that is intermediate between double precision and full multiprecision.

Updating the full multiprecision arrays from the intermediate precision arrays is done with matrix multiplication operations in a manner precisely analogous to that described above. Incorporating intermediate precision in this manner gives rise to what we will refer to as a "three-level" implementation of PSLQ.

One additional improvement that can be made to each of these schemes is to omit multiprecision computation of the A matrix (although the double precision and intermediate precision equivalents of A must be computed). The multiprecision A matrix (which is the inverse of the B matrix) is used in the PSLQ algorithm only to determine when execution must be halted due to the exhaustion of numeric precision. However, exhaustion of numeric precision can alternatively be handled by halting iterations when the smallest y entry is sufficiently close to the multiprecision epsilon level (the authors use a factor of 10^{25}).

These three PSLQ schemes (one-level, two-level and three-level) have been implemented by the first author, using the Fortran-90 MPFUN software [2]. Some performance results are shown in Table 3 for a class of problems. Here r, s define the constant $\alpha = 3^{1/r} - 2^{1/s}$, which is algebraic of degree rs, and $n = rs + 1$. The n-long vector of coefficients of the polynomial satisfied by α can thus be obtained by using a PSLQ program, as explained in section 3. The column headed "Iterations" gives the number of PSLQ iterations required for solution, while "Digits" gives the working precision level used, in decimal digits. "Time" gives CPU time in seconds for runs on a single processor of an SGI Origin-2000 system with 195 MHz R10000 CPUs.

It can be seen from these results that the two-level PSLQ program is up to 65 times faster than the one-level program, yet it finds relations just as well, usually in exactly the same course of iterations as the one-level program. The three-level program is faster than the two-level program for large problems, even though the special conditions mentioned above rarely arise in the particular problems mentioned in the table. The reason for this fortunate circumstance appears to be improved data locality in the three-level scheme, which is advantageous on modern cache-based computer systems. Fully detailed computer programs are available from the authors at the web site
`http://www.nersc.gov/~dhbailey`.

6. The Multi-Pair Algorithm

Even with the substantial accelerations described in the previous section, run times are painfully long for some very large problems of current interest in mathematics and physics. Thus one is led to consider employing highly parallel supercomputers, which have the potential of performance hundreds of times faster than for single-processor scientific workstations and personal computers.

Unfortunately, the standard PSLQ algorithm appears singularly unsuited for modern parallel computer systems, which require high levels of coarse-grained concurrency. The main difficulty is that large integer relation problems may require millions of PSLQ iterations, each of which must be completed before the next begins. Further, within an individual iteration, the key reduction operation (iteration step 4) has a recursion that inhibits any possibility for parallel execution, except at the innermost loop level. These considerations have led some researchers in the field to conclude that there is no hope for significant parallel acceleration of PSLQ-type computations.

r,s	n	Iterations	One-level Digits	One-level Time	Two-level Digits	Two-level Time	Three-level Digits	Three-level Time
5,5	26	5143	180	32.37	190	1.29		
5,6	31	9357	240	105.48	250	3.16		
6,6	37	15217	310	298.85	320	7.19		
6,7	43	25361	420	942.66	420	17.22		
7,7	50	36947	500	2363.71	510	36.29		
7,8	57	60817			680	90.08		
8,8	65	86684			850	195.19	910	233.48
8,9	73	124521			1050	425.67	1120	460.34
9,9	82	174140			1310	934.96	1370	922.90
9,10	91	245443			1620	2032.69	1680	1780.65
10,10	101	342931			2000	4968.64	2060	3366.92

Table 3: Run times for the three PSLQ programs

But it turns out that a variant of the PSLQ algorithm can be formulated that dramatically reduces the number of sequential iterations that must be performed, while at the same time exhibiting moderately high concurrency in the major steps of individual iterations. To that end, consider the following algorithm, which will be referred to as the "multi-pair" variant of PSLQ. Here $\gamma > \sqrt{4/3}$ as before, and $\beta = 0.4$.

Initialize:

1. For $j := 1$ to n: for $i := 1$ to n: if $i = j$ then set $A_{ij} := 1$ and $B_{ij} := 1$ else set $A_{ij} := 0$ and $B_{ij} := 0$; endfor; endfor.

2. For $k := 1$ to n: set $s_k := \sqrt{\sum_{j=k}^{n} x_j^2}$; endfor; set $t = 1/s_1$; for $k := 1$ to n: set $y_k := tx_k$; $s_k := ts_k$; endfor.

3. Initial H: For $j := 1$ to $n-1$: for $i := 1$ to $j-1$: set $H_{ij} := 0$; endfor; set $H_{jj} := s_{j+1}/s_j$; for $i := j+1$ to n: set $H_{ij} := -y_i y_j/(s_j s_{j+1})$; endfor; endfor.

Iteration: Repeat the following steps until precision has been exhausted or a relation has been detected.

