Short Division of Long Integers

David Harvey
Courant Institute of Mathematical Sciences
New York University
New York, USA
dmharvey@cims.nyu.edu

Paul Zimmermann
Centre de recherche Inria Nancy - Grand Est
Équipe-projet CARAMEL - bâtiment A
615 rue du jardin botanique
F-54600 Villers-lès-Nancy
Paul.Zimmermann@inria.fr

***Abstract*—We consider the problem of short division — i.e., approximate quotient — of multiple-precision integers. We present ready-to-implement algorithms that yield an approximation of the quotient, with tight and rigorous error bounds. We exhibit speedups of up to 30% with respect to GMP division with remainder, and up to 10% with respect to GMP short division, with room for further improvements. This work enables one to implement fast correctly rounded division routines in multiple-precision software tools.**

***Keywords*-floating-point division, arbitrary precision, GNU MP, GNU MPFR**

I. Introduction

The motivation for this work is to speed up the division routine in multiple-precision floating-point libraries like GNU MPFR [2], for numbers of say 200 to 20000 decimal digits. Floating-point division might reduce to integer division as follows. Assume for simplicity that one wants to divide a p-bit floating-point number a by another p-bit floating-point number b, with a result c of p bits. We also assume for simplicity that $a, b > 0$, and that the result c is rounded towards zero. We can write $a = m_a \cdot 2^{e_a}$ and $b = m_b \cdot 2^{e_b}$ with m_a, m_b positive integers, m_b having exactly p bits, $e_a, e_b \in \mathbb{Z}$, such that

$$2^{p-1} \le m_a/m_b < 2^p \qquad (1)$$

(then m_a has $2p$ or $2p-1$ bits). We then call an integer division routine — for example from the GNU MP library [3] — which returns the quotient $q \in \mathbb{N}$ and remainder $r \in \mathbb{N}$ such that $m_a = qm_b + r$, with $0 \le r < m_b$. It follows from Eq. (1) that $2^{p-1} \le q < 2^p$, thus q has exactly p bits, and the expected truncated quotient is $c = q \cdot 2^{e_a - e_b}$.

We assume that multiple-precision integers are written in base β, where $\beta \ge 2$ is an integer; in practice $\beta = 2^{64}$ on contemporary processors. We write such an integer as $a = \sum_{i=0}^{n-1} a_i \beta^i$, with $0 \le a_i < \beta$. By the quotient of two positive integers a and b we will mean their quotient rounded towards zero, i.e., $q = \lfloor a/b \rfloor$, sometimes denoted $a \operatorname{div} b$. We denote by $a \operatorname{mods} b$ the unique integer $r \equiv a \bmod b$ such that $-b/2 \le r < b/2$. We will use upper-case letters for integers consisting of several words in base β, and lower-case letters for individual words (or digits). We assume that we have available routines for the basic operations on multiple-precision integers, namely addition, subtraction, the full product Mul, and the division with remainder $\text{DivRem}(W, V)$, which returns Q and R such that $W = QV + R$ with $0 \le R < V$. We also write $\text{Div}(W, V)$ for a routine returning only Q. From those basic routines we will design several algorithms returning an *approximation* to the quotient Q. We never assume that the divisor V is invariant, thus we avoid precomputations involving it.

Several techniques are available to compute an approximate quotient. The most straightforward is Mulders' short division, which is itself based on Mulders' short product [8]. Mulders only considered the polynomial case; one contribution of this article is a detailed algorithm in the integer case with a tight and rigorous error analysis (§II-B). The other technique is based on Barrett's division [1], which first computes an approximation of the divisor inverse, and then multiplies it by the dividend. A folding technique enables one to decrease the inversion cost; this method is described in [5] in the context of modular reduction. We give a detailed algorithm using this folding technique, together with a tight and rigorous error analysis (§II-D). Finally we present some experimental results comparing the two approaches, together with plain division (§III).

II. Our Contribution

In this section we describe two algorithms computing an approximate quotient: Algorithm ShortDiv implementing Mulders' short division (§II-B) and Algorithm FoldDiv implementing Barrett division with the folding technique from [5] (§II-D). Both algorithms use as a subroutine Mulders' short product, which we describe in detail and analyze first (§II-A).

