RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe

Bidirectional Exact Integer Divison

Werner KRANDICK, Tudor JEBELEAN

(August 11, 1994)

RISC-Linz Report Series No. 94-50

Supported by: National Science Foundation (FWF), project no. M0135-PHY, Lise-Meitner Stipendium, and project no. P10002.

Published in: Proceedings, PASCO'94, World Scientific Publ. Comp.

Proceedings of PASCO'94

Bidirectional Exact Integer Division

Werner KRANDICK* Tudor JEBELEAN†
Research Institute for Symbolic Computation
Johannes Kepler University, A-4040 Linz, Austria
krandick, jebelean@risc.uni-linz.ac.at

Abstract: Division of integers is called *exact* if the remainder is zero. We show that the high-order part and the low-order part of the exact quotient can be computed independently from each other. A sequential implementation of this algorithm is up to twice as fast as ordinary exact division and four times as fast as the general classical division algorithm if the dividend is twice as long as the divisor. A shared-memory parallel implementation on two processors gains another factor of two in speed.

Keywords: exact arithmetic, parallel arithmetic

1 Introduction

Division of integers is called *exact* if the remainder is zero. Exact division arises systematically in exact calculations, e.g. when rational numbers are added or when the primitive part of an integral polynomial is computed.

Traditionally, these divisions are performed using a general quotient-remainder algorithm (e. g. Knuth's "Algorithm D" [6]), and then discarding the remainder. The number of digit multiplications required by this algorithm is

$$T_D(m,n) = n(m-n+1),$$

where m, n are the numbers of digits in dividend and divisor, respectively. The amount of work is suggested by the shaded area in Figure 1.

In 1993, Jebelean [4] proposed an algorithm for exact division which determines the quotient digits from right to left and requires only

$$T_J(m,n) = \begin{cases} \frac{m-n+4}{2}(m-n+1)-1 & \text{if } m \leq 2n-1 \\ n(m-\frac{3}{2}n+\frac{1}{2})+m & \text{otherwise} \end{cases}$$

digit multiplications (see Corollary 2).

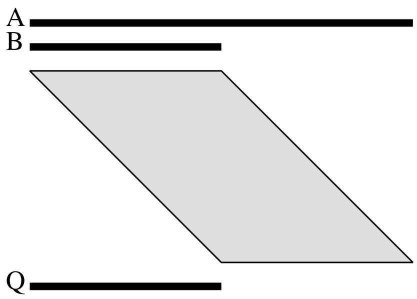

Figure 1: Classical division with remainder.

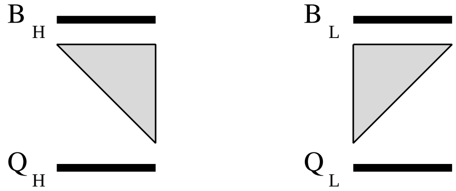

Figure 2: Bidirectional exact division.

*Supported by the Austrian Science Foundation (Grant M0135-PHY).
†Supported by the Austrian Science Foundation (Grant P10002-PHY).

In 1994, Krandick devised an algorithm for the seemingly unrelated problem of multiple precision floating point division. His method requires typically

$$T_K(m,n) = \begin{cases} n(m-n+1) & \text{if } n \leq 3 \\ \frac{m-n+6}{2}(m-n+1) & \text{if } n > 3 \wedge \\ & \quad m \leq 2n-4 \\ n(m-\frac{3}{2}n+\frac{7}{2})-3 & \text{otherwise} \end{cases}$$

digit multiplications (see Corollary 1). The method is modeled after Knuth's algorithm, but it does not compute the full remainder. Instances where this would lead to an incorrect result are detected by testing a condition sufficient for correctness. The condition is efficiently computable, and it is satisfied with very high probability — if dividend and divisor are chosen at random. If, however, the remainder is zero, the condition will not be satisfied. Thus, Krandick's method cannot be used to compute all the digits of the exact quotient.

We will present an algorithm which uses Krandick's method to compute the high-order part of the exact quotient, and Jebelean's method to compute the low-order part. The combined method requires

$$T_{KJ}(m,n) \leq \begin{cases} m & \text{if } n = 1 \\ \frac{(m-n)(m-n+11)}{4} + \frac{19}{8} & \text{if } n > 3 \wedge \\ & \quad m \leq 3n-6 \\ n(m-2n+5)-5 & \text{otherwise} \end{cases}$$

digit multiplications (see Theorem 1). In particular, if the dividend is twice as long as the divisor our method is almost four times as fast as the traditional method ($T_{KJ}(m,n) \leq n(n+11)/4 + 19/8$, see also Figure 2). The high order part Q_H of the quotient is computed using the high order part B_H of the divisor, while the low order part Q_L of the quotient is computed using the low order part B_L of the divisor. The method is well suited for coarse-grain parallelization, because the two computations are completely independent. When run in parallel on two processors, each processor has to compute at most

$$T^*_{KJ}(m,n) \leq \begin{cases} \frac{2m+1}{3} & \text{if } n = 1 \\ \frac{n^2(m-n+1)+mn}{2n+1} & \text{if } n = 2,3 \\ \frac{(m-n+11)(m-n+1)}{8} & \text{if } n > 3 \wedge \\ & \quad m \leq 3n-6 \\ \frac{n(m-2n+6)}{2} - 3 & \text{if } n > 3 \wedge \\ & \quad 3n-6 < m \leq \\ & \quad 4n-6 \\ \frac{n(m-2n+2)}{2} + \frac{m}{2} & \text{if } n > 3 \wedge \\ & \quad 4n-6 < m. \end{cases}$$

digit multiplications (see Theorem 2). In particular, if the dividend is twice as long as the divisor, the parallel version of our method is almost eight times as fast as the traditional method ($T^*_{KJ}(m,n) \leq (n+1)(n+11)/8$).