1. Sort the entries of the $(n-1)$-long vector $\{\gamma^i |H_{ii}|\}$ in decreasing order, producing the sort indices.

2. Beginning at the sort index m_1 corresponding to the largest $\gamma^i |H_{ii}|$, select pairs of indices $(m_i, m_i + 1)$, where m_i is the sort index. If at any step either m_i or $m_i + 1$ has already been selected, pass to the next index in the list. Continue until either βn pairs have been selected, or the list is exhausted. Let p denote the number of pairs actually selected in this manner.

3. For $i := 1$ to p, exchange the entries of y indexed m_i and $m_i + 1$, and the corresponding rows of A, B and H; endfor.

4. Remove corners on H diagonal: For $i := 1$ to p: if $m_i \le n - 2$ then set $t_0 := \sqrt{H^2_{m_i,m_i} + H^2_{m_i,m_i+1}}$, $t_1 := H_{m_i,m_i}/t_0$ and $t_2 := H_{m_i,m_i+1}/t_0$; for $i := m_i$ to n: set $t_3 := H_{i,m_i}$; $t_4 := H_{i,m_i+1}$; $H_{i,m_i} := t_1 t_3 + t_2 t_4$; and $H_{i,m_i+1} := -t_2 t_3 + t_1 t_4$; endfor; endif; endfor.

5. Reduce H: For $i := 2$ to n: for $j := 1$ to $n - i + 1$: set $l := i + j - 1$; for $k := j + 1$ to $l - 1$: set $H_{lj} := H_{lj} - T_{lk} H_{kj}$; endfor; set $T_{lj} := \mathrm{nint}(H_{lj}/H_{jj})$ and $H_{lj} := H_{lj} - T_{lj} H_{jj}$; endfor; endfor.

6. Update y: For $j := 1$ to $n - 1$: for $i := j + 1$ to n: set $y_j := y_j + T_{ij} y_i$; endfor; endfor.

7. Update A and B: For $k := 1$ to n: for $j := 1$ to $n - 1$: for $i := j + 1$ to n: set $A_{ik} := A_{ik} - T_{ij} A_{jk}$ and $B_{jk} := B_{jk} + T_{ij} B_{ik}$; endfor; endfor; endfor.

8. Norm bound: Compute $M := 1/\max_j |H_{jj}|$. Then there can exist no relation vector whose Euclidean norm is less than M.

9. Termination test: If the largest entry of A exceeds the level of numeric precision used, then precision is exhausted. If the smallest entry of the y vector is less than the detection threshold (see section 2), a relation has been detected and is given in the corresponding row of B.

There are several differences between this algorithm and the standard one-level PSLQ algorithm: (1) there is no reduction step in the initialization; (2) the B matrix is transposed from the standard PSLQ algorithm; (3) up to βn disjoint pairs (not just a single pair) of adjacent indices are selected in each iteration; (4) the H reduction loop proceeds along successive lower diagonals of the H matrix; (5) a T matrix is employed, which contains the t multipliers of the standard PSLQ; and (6) the y, A and B arrays are not updated with H, but in separate loops.

Since the multi-pair algorithm maintains the H matrix in lower triangular form, and the A and B matrices are maintained as invertible integer matrices, one can conclude from Theorem 1 of [18] that the norm bound stated in iteration step 8 above is valid, by an argument similar to that used for the original PSLQ algorithm.

Unfortunately, we cannot offer a proof that the multi-pair algorithm is guaranteed to recover a relation in a bounded number of iterations, as can be done with PSLQ. In fact, it has been found that for certain special problems, the multi-pair algorithm, as stated above, falls into a repeating cycle, with a period of (usually) two iterations. Our implementation deals with this difficulty by comparing the y vector at the end of each iteration with saved copies from eight previous iterations, and if a duplication is found, then only one pair of indices is selected in step 2 of the next iteration (so that the next iteration is equivalent to a standard PSLQ iteration). It should be added, however, that these repeating situations are extremely rare in nontrivial problems. We have not seen any instances of such repeats when $n \ge 20$.

On the positive side, we have found, based on our experience with a wide variety of sample problems, that the norm bound increases much more rapidly than in the standard PSLQ. Indeed, it appears that the selection of up to βn disjoint pairs of indices in step 2 above has the effect of reducing the iteration count by nearly the factor βn. This results in a significant saving in the number of expensive H reduction and array update steps. More importantly, without this dramatic reduction in the sequential iteration count, an efficient parallel implementation would not be possible. Parallel issues will be discussed in greater detail in the next section.

Given that the multi-level implementations of PSLQ are so much faster than the standard one-level PSLQ, one might also wonder whether there exist analogous multi-level implementations of the multi-pair algorithm. Happily, the multi-level scheme sketched in section 5 can be adopted almost without change. One change that is required is that the multiprecision arrays are updated as follows:

$$\begin{aligned} y &:= \bar{B} \cdot y \\ B &:= \bar{B} \cdot B \\ A &:= \bar{A} \cdot A \\ H &:= \bar{A} \cdot H \end{aligned}$$

Note that y and B are updated here in the same manner as the A and H arrays. This change stems from the fact that the B matrix in the multi-pair scheme is transposed from the B matrix in the standard PSLQ algorithm.