A. Mulders' Short Product

We describe here our implementation of an integer version of Mulders' short product [8], and analyze precisely its error. We give two variants: ShortMulNaive is a quadratic-time algorithm, and ShortMul is a subquadratic algorithm, assuming that the full product routine Mul implements subquadratic algorithms (such as Karatsuba, Toom-Cook, FFT).

Algorithm II.1 ShortMulNaive

```
Input: $U = \sum_{i=0}^{n-1} u_i \beta^i$, $V = \sum_{i=0}^{n-1} v_i \beta^i$, integer $n \geq 1$
Output: an integer $W$ with $UV\beta^{-n} - n < W \leq UV\beta^{-n}$
1: $W \leftarrow u_{n-1} v_0$
2: for $i$ from 1 to $n-1$ do
3:     $W \leftarrow W + (u_{n-1}\beta^i + \cdots + u_{n-1-i}) \cdot v_i$
4: $W \leftarrow \lfloor W\beta^{-1} \rfloor$
```

The error analysis of Algorithm ShortMulNaive is as follows. For each $0 \leq i < n-1$ we neglect the products $u_j \beta^j v_i \beta^i$ for $i + j < n - 1$, whose total contribution is less than β^n for each i. Thus the total error for all i is less than $(n-1)\beta^n$ with respect to the full product UV, and n with respect to $UV\beta^{-n}$, taking into account the truncation of step 4. In addition, all the neglected terms are nonnegative, which proves $W \leq UV\beta^{-n}$.

Note that the lower bound $UV\beta^{-n} - n$ is essentially optimal, as shown by the following example for $\beta = 1000$ and $n = 2$: take $U = 780996$, $V = 308999$, we get $W = 241325$, while $UV\beta^{-n} \approx 241326.983$.

Algorithm II.2 ShortMul (Mulders)

```
Input: $U = \sum_{i=0}^{n-1} u_i \beta^i$, $V = \sum_{i=0}^{n-1} v_i \beta^i$, integer $n \geq 1$
Output: an integer $W$ with $UV\beta^{-n} - n < W \leq UV\beta^{-n}$
1: if $n < n_0$ then                                ▷ We assume $n_0 \geq 5$
2:     $W \leftarrow \mathrm{ShortMulNaive}(U, V, n)$
3: else
4:     choose an integer $k$, $(n+3)/2 \leq k < n$, $\ell \leftarrow n - k$
5:     write $U = U_1\beta^\ell + U_0$, $V = V_1\beta^\ell + V_0$
6:     write $U = U_1'\beta^k + U_0'$, $V = V_1'\beta^k + V_0'$
7:     $W_{11} \leftarrow \mathrm{Mul}(U_1, V_1, k)$                      ▷ $2k$ words
8:     $W_{10} \leftarrow \mathrm{ShortMul}(U_1', V_0, \ell)$          ▷ $\ell$ upper words
9:     $W_{01} \leftarrow \mathrm{ShortMul}(U_0, V_1', \ell)$          ▷ $\ell$ upper words
10:    $W \leftarrow \lfloor W_{11}\beta^{2\ell-n} \rfloor + W_{10} + W_{01}$
```

For large n, Algorithm ShortMul is faster than ShortMulNaive because ShortMulNaive always performs about $n^2/2$ word products, whereas ShortMul benefits in step 7 from a subquadratic implementation of the full product U_1V_1, as provided for example by the GNU MP library.