The idea of combining a high-order algorithm with Jebelean's exact division was first suggested to us by Schönhage, who recently investigated an implementation and applications together with Vetter [10]. In contrast to his approach we use a different method for computing the high-order part of the quotient, we analyze the method in terms of the required number of digit products, we provide detailed empirical data, and we discuss and implement a parallel version of the method.

Coarse level parallelization of long integer division is apparently not treated in the literature. Parallel algorithms for division refer mostly to fixed-point fractions, and are designed at the level of bit processing. One research direction in this area is the theoretical investigation of time and area complexity of division on parallel computing models such as parallel random access memory (PRAM); for a survey see [9], Section 3.7. Research with practical applications is mainly VLSI-oriented (see e.g. [11]), again at bit-level. Word-level algorithms are based on the systolic approach (see e.g. [5]), which is too fine-grained for shared-memory architectures. Our algorithm, although not scalable, is suitable for coarse-grain parallelization on shared-memory machines and it will increase the performance of parallel algebraic algorithms which contain exact division as a subalgorithm.

Under a title quite similar to the title of this paper Vacariu [12] treats the computation of the quotient of fixed-point numbers at bit level. He uses the term "exact quotient" to refer to the representation of the quotient as a periodic fraction. The computation is performed from both directions, on two parallel processors, but it is organized according to a "master-slave" scheme, with communication at each step. In our algorithm only final synchronization is needed.

Section 2 describes Krandick's algorithm, Section 3 reviews Jebelean's method. Section 4 combines the two methods to maximize the performance of a sequential implementation. Section 5 shows how a parallel implementation best combines the two methods. Section 6 compares the new method empirically with the algorithms by Knuth and Jebelean.

2 Computing the high-order part

Given positive integers

$$A = \sum_{i=0}^{m-1} a_i\beta^i, \qquad B = \sum_{j=0}^{n-1} b_j\beta^j, \qquad (1)$$

of β-*length* $m, n \geq 1$ and with β-*digits* $0 \leq a_i, b_j < \beta$ and $a_{m-1}, b_{n-1} > 0$, we want to find the h high-order digits of $Q = \lfloor A/B \rfloor$, where $1 \leq h \leq m-n+1$. Here we consider Q as having $m-n+1$ digits, with the high-order digit being possibly zero. In other words, we want to compute

$$\bar{Q} = \left\lfloor \frac{A}{\bar{B}} \right\rfloor,$$

where $\bar{B} = B\beta^{m-n+1-h}$.

Traditional classical division requires hn digit multiplications for this task. We will describe a method that typically requires only $h(h+5)/2$ digit multiplications when $n > 3$ and $h \leq n-3$. The savings are obtained by suppressing the computation of the remainder.

Instead of computing $\bar{Q}$ and $\bar{R}$, $0 \leq \bar{R} < \bar{B}$, with

$$A = \bar{Q} \cdot \bar{B} + \bar{R}$$

we compute Q^* and R^* such that

$$A = Q^* \circ \bar{B} + R^*,$$

where $Q^* \circ \bar{B}$ is a well-defined approximation to the product $Q^*\bar{B}$. Computing $Q^* \circ \bar{B}$ requires fewer digit products than computing the exact product $Q^*\bar{B}$.

We will define the approximate product $\circ$ and derive an error bound

$$\epsilon \geq Q^*\bar{B} - Q^* \circ \bar{B}.$$

We will state a condition involving ϵ and R^* which will be easy to test and which will imply equality of Q^* and $\bar{Q}$. We will argue that Q^* can be determined in such a way that the sufficient condition will be satisfied with high probability.

Definition 1 *Let A, B be integers as in (1) with B having n digits. We define the* approximate product

$$A \circ B = \sum_{i+j \geq n-3} a_i b_j \beta^{i+j}.$$

Clearly, for $n = 1, 2$ or 3, the approximate product coincides with the exact product. In the notation of Krandick and Johnson [8], $A \circ B = (A \times_{n-1} B)\beta^{n-3}$ where $\times_{n-1}$ denotes "the short product with respect to $(n-1)$". The deviation from the exact product can be estimated as follows.

Proposition 1 *Let A,B as in (1) with B having n digits, let*

$$\delta = \begin{cases} 0 & \textit{if } n = 1, 2 \\ (n-3)\beta^{n-2} & \textit{if } n \geq 3. \end{cases}$$

Then

$$A \circ B \leq AB \leq A \circ B + \delta.$$

Proof. The cases $n = 1, 2$ are trivial. By induction on $n \geq 3$,

$$\sum_{i+j<n-3} (\beta-1)^2\beta^{i+j} \leq (n-3)\beta^{n-2} + (2-n)\beta^{n-3} + 1.$$

The right hand side is $\leq \delta$. This proves the second inequality. □

The proposed division method uses the approximate product of Q^* and $\bar{B}$. Noting that $\bar{B}$ has $m-h+1$ digits we obtain

$$Q^* \circ \bar{B} = \sum_{i+j \geq m-h-2} q_i^* \bar{b}_j \beta^{i+j}, \qquad (2)$$

with $Q^* = \sum_{i=0}^{h-1} q_i^*\beta^i$, $0 \leq q_i^* < \beta$ and with $\bar{B} = \sum_{j=0}^{m-h} \bar{b}_j\beta^j$, $0 \leq \bar{b}_j < \beta$. Furthermore, we have

$$Q^* \circ \bar{B} \leq Q^*\bar{B} \leq Q^* \circ \bar{B} + \epsilon, \qquad (3)$$

where

$$\epsilon = \begin{cases} 0 & \text{if } m-h = 0, 1 \\ (m-h-2)\beta^{m-h-1} & \text{if } m-h \geq 2. \end{cases} \qquad (4)$$

We now give a sufficient condition for the success of the method.