The multi-pair algorithm and the multi-level implementations described here were all devised to permit parallel processing. But it turns out that these programs also run faster on a single processor system, compared with the standard PSLQ equivalents. Some one-processor timings are shown in Table 4 for the suite of test problems used in Table 3. Note for example that the one-level multi-pair program is up to twice as fast as the one-level PSLQ program, and the three-level multi-pair program is up to 22% faster than the three-level PSLQ program. Note also that the iteration counts are reduced by a factor of up to 34. Finally, note that the multi-pair schemes require slightly less numeric precision for solution than their PSLQ counterparts. The reason for this unanticipated benefit is not known.

7. Parallel Implementations of the Multi-Pair Algorithm

The key steps of the multi-pair iterations are all suitable for parallel execution. First note that the p row exchanges in iteration step 3, as well as the p corner removal operations in step 4, can be performed concurrently, since the p pairs of indices $(m_i, m_i + 1)$ are all disjoint. Secondly, the reorganized H matrix reduction step (step 5), which is equivalent to the H matrix reduction scheme in the standard PSLQ, may be performed concurrently at the second loop level, instead of only at the innermost loop level as in standard PSLQ. The update of the A and B arrays (step 7) is even more favorable to parallel processing: this loop may be performed concurrently at the outermost loop level. The change in the B matrix, which is transposed from the standard PSLQ algorithm, is favorable for an implementation on a distributed memory parallel computer.

The two- and three-level multi-pair schemes are also well suited for parallel computation. This is because the dominant cost of these programs is the matrix multiplication

r,s	n	Iterations	One-level Digits	One-level Time	Two-level Digits	Two-level Time	Three-level Digits	Three-level Time
5,5	26	558	180	26.08	180	1.48		
5,6	31	840	230	70.71	240	3.43		
6,6	37	1136	310	189.27	310	7.84		
6,7	43	1625	400	479.07	410	17.22		
7,7	50	2071	500	1130.85	500	35.64		
7,8	57	2410			660	69.39		
8,8	65	3723			800	169.62	880	214.66
8,9	73	4943			1010	358.07	1100	427.29
9,9	82	6169			1260	744.20	1320	804.51
9,10	91	7850			1560	1556.37	1600	1450.29
10,10	101	10017			1890	3283.08	1950	2747.12

Table 4: Run times for the three multi-pair programs

operations involved in the multiprecision array updates, and these matrix multiplications can be performed concurrently at the outermost loop level. The parallel techniques mentioned in the previous paragraph can still be applied to the double precision and intermediate precision iterations. It turns out, though, that the double precision iterations run so rapidly that parallel processing of these iterations is often not worth the overhead. Nonetheless, we have achieved modest acceleration on very large problems by using parallel processing on some steps of double precision iterations. Some parallel performance results will be given in the next section.

8. Large Applications and Parallel Performance

Four recent applications will be described here, each of which involves very large integer relation problems. Thus they are excellent test cases for the new multi-pair programs.

Reduction of Euler sums: In section 3, we mentioned recent research on multiple zeta values, which play a key role in quantum field theory [13]. More generally, one may define *Euler sums* by [9]

$$\zeta\begin{pmatrix} s_1, & s_2 & \cdots & s_r \\ \sigma_1, & \sigma_2 & \cdots & \sigma_r \end{pmatrix} := \sum_{k_1>k_2>\cdots>k_r>0} \frac{\sigma_1^{k_1}}{k_1^{s_1}} \frac{\sigma_2^{k_2}}{k_2^{s_2}} \cdots \frac{\sigma_r^{k_r}}{k_r^{s_r}}$$

where $\sigma_j = \pm 1$ are signs and $s_j > 0$ are integers. When all the signs are positive, one has a multiple zeta value. Constants with alternating signs appear in problems such as computation of the magnetic moment of the electron.

It was conjectured by the second author that the dimension of the space of Euler sums with weight $w := \sum_j s_j$ is the Fibonacci number $F_{w+1} = F_w + F_{w-1}$, with $F_1 = F_2 = 1$. Complete reductions of all Euler sums to a basis of size F_{w+1} have been obtained with PSLQ at weights $w \le 9$. At weights $w = 10$ and $w = 11$ the conjecture has been stringently tested by application of PSLQ in more than 600 cases. At weight $w = 11$ such tests involve solving integer relations of size $n = F_{12} + 1 = 145$. In a typical case, each of the 145 constants was computed to more than 5,000 digit accuracy, and a working precision level of 5,000 digits was employed in the three-level multi-pair program. A relation was detected at iteration 31,784. The minimum and maximum y vector entries at the point of detection were 9.515×10^{-4970} and 4.841×10^{-4615}, respectively. The ratio of these two values (i.e. the "confidence level") is a tiny 1.965×10^{-355}. Moreover, the ratio of the last two recovered integer coefficients is precisely $-11! = -39916800$. These facts argue strongly against the possibility that the recovered relation is a spurious numerical artifact.