Lemma 1: The output of Algorithm ShortMul satisfies

$$UV\beta^{-n} - n < W \leq UV\beta^{-n}.$$

Proof: For $n < n_0$, the error analysis is the same as that of Algorithm ShortMulNaive. Assume $n \geq n_0$. The parts neglected in the full product UV are the products $U_0'V_0$ and U_0V_0' (which overlap on U_0V_0) and the neglected parts from the two recursive calls to ShortMul, which return approximations of $U_1'V_0\beta^{-\ell}$ and $U_0V_1'\beta^{-\ell}$ respectively. Since $U_0'V_0 < \beta^k\beta^\ell = \beta^n$ and similarly $U_0V_0' < \beta^n$, and since by induction each recursive call yields an error of at most ℓ units

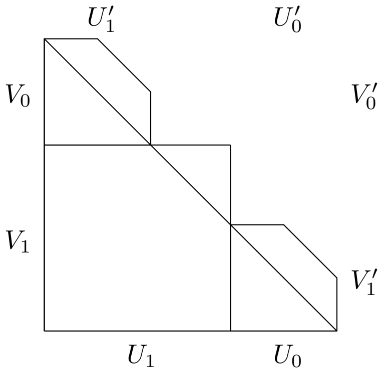

Figure 1. A graphical view of Algorithm ShortMul, with most significant parts bottom left. The cut squares represent (recursive) short products.

in last place, the total error is less than $2\ell + 2 + 1$, where the term 2 stands for the neglected terms $U_0'V_0$ and U_0V_0', and the term 1 stands for the truncation error in $\lfloor W_{11}\beta^{2\ell-n} \rfloor$. This is bounded by $2\ell + 3 \leq n$, since $\ell = n - k \leq (n-3)/2$. A "graphical" proof is as follows: we see on Fig. 1 that the neglected products u_iv_j are above the diagonal, thus correspond to $i + j < n - 1$; as a consequence, the value computed by ShortMul is greater or equal to that computed by ShortMulNaive. ■

NOTE: Algorithm ShortMul takes two inputs less than β^n. We will use it in some cases with $\beta^n \leq U < 2\beta^n$, with the convention that

$$\mathrm{ShortMul}(U, V, n) := V + \mathrm{ShortMul}(U - \beta^n, V, n).$$

Lemma 1 still applies since the V term is computed exactly.

B. Mulders' Short Division

The following lemma gives a bound on the difference between the true quotient $Q = \lfloor W/V \rfloor$ and the approximate quotient obtained by discarding the lower order words of W and V. It is a generalization of a classical result, see for example [7, Theorem 4.3.1.B].

Lemma 2: Let W, V be two positive integers, and $Q = \lfloor W/V \rfloor$ their quotient. Consider an integer B, $0 < B \leq V$, and define $Q_1 = \lfloor (W \text{ div } B)/(V \text{ div } B) \rfloor$. Then

$$Q \leq Q_1.$$

If in addition $Q < \lambda(V \text{ div } B)$, with λ a positive integer, then

$$Q_1 \leq Q + \lambda.$$

Proof: Let $W = W_1B + W_0$ and $V = V_1B + V_0$ with $0 \leq W_0, V_0 < B$. We have

$$Q \leq \frac{W}{V} < Q + 1 \qquad (2)$$

and

$$Q_1 \leq \frac{W_1}{V_1} < Q_1 + 1.$$

Expanding the left inequality of (2) leads to $(W_1 - QV_1)B \geq QV_0 - W_0 \geq -W_0$. Since the left hand side is a multiple of B and $-W_0 > -B$, we conclude $W_1 - QV_1 \geq 0$, thus $W_1/V_1 \geq Q$ and $Q_1 \geq Q$.

Expanding the right inequality of (2) leads to $W_1 B \leq W_1 B + W_0 < (Q+1)(V_1 B + V_0) < (Q+1)V_1 B + (Q+1)B$. Dividing by B leads to $W_1 < (Q+1)V_1 + Q + 1$, thus $W_1 \leq (Q+1)V_1 + Q < (Q+1)V_1 + \lambda V_1$, and $W_1/V_1 < Q + \lambda + 1$. By taking the floor on both sides, it follows $Q_1 = \lfloor W_1/V_1 \rfloor \leq Q + \lambda$ since λ is an integer. ■

In [7] this lemma is used with W a number consisting of $n+1$ words in base β, V consisting of n words with its most significant word larger than $\beta/2$, and $B = \beta^{n-1}$, thus Q_1 is the result of dividing the two most significant words of W by the most significant word of V; in that case, if we know in advance that the quotient Q is less than β — for example within a multiple-precision division — then $\lambda = 2$ applies, and the lemma yields $Q \leq Q_1 \leq Q + 2$.