Proposition 2 *Let ϵ as in (4). If*

$$\epsilon \leq R^* < \bar{B} \qquad (5)$$

then

$$Q^* = \bar{Q}.$$

Proof. Let

$$R^{**} = A - Q^*\bar{B}$$

and assume (5). Then (3) implies

$$R^{**} = (Q^* \circ \bar{B} + R^*) - Q^*\bar{B} \geq (Q^* \circ \bar{B} + \epsilon) - Q^*\bar{B} \geq 0$$

and

$$R^{**} \leq A - Q^* \circ \bar{B} = R^* < \bar{B}.$$

Hence, $A = Q^*\bar{B} + R^{**}$ with $0 \leq R^{**} < \bar{B}$, so Q^* must be the desired quotient $\bar{Q}$. □

The proposed division method will produce a Q^* such that $A = Q^* \circ \bar{B} + R^*$. It will be shown that $Q^* = \bar{Q}$ with high probability. Condition (5) will be used to establish $Q^* = \bar{Q}$ with certainty. Therefore, the following question has to be discussed. Given $Q^* = \bar{Q}$, what is the probability that condition (5) is satisfied? — Because of (3) we have

$$A - Q^* \circ \bar{B} \geq A - Q^* \bar{B} \geq A - Q^* \circ \bar{B} - \epsilon.$$

Hence, letting $Q^* = \bar{Q}$ in the middle,

$$R^* \geq \bar{R} \geq R^* - \epsilon$$

or, equivalently,

$$\bar{R} \leq R^* \leq \bar{R} + \epsilon.$$

Thus, condition (5) will be satisfied if

$$\epsilon \leq \bar{R} < \bar{B} - \epsilon. \tag{6}$$

If all possible values $0, \ldots, \bar{B}-1$ for $\bar{R}$ are equally likely, (6) is true with probability

$$\frac{\bar{B} - 2\epsilon}{\bar{B}} \geq 1 - \frac{2(m-h-2)\beta^{m-h-1}}{\bar{B}}$$

$$\geq 1 - \frac{2(m-h-2)}{\beta}.$$

This value is very close to 1 if β is the word size of a computer (e.g. $\beta = 2^{32}$).

The digits of Q^* are determined by Algorithm H in Figure 3 analogously to Knuth's "Algorithm D" ([6], p.257f). We assume radix β to be a power of 2, for example the word size of the computer. We represent β-digits as words; hence we may refer in step H1 to the number of leading 0-bits of a β-digit. The normalization is effected by a binary shift which is applied to all digits of B, but only to those digits of A that will be needed in step H9. Steps H3 and H4 together subtract $[b_{n-1}, \ldots, b_J] * q_k$ from $[a_i, \ldots, a_{k+J}]$ with $J = \max(0, m-h-2-k)$. In total, the loop subtracts $[q_{m-n}, \ldots, q_{m-n-h+1}] \circ [b_{n-1}, \ldots, b_0, 0, \ldots, 0] = [q_{m-n}, \ldots, q_{m-n-h+1}] \circ [\bar{b}_{m-h}, \ldots, \bar{b}_{m-n-h+1}, \bar{b}_{m-n-h}, \ldots, \bar{b}_0] = Q^* \circ \bar{B}$ from $[a_m, \ldots, a_0]$. This motivates the analysis in Proposition 3. Steps H8 and H9 test for the condition of Proposition 2.

We will now argue that the loop in Algorithm H will produce $\bar{Q}$ with high probability. Knuth shows in exercise 4.3.1.21 of his book [6] that step D3 of his algorithm will fail to supply the correct quotient digit with approximate probability $2/\beta$. This number is very small when β is the word size of a computer. For Algorithm H this means that quotient digit q_{i-n} calculated in step H3 will be correct in almost all cases where the three leading digits a_i, a_{i-1}, a_{i-2} of the current "remainder" are correct. We will use this result in an induction argument to show that all digits q_{i-n} are most likely correct. We start with the observation that q_{m-n} is correct. The induction argument states that if the quotient digits $q_{m-n}, \ldots, q_{i-n}$ are correct for an $m \geq i \geq m-h+2$, it is highly probable that $[a_i, a_{i-1}, a_{i-2}]$ is correct. Indeed, the probability that the correctness of $q_{m-n}, \ldots, q_{i-n}$ implies the correctness of

Algorithm H (*High-order part of quotient*). Let A, B as in (1) with m and n digits, respectively, and let $1 \leq h \leq m-n+1$.
The algorithm will compute the h high-order digits $q_{m-n}, \ldots, q_{m-n-h+1}$ of $\lfloor A/B \rfloor$ or fail. The probability of a failure is very small.
We may assume $m \geq n > 1 \wedge (m = n \Rightarrow a_{m-1} \geq b_{n-1})$.