Bifurcation to a 16-cycle: A second large application that we shall mention here is the problem of determining the polynomial satisfied by the constant $B_4 = 3.564407266095\cdots$, the fourth bifurcation point of the logistic map $x_{k+1} = rx_k(1 - x_k)$. In section 3 we noted that an 8-cycle begins at $r = B_3$, where B_3 satisfies a polynomial equation of degree 12. At $r = B_4$, this gives way to 16-cycle. It has been recognized that all B_k are algebraic, but nothing has been known about the degrees or the coefficients of the polynomials satisfied by these constants for $k > 3$. Some conjectural reasoning had suggested that B_4 might satisfy a 240-degree polynomial, and some further analysis had suggested that the constant $\alpha = -B_4(B_4 - 2)$ might satisfy a 120-degree polynomial. In order to test this hypothesis, the three-level multi-pair program was applied to the 121-long vector $(1, \alpha, \alpha^2, \cdots, \alpha^{120})$.

In this case the input data was computed to over 10,000 digit accuracy, and a working precision of 9,500 digits was employed in the three-level multi-pair program. A relation was detected at iteration 56,666. The minimum and maximum y vector entries at the point of detection were 1.086×10^{-9428} and 3.931×10^{-8889}, which form the ratio 2.763×10^{-540}. Further, the recovered integer coefficients descend monotonically from $257^{30} \approx 1.986 \times 10^{72}$ to one. Again, these facts argue very strongly against the solution being a spurious numerical artifact.

Reductions to Multiple Clausen Values: As a third application, consider sums of the form

$$S(k) \;:=\; \sum_{n>0} \frac{1}{n^k \binom{2n}{n}}$$

with, for example, $S(4) = 17\pi^4/3240$. Researchers have sought analytic evaluations of these constants for $k > 4$. As a result of PSLQ computations, the constants $\{S(k) \mid k = 5 \ldots 20\}$ have been evaluated in terms of multiple zeta values and *multiple Clausen values* of the form [11]

$$M(a, b) \;:=\; \sum_{n_1 > n_2 > \ldots > n_b > 0} \frac{\sin(n_1 \pi/3)}{n_1^a} \prod_{j=1}^{b} \frac{1}{n_j}$$

with, for example,

$$\begin{aligned} S(9) \;=\; & \pi \left[2M(7,1) + \frac{8}{3}M(5,3) + \frac{8}{9}\zeta(2)M(5,1)\right] - \frac{13921}{216}\zeta(9) \\ & + \frac{6211}{486}\zeta(7)\zeta(2) + \frac{8101}{648}\zeta(6)\zeta(3) + \frac{331}{18}\zeta(5)\zeta(4) - \frac{8}{9}\zeta^3(3) \end{aligned}$$

The evaluation of the constant $S(20)$ is a 118-dimensional integer relation problem, which required 4800 digit arithmetic. In this case a relation was detected at iteration 27,531. The minimum and maximum y vector entry at detection were 7.170×10^{-4755} and 3.513×10^{-4375}, which gives a confidence ratio of 2.040×10^{-380}. The actual solution for this problem is shown in Table 5. In this table, irreducible multiple zeta values such $\zeta(5,3) := \sum_{j>k>0} j^{-5} k^{-3}$ occur. Moreover, there are alternating Euler sums, such as $\zeta(\overline{9},\overline{3}) := \sum_{j>k>0} (-1)^j j^{-9}\,(-1)^k k^{-3}$, where an alternating sign is indicated by a bar. The presence of the latter results from another discovery obtained with PSLQ [12], namely that some multiple zeta values may be reduced to alternating Euler sums with fewer summations. Finally, the combinations [11]

$$\zeta_A(a,b,c) := \zeta(\overline{a},\overline{b},c) + \zeta(\overline{a},b,\overline{c}) + \zeta(a,\overline{b},\overline{c})$$

serve to reduce 5-fold multiple zeta values to 3-fold alternating Euler sums.

These three problems were first solved by the second author running a three-level implementation of PSLQ on a DecAlpha machine at the Open University, with a single 433 MHz processor, and 1 Gbyte of main memory. They were then used as benchmarks for a multiprocessor version of the new three-level multi-pair program, using the OpenMP programming model, on a 64-CPU SGI Origin-2000 system at the Lawrence Berkeley Laboratory. Run times are given in Table 6. Timings on 48 processors show a speedup of 19.40 times on the Fibonacci conjecture problem, 22.44 times on the B_4 problem, and 17.81 times on the $S(20)$ problem. Given the challenge of very limited concurrency inherent in this type of calculation, we are encouraged by these figures.

A Polylogarithm Ladder Calculation: The fourth calculation arose from the discovery by the second author that

$$\alpha^{630} - 1 \;=\; \frac{(\alpha^{315}-1)(\alpha^{210}-1)(\alpha^{126}-1)^2(\alpha^{90}-1)(\alpha^3-1)^3(\alpha^2-1)^5(\alpha-1)^3}{(\alpha^{35}-1)(\alpha^{15}-1)^2(\alpha^{14}-1)^2(\alpha^5-1)^6\alpha^{68}}$$

where

$$\alpha_1 = 1.17628081825991750654407033847403505069341580656\ldots$$

is the largest real root of Lehmer's remarkable polynomial [22]

$$0 \;=\; 1 + \alpha - \alpha^3 - \alpha^4 - \alpha^5 - \alpha^6 - \alpha^7 + \alpha^9 + \alpha^{10}$$

The above cyclotomic relation was first discovered by a PSLQ computation, although subsequently proven by repeated substitution for α^{10}. This result then suggested that an integer relation may exist between a certain set of 125 related polylogarithmic constants. In particular, it was conjectured that there may be integers a, b_j, c_k such that

$$a\,\zeta(17) \;=\; \sum_{j=0}^{8} b_j\,\pi^{2j}(\log\alpha)^{17-2j} + \sum_{k\in D(\mathcal{S})} c_k\,\mathrm{Li}_{17}(\alpha^{-k})$$

where the 115 indices k in $\mathrm{Li}_n(\alpha^{-k}) := \sum_{r>0} \alpha^{-kr}/r^n$ are drawn from the set, $D(\mathcal{S})$, of positive integers that divide at least one element of

$$\mathcal{S} \;=\; \{29, 47, 50, 52, 56, 57, 64, 74, 75, 76, 78, 84, 86, 92, 96, 98, 108, 110, 118, 124, 130, \\ 132, 138, 144, 154, 160, 165, 175, 182, 186, 195, 204, 212, 240, 246, 270, 286, 360, 630\}$$