H1. [Normalize.] Set $d \leftarrow$ the number of leading 0-bits in b_{n-1}, $I \leftarrow \max(0, m-n-h)$. Then perform the left-shifts
$[a_m, a_{m-1}, \ldots, a_I] \leftarrow [a_{m-1}, \ldots, a_I] * 2^d$
and $[b_{n-1}, \ldots, b_0] \leftarrow [b_{n-1}, \ldots, b_0] * 2^d$.

H2. [Initialize loop.] Set $k \leftarrow m-n$, $i \leftarrow m$.

H3. [Calculate quotient digit.]
If $a_i = b_{n-1}$, set $q_k \leftarrow \beta - 1$; otherwise set $q_k \leftarrow \lfloor [a_i, a_{i-1}]/b_{n-1} \rfloor$.
Subtract $b_{n-1} q_k$ from $[a_i, a_{i-1}]$; then subtract $b_{n-2} q_k$ from $[a_i, a_{i-1}, a_{i-2}]$. If this leaves a_i negative, decrement q_k by 1 and add $[b_{n-1}, b_{n-2}]$ to $[a_i, a_{i-1}, a_{i-2}]$.

H4. [Multiply and subtract.]
Set $J \leftarrow \max(0, m-h-2-k)$; then subtract $[b_{n-3}, \ldots, b_J] * q_k$ from $[a_i, \ldots, a_{k+J}]$.

H5. [Test remainder.] If $a_i \geq 0$, go to step H7.

H6. [Add back.] Decrement q_k by 1;
add $[b_{n-1}, \ldots, b_J]$ to $[a_i, \ldots, a_{k+J}]$.

H7. [Loop on k.] If $k > m-n-h+1$, decrement k and i by 1 and go back to H4.

H8. [Remainder too small?]
If $a_{m-h} = 0 \wedge a_{m-h-1} < n-2$, fail.

H9. [Remainder too large?]
If $[a_{m-h}, \ldots, a_{m-n+1-h}] \geq [b_{n-1}, \ldots, b_0]$, fail.

Figure 3: Computing the high-order part of the quotient.

$[a_i, a_{i-1}, a_{i-2}]$ is smallest for $i = m-h+2$. For $i = m-h+2$ this probability can be bounded by the probability that the first $h+1$ digits of $\bar{Q} \circ \bar{B}$ agree with the the first $h+1$ digits of $\bar{Q}\bar{B}$. This is certainly the case if adding ϵ to $\bar{Q} \circ \bar{B}$ does not produce a carry into the $h+1$ high-order digits. In fact, adding ϵ to $\bar{Q} \circ \bar{B}$ will — with a probability of $< \frac{m-h-2}{\beta}$ — not even generate a carry into the $h+2$ high-order digits: If $a_{m-h-1} < \beta - (m-h-2)$, $[a_{m-h+1}, a_{m-h}]$ will be correct; assuming that all numbers $0, \ldots, \beta-1$ are equally likely as values of a_{m-h-1} (cf. [7]), the probability that $a_{m-h-1} < \beta-(m-h-2)$ is $1 - \frac{m-h-2}{\beta}$. Finally, since $[a_{m-h+1}, a_{m-h}, a_{m-h-1}]$ deviates from its true value by at most $m-h-2$, even $q_{m-n-h+1}$ will most likely be correct. Thus, for all practical purposes, the loop in Algorithm H will produce $\bar{Q}$ with a probability in the neighborhood of $1 - 1/\beta$.

Proposition 3 *The number $\mu(n, h)$ of digit products in formula (2) is*

$$\mu(n,h) = \begin{cases} nh & \text{if } n \le 3 \\ \frac{h(h+5)}{2} & \text{if } n > 3 \wedge h \le n-3 \\ \frac{2hn-n^2+5n-6}{2} & \text{if } n > 3 \wedge h > n-3. \end{cases} \tag{7}$$

Proof. For each index $i = 0, \ldots, h-1$ index j ranges from $\max(0, m-h-2-i)$ to $m-h$, but since $\bar{b}_0 = \ldots = \bar{b}_{m-h-n} = 0$, only the $j \ge m-h-n+1$ have to be considered. Hence the number μ_i of digit products for a given i is

$$\begin{aligned} \mu_i &= (m-h)+1- \\ &\quad \max((m-h)-(n-1), 0, (m-h)-(2+i)) \\ &= (m-h)+1- \\ &\quad \max(0, (m-h)-\min(n-1, 2+i)). \end{aligned}$$

The expression can be simplified by distinguishing two cases.

1. In case $i \ge n-3$ we have $n-1 \le 2+i$, so

   $$\mu_i = (m-h) - \max(0, (m-h)-(n-1)) + 1.$$

   Noting that $(m-h)-(n-1) \ge 0$ since $h \le m-n+1$, we obtain

   $$\mu_i = (m-h) - ((m-h)-(n-1)) + 1 = n.$$

2. In case $i < n-3$ we have $n-1 > 2+i$, so

   $$\mu_i = (m-h) - \max(0, (m-h)-(2+i)) + 1.$$

   Since $i < n-3 \le m-h-2$, the expression evaluates to

   $$\mu_i = (m-h) - ((m-h)-(2+i)) + 1 = 3+i.$$

Knowing the μ_i we can now verify the three cases of the proposition.

1. In case $n \le 3$ we have $n-3 \le 0 \le h-1$, so all i are $\ge n-3$, hence all $\mu_i = n$ and so there are

   $$\mu(n,h) = \sum_{i=0}^{h-1} \mu_i = nh$$

   digit products.