$$
\begin{aligned}
&525990827847624469523748125835264000\, S(20) = \\
&- 150244020066395453474763414663584808960000\, \pi\, M(17,2) + 6143572869260253804037378109755883520000\, \pi\, M(15,4) \\
&- 336634129822479660495198800534568960000\, \pi\, M(13,6) + 2047857623086751268012459369918627840000\, \pi\, M(15,2)\, \zeta(2) \\
&- 1122113766074932201650662668448563200 00\, \pi\, M(13,4)\, \zeta(2) - 7792456708853695844796268530892800000\, \pi\, M(13,2)\, \zeta(4) \\
&+ 6583242682954580166119734539029003325380 0417\, \zeta(20) - 1655150248639886171642409815524246277640960\, \zeta(17,3) \\
&- 874078578679726460633187922045458195456 00\, \zeta(17)\, \zeta(3) + 239001490518032437117759318070284363571904\, \zeta(15,5) \\
&+ 647549707213487635749714075958718250393 6\, \zeta(15,3)\, \zeta(2) - 1134338891089163397174552494647558181151360 0\, \zeta(15)\, \zeta(5) \\
&+ 765053105940549689685415963014774353264 64\, \zeta(15)\, \zeta(3)\, \zeta(2) - 542750687279333062134398429874111986112 0\, \zeta(14)\, \zeta^2(3) \\
&- 337251869008851810725422166365424949772 80\, \zeta(13,7) - 107923659414904307232986232333819751833 6\, \zeta(13,5)\, \zeta(2) \\
&- 504859318011860793420108954252906330624 00\, \zeta(13,3)\, \zeta(4) - 244306108799562731040229637483037115104470 40\, \zeta(13)\, \zeta(7) \\
&+ 796530831594947602965411064203762718396416\, \zeta(13)\, \zeta(5)\, \zeta(2) - 484763227029402939393977637221851473408 00\, \zeta(13)\, \zeta(4)\, \zeta(3) \\
&- 245944614254257828083385316364779520 0\, \zeta(12)\, \zeta(5,3) - 718391741998187361535584654611010700800 0\, \zeta(12)\, \zeta(5)\, \zeta(3) \\
&+ 655403673832669065999112368826215674880 0\, \zeta(11,5)\, \zeta(4) - 674581129238392279111385274785342054400\, \zeta(11,3,3,3) \\
&+ 155743130140661296228413518954716262400\, \zeta(11,3,3)\, \zeta(3) + 1385699684530152789142330538230155878400 0\, \zeta(11,3)\, \zeta(6) \\
&+ 339959536740516778440799419126460108800\, \zeta(11,3)\, \zeta^2(3) - 355430278060696093692377459977974318351225 60\, \zeta(11)\, \zeta(9) \\
&+ 1912599458053045671374932296869893271531520\, \zeta(11)\, \zeta(7)\, \zeta(2) - 862450922069301253796984760032279370240 0\, \zeta(11)\, \zeta(6)\, \zeta(3) \\
&- 159424648200337153322748394462255349760000\, \zeta(11)\, \zeta(5)\, \zeta(4) - 386372041666595966843560058208603955200\, \zeta(11)\, \zeta^3(3) \\
&- 452614452147121967588604063991726080 0\, \zeta(10)\, \zeta(7,3) - 423568412107231960503083624865797062656 0\, \zeta(10)\, \zeta(7)\, \zeta(3) \\
&- 427442756244252413519815126113264565248 0\, \zeta(10)\, \zeta^2(5) + 174910231480430088343102177690512998400\, \zeta(9,5,3,3) \\
&+ 232018518440712660805841414992478208 00\, \zeta(9,5,3)\, \zeta(3) - 538896527277543029720044315425444839424 0\, \zeta(9,5)\, \zeta(6) \\
&- 961444808023442822569623466946156544 00\, \zeta(9,5)\, \zeta^2(3) - 564799665543005814719751486159037931520\, \zeta(9,3,5,3) \\
&+ 192405432086205157974414874044727296000\, \zeta(9,3,3,3)\, \zeta(2) - 437636171132005416578131168531552665600\, \zeta(9,3,3)\, \zeta(5) \\
&+ 824102322609285791412381867013963776 00\, \zeta(9,3,3)\, \zeta(3)\, \zeta(2) - 568203098315511941673340529133785088 00\, \zeta(9,3)\, \zeta(8) \\
&- 782907501820074910179995871608832 00\, \zeta(9,3)\, \zeta(5,3) + 173223299338939829293781467642177536000\, \zeta(9,3)\, \zeta(5)\, \zeta(3) \\
&- 538946187972632260146372374750822400 0\, \zeta(9,3)\, \zeta^2(3)\, \zeta(2) + 1395360857314382550903663041050280719202304\, \zeta^2(9)\, \zeta(2) \\
&- 154345423026190013888195117210716938240 0\, \zeta(9)\, \zeta(8)\, \zeta(3) - 511939532590839950285975762448130830336000\, \zeta(9)\, \zeta(7)\, \zeta(4) \\
&+ 897851046808128210692781912394041953280 00\, \zeta(9)\, \zeta(6)\, \zeta(5) - 130913272708742090192577311318999040 0\, \zeta(9)\, \zeta(5,3)\, \zeta(3) \\
&+ 173199470860052306637152021264019271680 0\, \zeta(9)\, \zeta(5)\, \zeta^2(3) - 130913272708742090192577311318999040 0\, \zeta(9)\, \zeta(3,5,3) \\
&+ 498638663445086363059472499319111680 00\, \zeta(9)\, \zeta^3(3)\, \zeta(2) - 137974821835122835609401624298185801216 00\, \zeta(8)\, \zeta(7)\, \zeta(5) \\
&+ 205252482965220648410592154857633792 00\, \zeta(8)\, \zeta^4(3) + 533245759266957435712480647773027635200\, \zeta(7,7,3,3) \\
&+ 391575038329841217165725217164886528 00\, \zeta(7,7,3)\, \zeta(3) + 223377519430349618539918571265503416320\, \zeta(7,5,5,3) \\
&- 237007682890194482344043481035520000 00\, \zeta(7,5,5)\, \zeta(3) - 184392479550115407127835175133101465600\, \zeta(7,5,3,5) \\
&- 192646077208687087875906081369587712000\, \zeta(7,5,3,3)\, \zeta(2) + 609805989326475901096307023020578611200\, \zeta(7,5,3)\, \zeta(5) \\
&- 928830123397751571136729107189006336 00\, \zeta(7,5,3)\, \zeta(3)\, \zeta(2) - 108300423278181917035100490348625920 0\, \zeta(7,3,5,3)\, \zeta(2) \\
&+ 642228810086780199757027863429120000\, \zeta(7,3,3)\, \zeta(7) + 171516507454134257747896704172032000 0\, \zeta(7,3,3)\, \zeta(5)\, \zeta(2) \\
&- 256615593289239779065106729533440000\, \zeta^2(7,3) - 117611319397120633884272025717964800\, \zeta(7,3)\, \zeta(5,3)\, \zeta(2) \\
&+ 266618082812610684431891803668480000\, \zeta(7,3)\, \zeta^2(5) + 680547847373279252823896549003025331200 00\, \zeta^2(7)\, \zeta(6) \\
&+ 157109073239336260123589275943058723840 0\, \zeta^2(7)\, \zeta^2(3) - 101643413009734412948212276518715392 0\, \zeta(7)\, \zeta(5,5,3) \\
&- 293590313182528091317498451853312000\, \zeta(7)\, \zeta(5,3)\, \zeta(5) + 1133345755060987206065174842254330470400\, \zeta(7)\, \zeta^2(5)\, \zeta(3) \\
&+ 624199833171492314007208258301460480 00\, \zeta(7)\, \zeta(5)\, \zeta^2(3)\, \zeta(2) + 101029201230288166627783621503025152000\, \zeta(7)\, \zeta(4)\, \zeta^3(3) \\
&- 392739818126226270577731933956997120 0\, \zeta(6)\, \zeta(5,3,3,3) + 881229448062492328848398017487339520 00\, \zeta(6)\, \zeta(5)\, \zeta^3(3) \\
&- 196369909063113135288865966978498560 0\, \zeta(6)\, \zeta(3,5,3)\, \zeta(3) + 123944391418020200398288878971859763 2\, \zeta(5,5,5,3)\, \zeta(2) \\
&- 271453459130743151929004517136269312 0\, \zeta(5,5,3)\, \zeta(5)\, \zeta(2) - 785479636252452541155463867913994240 0\, \zeta(5,3,3,3,3,3) \\
&+ 785479636252452541155463867913994240 0\, \zeta(5,3,3,3,3)\, \zeta(3) - 392739818126226270577731933956997120 0\, \zeta(5,3,3,3)\, \zeta^2(3) \\
&- 117611319397120633884272025717964800\, \zeta(5,3)\, \zeta^2(5)\, \zeta(2) + 327283181771855225481443278297497600\, \zeta(5,3)\, \zeta^4(3) \\
&- 310534753804603441554226729609432166400\, \zeta^4(5) + 465956614411204439763551829521203200 00\, \zeta^3(5)\, \zeta(3)\, \zeta(2) \\
&+ 151543801845432249941675432254537728000\, \zeta^2(5)\, \zeta(4)\, \zeta^2(3) - 888340064809321326306774612521779200\, \zeta(5)\, \zeta^5(3) \\
&- 654566363543710450962886556594995200\, \zeta(3,5,3)\, \zeta^3(3) + 155849134177073916895925370617856 00\, \zeta^6(3)\, \zeta(2) \\
&+ 313388607502074445794743966572216320 00\, \zeta(8)\, \zeta(\overline{9},\overline{3}) + 185425460957386162937367443275776000 00\, \zeta(5)\, \zeta(3)\, \zeta(\overline{9},\overline{3}) \\
&+ 853771059327246083011752482989670400 0\, \zeta^2(3)\, \zeta(2)\, \zeta(\overline{9},\overline{3}) + 675871149225360968655980361248931840000\, \zeta(4)\, \zeta(\overline{13},\overline{3}) \\
&+ 2540150075377491547753898716020300840 96\, \zeta(2)\, \zeta(\overline{15},\overline{3}) - 169298087693787229141218559994961592320 0\, \zeta(\overline{17},\overline{3}) \\
&- 123616973971590775291578295517184000 00\, \zeta(5)\, \zeta_A(9,3,3) - 113836141243632811068233664398622720 00\, \zeta(3)\, \zeta(2)\, \zeta_A(9,3,3) \\
&+ 212786017863098254535236772683579392000\, \zeta(3)\, \zeta_A(11,3,3) + 174238991699437976124847445508096000\, \zeta(2)\, \zeta(6,\overline{5},\overline{4},3) \\
&+ 652422918183395758483329893007360 00\, \zeta(8,\overline{5},\overline{4},3) + 103014144976325646076315246264320000\, \zeta(6,\overline{5},\overline{6},3)
\end{aligned}
$$

Table 5: Solution for $S(20)$ found with the three-level program

	Fibonacci		B_4		$S(20)$	
Processors	Time	Speedup	Time	Speedup	Time	Speedup
1	47788	1.00	90855	1.00	23208	1.00
2	24665	1.94	46134	1.97	11973	1.94
4	12945	3.69	23966	3.79	6305	3.68
8	7076	6.75	12924	7.03	3470	6.69
16	4180	11.43	7424	12.24	2126	10.92
32	2994	15.96	4865	18.68	1548	14.99
48	2463	19.40	4049	22.44	1303	17.81

Table 6: Timings for three large problems using the parallel three-level multi-pair program

This relation was found using an implementation of the three-level multi-pair algorithm, programmed in the Message Passing Interface (MPI) [20] programming model, and run on the SGI/Cray T3E computer system at Lawrence Berkeley Laboratory. In spite of the higher latency on this distributed memory system, which presents a greater challenge for an efficient multiprocessor implementation, we were able to achieve reasonable scaling efficiency with 64 CPUs. The actual run employed 50,000 decimal digit arithmetic, and required approximately 44 hours on 32 CPUs, completing after 236,713 iterations. The minimum and maximum y entries at detection were 1.649×10^{-49718} and 1.363×10^{-36364}, respectively, which gives a confidence ratio less than 10^{-13354}. The largest of the resulting integer coefficients had 292 digits. We believe this to be the largest integer relation computation ever performed.

9. Conclusion

We have accelerated the conventional implementation of the PSLQ algorithm in three ways. First, we utilized a two-level and a three-level scheme, which permit most if not all iterations to be performed using ordinary 64-bit double precision arithmetic, and updating the multiprecision arrays only as needed. This resulted in a speedup of up to 65 times over the straightforward one-level program. Secondly, we developed a new integer relation algorithm, a variant of PSLQ that we have termed the "multi-pair" algorithm. We also demonstrated two-level and three-level implementations of this new algorithm. These techniques resulted in an additional speedup of up to 22%, comparing the three-level multi-pair program to the three-level PSLQ program. Finally, we showed how that this new algorithm, unlike PSLQ, is reasonably well suited for parallel processing. We demonstrated a parallel three-level implementation of the multi-pair algorithm that achieved an *additional* speedup of up to 22 times.