2. In case $n > 3 \wedge h \le n-3$ we have $0 \le h-1 < n-3$, so all indices i are $< n-3$, hence all $\mu_i = 3+i$ and so there are

   $$\mu(n,h) = \sum_{i=0}^{h-1} (3+i) = \frac{h(h+5)}{2}$$

   digit products.

3. In case $n > 3 \wedge h > n-3$ we have $0 < n-3 \le h-1$, so there are

   $$\begin{aligned} \mu(n,h) &= \sum_{i=0}^{n-4} (3+i) + \sum_{i=n-3}^{h-1} n \\ &= \frac{2hn-n^2+5n-6}{2} \end{aligned}$$

   digit products. $\square$

Corollary 1 *Letting $h = m-n+1$ in Proposition 3 we obtain*

$$T_K(m,n) = \begin{cases} n(m-n+1) & \text{if } n \le 3 \\ \frac{m-n+6}{2}(m-n+1) & \text{if } n > 3 \wedge \\ & \quad m \le 2n-4 \\ n(m - \frac{3}{2}n + \frac{7}{2}) - 3 & \text{otherwise.} \end{cases}$$

3 Computing the low-order part

Let A and B as in (1), and assume that B divides A. Jebelean's exact division algorithm [4] exploits the implication

$$(A'\beta + a_0) = (B'\beta + b_0) * (Q'\beta + q_0)$$

$$\Downarrow$$

$$a_0 = (b_0 q_0) \bmod \beta.$$

The latter equation can be used to compute q_0

$$q_0 = (a_0 (b_0)^{-1}_{\text{mod } \beta})_{\text{mod } \beta}$$

— provided $\text{GCD}(b_0, \beta) = 1$. When β is a power of 2 this condition can be ensured by shifting A

Algorithm L (*Low-order part of quotient*). Let A, B be as in (1) with m and n digits, respectively, with $A \bmod B = 0$ and with $b_0 \neq 0$, and let $1 \leq l \leq m-n+1$. The algorithm will compute the l low-order digits $q_{l-1}, \ldots, q_0$ of $Q = A/B$.

L1. [Right-shift.]
Set $d \leftarrow$ the number of trailing 0-bits in b_0, and set $L \leftarrow \min(n, l)$.
Then perform the right-shifts
$[a_{l-1}, \ldots, a_1] \leftarrow [a_l, a_{l-1}, \ldots, a_0]/2^d$ and $[b_{L-1}, \ldots, b_0] \leftarrow [b_L, b_{L-1}, \ldots, b_0]/2^d$.

L2. [Compute modular inverse.]
Set $b' \leftarrow (b_0)^{-1}_{\bmod \beta}$.

L3. [Initialize loop.] Set $k \leftarrow 0$.

L4. [Calculate quotient digit.]
Set $q_k \leftarrow (b' * a_k)_{\bmod \beta}$.

L5. [Test termination] If $k = l - 1$ then STOP.

L6. [Multiply and subtract.]
Set $J \leftarrow \min(L, l - k)$; then subtract $[b_{J-1}, \ldots, b_0] * q_k$ from $[a_{l-1}, \ldots, a_k]$.

L7. [Loop.] Increment k and go back to step L4.

Figure 4: Computing the low-order part of the quotient.

and B to the right until b_0 becomes odd. After the least-significant quotient digit q_0 has been found, A is replaced by

$$(A - q_0 B)/\beta = Q' * B,$$

and the procedure is repeated to find q_1, and so on. This method is faster than the traditional classical algorithm, because only the l low order digits of the intermediate results $A - q_0 B$ etc. have to be computed in order to determine the l low order digits of the quotient. Algorithm L in Figure 4 takes advantage of this insight. We may assume that the least-significant β-digit of B is non-zero; indeed, A must have at least as many trailing zero β-digits as B, and common trailing zeros can be deleted without affecting the quotient.

When analyzing Algorithm L, we will not consider the (constant) cost of finding the modular inverse of b_0 in step L2. It is shown in [4] that b_0 can be inverted using one or two digit multiplications and a table look-up when β is a power of 2; the extended Euclidean algorithm need not be applied. In our experiments the modular inverse costs 2.25 digit products.

Proposition 4 *The number $\nu(n, l)$ of digit products in Algorithm L is*

$$\nu(n,l) = \begin{cases} \frac{l(l+3)}{2} - 1 & \text{if } l \leq n \\ l(n+1) - \frac{n^2-n+2}{2} & \text{if } l > n. \end{cases} \qquad (8)$$

Proof. Let $L = \min(n, l)$. We will show that

$$\nu(n,l) = l(L+1) - \frac{L^2 - L + 2}{2}. \qquad (9)$$

From Algorithm L we obtain

$$\begin{aligned} \nu(n,l) &= l + \sum_{k=0}^{l-2} \min(L, l-k) \\ &= l + \sum_{k=0}^{\min(l-2,l-L)} L + \\ &\qquad \sum_{k=\max(0,\min(l-2,l-L)+1)}^{l-2} (l-k) \\ &= l + \sum_{k=0}^{l-\max(2,L)} L + \\ &\qquad \sum_{k=\max(0,l-\max(2,L)+1)}^{l-2} (l-k). \end{aligned}$$

1. If $L = 1$ we have

$$\begin{aligned} \nu(n,l) &= l + \sum_{k=0}^{l-2} 1 + \sum_{k=l-1}^{l-2} (l-k) \\ &= l + (l-1) + 0 = 2l - 1. \end{aligned}$$

2. If $L = 2$ we have

$$\begin{aligned} \nu(n,l) &= l + \sum_{k=0}^{l-2} 2 + \sum_{k=l-1}^{l-2} (l-k) \\ &= l + 2(l-1) + 0 = 3l - 2. \end{aligned}$$

3. If $L > 2$ we have

$$\nu(n,l) = l + \sum_{k=0}^{l-L} L + \sum_{k=l-L+1}^{l-2} (l-k)$$

$$= l + (l - L + 1)L + \frac{(L-2)(L+1)}{2}.$$

In all three cases equation (9) is satisfied; and equation (9) implies equation (8). □

Corollary 2 *Letting $l = m - n + 1$ in Proposition 4 we obtain*

$$T_J(m,n) = \begin{cases} \frac{m-n+4}{2}(m-n+1) - 1 & \text{if } m \leq 2n-1 \\ n(m - \frac{3}{2}n + \frac{1}{2}) + m & \text{otherwise.} \end{cases}$$

This result corrects the analysis given in the original paper [4], which did not account for the digit multiplications in step L4 of the algorithm.

4 Sequential exact division

The digits of the exact quotient can be computed sequentially by first using Algorithm H to calculate the high-order part of the quotient and then Algorithm L for the low-order part. This is most efficient when the quotient is split in such a way that the combined number of digit products is minimized.

Definition 2 *Let $\mu(n,h)$ as in (7), $\nu(n,l)$ as in (8), and let $\mu(n,0)=\nu(n,0)=0$. Now define* $T_{KJ}(m,n) =$

$$\min_{0\le h\le m-n+1} \mu(n,h)+\nu(n,m-n+1-h).$$

In order to avoid a profusion of unproductive case distinctions we only give an upper bound for $T_{KJ}(m,n)$.

Theorem 1

$$T_{KJ}(m,n) \le \begin{cases} m & \textit{if } n=1 \\ \frac{(m-n)(m-n+11)}{4}+\frac{19}{8} & \textit{if } n>3 \wedge m\le 3n-6 \\ n(m-2n+5)-5 & \textit{otherwise} \end{cases}$$

Proof. Let

$$h = \begin{cases} m & \text{if } n=1 \\ \lfloor\frac{m-n}{2}\rfloor & \text{if } n>3 \wedge m\le 3n-6 \\ m-2n+2 & \text{otherwise,} \end{cases} \tag{10}$$

and let $l = m-n+1-h$. Now the desired inequality is obtained by bounding $\mu(n,h)+\nu(n,l)$ from above. For easy application of equations (7) and (8) we distinguish the following cases.

1. In case $n=1$ the result is straightforward.
2. In case $n>3\wedge m\le 3n-6$ let $\tilde h=(m-n)/2$ and $\tilde l=(m-n+3)/2$. Then $h\le\tilde h\le n-3$, $l\le\tilde l<n$ and
$$\mu(n,h)+\nu(n,l)\le\mu(n,\tilde h)+\nu(n,\tilde l) = \frac{m-n}{4}(m-n+11)+\frac{19}{8}.$$
3. In case $n>3\wedge m=3n-5$ we have $h=n-3$, $l=n-1$, and $\mu(n,h)+\nu(n,l)=n^2-5=n(m-2n+5)-5$.
4. In case $n>3\wedge m>3n-5$ we have $h>n-3$, $l=n-1$ and $\mu(n,h)+\nu(n,l)=n(m-2n+5)-5$.
5. In case $n=2,3$, $h=m-2n+2$, $l=n-1$ and $\mu(n,h)+\nu(n,l)=\frac{2mn-3n^2+5n-4}{2}$. For $n=2,3$ this equals $n(m-2n+5)-5$. □

5 Parallel exact division

The high-order and the low-order part of the exact quotient can be computed by executing Algorithm H and Algorithm L in parallel on two processors. This is most efficient when the quotient is split in such a way that the number of digit products in either algorithm is minimized.

Definition 3 *Let $\mu(n,h)$ and $\nu(n,l)$ as in Definition (2). Define* $T^*_{KJ}(m,n)=$

$$\min_{0\le h\le m-n+1} \max(\mu(n,h),\nu(n,m-n+1-h)).$$

For simplicity we only give an upper bound for $T^*_{KJ}(m,n)$.

Theorem 2

$$T^*_{KJ}(m,n) \le \begin{cases} \frac{2m+1}{3} & \textit{if } n=1 \\ \frac{n^2(m-n+1)+mn}{2n+1} & \textit{if } n=2,3 \\ \frac{(m-n+11)(m-n+1)}{8} & \textit{if } n>3 \wedge m\le 3n-6 \\ \frac{n(m-2n+6)}{2}-3 & \textit{if } n>3 \wedge 3n-6<m\le 4n-6 \\ \frac{n(m-2n+2)}{2}+\frac{m}{2} & \textit{if } n>3 \wedge 4n-6<m. \end{cases}$$

Proof. Let

$$h = \begin{cases} \left\lceil\frac{n+1}{2n+1}(m-n)\right\rceil & \text{if } n\le 3 \\ \left\lceil\frac{m-n}{2}\right\rceil & \text{otherwise,} \end{cases} \tag{11}$$

and let $l=m-n+1-h$.

We first prove the theorem for the case $n\le 3$.