We have also applied these programs to four large integer relation problems, obtaining results that were not previously known in the literature, and which would have required years of computation using more conventional means. We believe that these demonstrations open up a novel way of doing pure and applied mathematics. We are confident that many more discoveries can be made in this manner.

References

[1] David H. Bailey, "Multiprecision Translation and Execution of Fortran Programs", *ACM Transactions on Mathematical Software*, vol. 19, no. 3, 1993, pp. 288–319.

[2] David H. Bailey, "A Fortran-90 Based Multiprecision System", *ACM Transactions on Mathematical Software*, vol. 21, no. 4, 1995, pg. 379-387. This software and documentation is available from the URL `http://www.nersc.gov/~dhbailey`.

[3] David H. Bailey, Jonathan M. Borwein and Roland Girgensohn, "Experimental Evaluation of Euler Sums", *Experimental Mathematics*, vol. 4, no. 1, 1994, pp. 17–30.

[4] David H. Bailey, Peter B. Borwein and Simon Plouffe, "On The Rapid Computation of Various Polylogarithmic Constants", *Mathematics of Computation*, vol. 66, no. 218, 1997, pp. 903–913.

[5] David H. Bailey and David Broadhurst, "A Seventeenth-Order Polylogarithm Ladder". This manuscript is available from the URL `http://xxx.lanl.gov/abs/math.NA/9905048`.

[6] David Borwein and Jonathan M. Borwein, "On An Intriguing Integral and Some Series Related to $\zeta(4)$", *Proceedings of the American Mathematical Society*, vol. 123, 1995, pp. 111-118.

[7] David Borwein, Jonathan M. Borwein and Roland Girgensohn, "Explicit Evaluation of Euler Sums", *Proceedings of the Edinburgh Mathematical Society*, vol. 38, 1995, pp. 277–294.

[8] Jonathan M. Borwein and Peter B. Borwein, *Pi and the AGM*, John Wiley, New York, 1987.

[9] Jonathan M. Borwein, David M. Bradley and David J. Broadhurst, "Evaluations of k-fold Euler/Zagier Sums: A Compendium of Results for Arbitrary k", *Electronic Journal of Combinatorics*, vol. 4, no. 2, 1997, #R5.

[10] Jonathan M. Borwein, David M. Bradley, David J. Broadhurst and Petr Lisonek, "Combinatorial Aspects of Multiple Zeta Values", *Electronic Journal of Combinatorics*, vol. 5, no. 1, 1998, #R38.

[11] Jonathan M. Borwein and David J. Broadhurst, "Apéry-like Reductions to Multiple Clausen Values and Euler Sums", in preparation.

[12] David J. Broadhurst, John A. Gracey and Dirk Kreimer, "Beyond the Triangle and Uniqueness Relations: Non-zeta Counterterms at Large N from Positive Knots", *Zeitschrift für Physik*, vol. C75, 1997, pp. 559–574.

[13] David J. Broadhurst and Dirk Kreimer, "Association of Multiple Zeta Values with Positive Knots via Feynman Diagrams up to 9 Loops", *Physics Letters*, vol. B383, 1997, pp. 403–412.

[14] David J. Broadhurst, "Massive 3-loop Feynman Diagrams Reducible to SC* Primitives of Algebras of the Sixth Root of Unity", preprint, March 1998, to appear in *European Physical Journal C*. The manuscript is available from the URL `http://xxx.lanl.gov/abs/hep-th/9803091` .

[15] David J. Broadhurst, "Polylogarithmic Ladders, Hypergeometric Series and the Ten Millionth Digits of $\zeta(3)$ and $\zeta(5)$', preprint, March 1998. The manuscript is available from the URL `http://xxx.lanl.gov/abs/math/9803067` .

[16] Sid Chatterjee and Herman Harjono, "MPFUN++: A Multiple Precision Floating Point Computation Package in C++", University of North Carolina, Sept. 1998. This software is available from the URL `http://www.cs.unc.edu/Research/HARPOON/mpfun++`.

[17] Jack J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software", University of Tennessee Computer Science Technical Report, CS-89-85, 1999. The Linpack software is available from the URL `http://www.netlib.org/linpack`.

[18] Helaman R. P. Ferguson, David H. Bailey and Stephen Arno, "Analysis of PSLQ, an Integer Relation Finding Algorithm", *Mathematics of Computation*, to appear.

[19] Helaman R. P. Ferguson and Rodney W. Forcade, "Generalization of the Euclidean Algorithm for Real Numbers to All Dimensions Higher Than Two", *Bulletin of the American Mathematical Society*, vol. 1, 1979, pp. 912–914.

[20] William Gropp, Ewing Lusk and Anthony Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, Mass., 1996.

[21] J. Hastad, B. Just, J. C. Lagarias and C. P. Schnorr, "Polynomial Time Algorithms for Finding Integer Relations Among Real Numbers", *SIAM Journal of Computing*, vol. 18, 1989, pp. 859–881.

[22] Derrick H. Lehmer, "Factorization of Certain Cyclotomic Functions", *Annals of Mathematics*, vol. 34, 1933, pp. 461–479.