1. The first branch of ν in (8) is only relevant if $n=1$ and $m=1,2,3$ or if $n=2$ and $m=2,\ldots,6$ or if $n=3$ and $m=3,\ldots,9$. In each of these 15 cases the theorem can be verified explicitely.
2. If m and n are such that ν is defined by its second branch in (8), we handle the ceiling function in the definition of h by letting
$$\tilde h = \frac{(n+1)(m-n)+2n}{2n+1}\ge h.$$
Since the first branch of μ in (7) is monotone increasing in h we have
$$\mu(n,h)\le\mu(n,\tilde h)=\frac{n^2(m-n+1)+mn}{2n+1}.$$
Furthermore, let
$$\tilde l = m-n+1-\frac{n+1}{2n+1}(m-n)\ge l.$$

Now $\nu(n,l) \le \nu(n,\tilde{l})$, where

$$\nu(n,\tilde{l}) = \frac{n(2mn + 2m - 4n^2 + 3n + 3)}{4n+2}.$$

Now note $\nu(n,\tilde{l}) \ge \mu(n,\tilde{h})$ if $n = 1$, and $\mu(n,\tilde{h}) \ge \nu(n,\tilde{l})$ if $n = 2, 3$.

We now prove the theorem for the case $n > 3$. Here we let

$$\tilde{h} = \frac{m-n+1}{2} \ge h$$

and

$$\tilde{l} = m - n + 1 - \frac{m-n}{2} \ge l.$$

1. In case $m \le 3n-7$ we have $\tilde{h} \le n-3$ and $\tilde{l} < n$; hence the second branch of μ and the first branch of ν have to be used. Thus,

$$\mu(n,h) \le \mu(n,\tilde{h}) = \frac{m-n+11}{8}(m-n+1),$$

$$\nu(n,l) \le \nu(n,\tilde{l}) = \frac{m-n}{8}(m-n+10)+1,$$

and $\mu(n,\tilde{h}) \ge \nu(n,\tilde{l})$.

2. In case $m = 3n-6$ we have $h = n-3$, $l = n-2$, and the theorem can be verified explicitely.

3. In case $3n-5 \le m \le 3n-2$ we have to use the third branch of μ and the first branch of ν. We obtain

$$\mu(n,h) \le \mu(n,\tilde{h}) = \frac{n}{2}(m-2n+6)-3$$

and

$$\nu(n,l) \le \nu(n,\tilde{l}) = \frac{m-n}{8}(m-n+10)+1.$$

For each $3n-5 \le m \le 3n-2$, $\mu(n,\tilde{h}) \ge \nu(n,\tilde{l})$.

4. In case $m = 3n-1$ we have $h = l = n$, and $\mu(n,h) > \nu(n,l)$ can be bounded as in the previous case.

5. In case $m > 3n-1$ we have $h \ge \frac{m-n}{2} > n - \frac{1}{2}$ and $l \ge m-n+1-\frac{m-n+1}{2} > n$. Using the third branch of μ and the second branch of ν we have

$$\mu(n,h) \le \mu(n,\tilde{h}) = \frac{n}{2}(m-2n+6)-3$$

and

$$\nu(n,l) \le \nu(n,\tilde{l}) = \frac{n}{2}(m-2n+2)+\frac{m}{2}.$$

Now, if $m \le 4n-6$, $\mu(n,\tilde{h}) \ge \nu(n,\tilde{l})$; if $m > 4n-6$, $\nu(n,\tilde{l}) > \mu(n,\tilde{h})$. □

Our method for exact division on two processors will be most useful on a shared-memory machine when invoked by an algebraic algorithm with a higher level of parallelism. When several exact divisions have to be executed in parallel, our method will add another level of parallelism to the program.

6 Experiments

We ran a sequential and a parallel implementation of our method on the shared-memory architecture of the *Sequent Symmetry*. We used the `PACLIB` environment [3] which combines the computer algebra library `SACLIB` [2] with the parallel features of the μSystem library [1].

Table 1 lists computing times and computing time ratios for inputs of various lengths. The row heading 20/15 refers to a dividend of 20 words and a divisor of 15 words. The column heading IQR stands for the `SACLIB` implementation of Knuth's integer quotient-remainder algorithm, Algorithm D; IEQ stands for the `SACLIB` implementation of Jebelean's integer exact quotient method; Sequential and Parallel refer to a sequential and a parallel implementation of our new method. The sequential implementation splits the quotient as in the proof of Theorem 1; the parallel implementation splits the quotient as in the proof of Theorem 2.

Table 2 has the same structure as Table 1, but instead of the computing time it lists the number of digit products that were computed. Those numbers agree very well with the bounds given in Theorems 1 and 2. The ratios of those numbers with respect to the number of digit products required in the classical algorithm are a measure for the expected speed-up.

The observed speed-up agrees well with the expected speed-up when the quotient is more than 30 words long. When the quotient is shorter, certain linear-time operations are significant. In particular, since `PACLIB` integers are represented as linked lists, we copy the inputs from lists to arrays and the output from an array to a list.

Surprisingly, the observed speed-up of IEQ and Sequential in the third section of Table 1 exceeds the expectations. This can be explained by noting that IQR and Algorithm H use digit divisions in order to determine the quotient digits. Table 2 counts those digit divisions as digit products, but the true cost of digit division is about 2.5 times the cost of a digit product in the `SACLIB` implementation we used. Hence the unexpected speed-up is due to the replacement of a

Table 1: Computing times in milliseconds (left) and speed-up ratios (*right*) with respect to the classical algorithm.

Lengths	IQR	IEQ		Sequential		Parallel	
20/15	4.7	1.5	*3.1*	1.5	*3.1*	1.5	*3.1*
40/30	16.1	3.7	*4.4*	3.4	*4.7*	2.4	*6.7*
60/45	34.4	7.2	*4.8*	5.8	*5.9*	3.7	*9.3*
100/75	91.2	17.5	*5.2*	12.3	*7.4*	7.4	*12.3*
150/112	201.5	37.6	*5.4*	23.7	*8.5*	13.1	*15.4*
200/150	351.5	62.6	*5.6*	37.7	*9.3*	20.6	*17.1*
20/10	6.0	3.7	*1.6*	3.0	*2.0*	2.3	*2.6*
40/20	20.6	14.4	*1.4*	8.1	*2.5*	4.7	*4.4*
60/30	44.7	24.2	*1.8*	14.9	*3.0*	8.6	*5.2*
100/50	119.3	61.5	*1.9*	35.0	*3.4*	20.0	*6.0*
150/75	262.5	133.3	*2.0*	72.1	*3.6*	40.6	*6.5*
200/100	462.3	233.4	*2.0*	122.3	*3.8*	66.8	*6.9*
20/5	5.0	4.2	*1.2*	4.2	*1.2*	2.9	*1.7*
40/10	16.4	13.8	*1.2*	12.6	*1.3*	7.2	*2.3*
60/15	34.8	28.8	*1.2*	25.1	*1.4*	14.5	*2.4*
100/25	91.4	75.0	*1.2*	62.8	*1.5*	35.1	*2.6*
150/37	198.1	162.8	*1.2*	133.0	*1.5*	72.8	*2.7*
200/50	351.0	286.9	*1.2*	229.6	*1.5*	123.9	*2.8*

Table 2: Count of digit products (left) and expected speed-up ratios (*right*) with respect to the classical algorithm.

Lengths	IQR	IEQ		Sequential		Parallel	
20/15	90	26	*3.5*	20	*4.5*	12	*7.5*
40/30	330	76	*4.3*	51	*6.5*	26	*12.7*
60/45	720	151	*4.8*	95	*7.6*	52	*13.8*
100/75	1950	376	*5.2*	220	*8.9*	117	*16.7*
150/112	4368	818	*5.3*	457	*9.6*	229	*19.1*
200/150	7650	1376	*5.6*	751	*10.2*	376	*20.3*
20/10	110	75	*1.5*	51	*2.2*	26	*4.2*
40/20	420	250	*1.7*	151	*2.8*	76	*5.5*
60/30	930	525	*1.8*	301	*3.1*	151	*6.2*
100/50	2550	1375	*1.9*	751	*3.4*	376	*6.8*
150/75	5700	3000	*1.9*	1595	*3.6*	817	*7.0*
200/100	10100	5250	*1.9*	2751	*3.7*	1376	*7.3*
20/5	80	85	*0.9*	70	*1.1*	37	*2.2*
40/10	310	295	*1.1*	245	*1.3*	130	*2.4*
60/15	690	630	*1.1*	520	*1.3*	267	*2.6*
100/25	1900	1675	*1.1*	1370	*1.4*	697	*2.7*
150/37	4218	3665	*1.2*	2992	*1.4*	1514	*2.8*
200/50	7550	6475	*1.2*	5245	*1.4*	2650	*2.8*

linear number of divisions by multiplications.

Finally we note that the parallel algorithm provides a significant speed-up even when the quotient is only 10 words long. In our experiments the efficiency of the parallel implementation exceeds 83% for quotients longer than 25 words and reaches 93% in some cases.

References

[1] P. A. Buhr, H. I. Macdonald, and R. A. Stroobosscher. *μSystem* Annotated Reference Manual. Version 4.4.1. Technical report, Department of Computer Science, University of Waterloo, Ontario, October 1991.

[2] George E. Collins et al. `SACLIB` user's guide. Technical Report 93-19, RISC-Linz 1993.

[3] H. Hong et al. `PACLIB` User Manual. Technical Report 92-32, RISC-Linz 1992.

[4] T. Jebelean. An algorithm for exact division. *Journal of Symbolic Computation*, 15(2):169–180, February 1993.

[5] T. Jebelean. Systolic algorithms for exact division. *Mitteilungen – Gesellschaft für Informatik e. V. Parallel-Algorithmen und Rechnerstrukturen*, Nr. 12, July 1993, pp. 40–50.

[6] D. E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms, 2nd edition. Addison-Wesley 1981.

[7] W. Krandick and J. R. Johnson. Efficient multiprecision floating point multiplication with exact rounding. Technical Report 93-76, RISC-Linz 1993.

[8] W. Krandick and J. R. Johnson. Efficient multiprecision floating point multiplication with optimal directional rounding. In E. Swartzlander, Jr., M. J. Irwin, and G. Jullien, editors, *Proceedings of the 11th IEEE Symposium on Computer Arithmetic, Windsor, Ontario, July 1993.* IEEE Computer Society Press 1993, pages 228–233.

[9] S. Lakshmivarahan and S. K. Dhall. *Analysis and design of parallel algorithms: Arithmetic and matrix problems.* McGraw-Hill, 1990.

[10] A. Schönhage and E. Vetter. A new approach to resultant computations and other algorithms with exact division. In Jan van Leeuwen, editor, *Proceedings of the 2nd Annual European Symposium on Algorithms, Utrecht, The Netherlands, September 1994*, Lecture Notes in Computer Science. Springer-Verlag, 1994. To appear.

[11] E. E. Swartzlander, editor. *Computer Arithmetic*, volume 1, part IV, and volume 2. IEEE Computer Society Press, 1990.

[12] C. T. Văcariu. Method and symmetrical architecture circuit for performing bidirectional exact division through step by step approximation, in various ways and formats (German). Patent Application 892/92, Österreichisches Patentamt, Vienna, 1992.