To describe short division algorithms, we consider a division of $2n$ words by n words. In the application to floating-point arithmetic, we are mainly interested in division of n words by n words. However, retaining a dividend of $2n$ words has several advantages:

- it makes it easier to call division with remainder in the recursive calls (if any);
- it is interesting to have one more significant word of the dividend, to get more accurate partial quotients. We use this especially in Algorithm FoldDiv (§II-D);
- for floating-point arithmetic, to get correct rounding, in some rare cases one needs to consider the remainder, i.e., the low limbs of the initial dividend.

Algorithm II.3 ShortDiv (Mulders)

```
Input: $W = \sum_{i=0}^{2n-1} w_i \beta^i$, $V = \sum_{i=0}^{n-1} v_i \beta^i$
Require: $v_{n-1} \geq \beta/2$, $n \geq 1$
Output: an integer approximation $U$ of $Q = \lfloor W/V \rfloor$
 1: if $n < n_1$ then                                  ▷ We assume $n_1 \geq 5$
 2:     $U \leftarrow \mathrm{Div}(W, V)$                 ▷ Returns $\lfloor W/V \rfloor$
 3: else
 4:     choose an integer $k$, $(n+3)/2 \leq k < n$, $\ell \leftarrow n - k$
 5:     write $W = W_1 \beta^{2\ell} + W_0$, $V = V_1 \beta^\ell + V_0$
 6:     write $V = V_1' \beta^k + V_0'$
 7:     $(U_1, R_1) \leftarrow \mathrm{DivRem}(W_1, V_1)$
 8:     write $U_1 = U_1' \beta^{k-\ell} + S$ with $0 \leq S < \beta^{k-\ell}$
 9:     $T \leftarrow \mathrm{ShortMul}(U_1', V_0, \ell)$
10:     $W_{01} \leftarrow R_1 \beta^\ell + (W_0 \text{ div } \beta^\ell) - T\beta^k$
11:     while $W_{01} < 0$ do
12:         $(U_1, W_{01}) \leftarrow (U_1 - 1, W_{01} + V)$
13:     $U_0 \leftarrow \mathrm{ShortDiv}(W_{01} \text{ div } \beta^{k-\ell}, V_1', \ell)$
14:     return $U_1 \beta^\ell + U_0$
```

Theorem 1: Algorithm ShortDiv returns an approximation U of $Q = \lfloor W/V \rfloor$, with $Q \leq U \leq Q + 2n$.

Proof: After step 7, we have $W = (U_1 V_1 + R_1)\beta^{2\ell} + W_0$, where $U_1 < 2\beta^k$, $0 \leq R_1 < V_1 < \beta^k$, and $0 \leq W_0 < \beta^{2\ell}$. Thus $W = U_1 V \beta^\ell + R_1 \beta^{2\ell} - U_1 V_0 \beta^\ell + W_0$. After step 8, we have $W = U_1 V \beta^\ell + R_1 \beta^{2\ell} - U_1' V_0 \beta^k - S V_0 \beta^\ell + W_0$, where $U_1' < 2\beta^\ell$. After step 9, we have $U_1' V_0 = T\beta^\ell + T_0$, where $0 \leq T_0 < \ell \beta^\ell \leq (n-3)/2 \cdot \beta^\ell$ by Lemma 1 and $T < 2\beta^\ell$, thus $W = U_1 V \beta^\ell + R_1 \beta^{2\ell} - T\beta^n - T_0 \beta^k - S V_0 \beta^\ell + W_0$. After step 10, we have

$$W = U_1 V \beta^\ell + W_{01} \beta^\ell + W_{00}, \qquad (3)$$

where $W_{00} = -T_0 \beta^k - S V_0 \beta^\ell + (W_0 \bmod \beta^\ell)$. Thus

$$-\frac{n-3}{2}\beta^n - (\beta^{k-\ell} - 1)(\beta^\ell - 1)\beta^\ell < W_{00} < \beta^\ell,$$

and so $-\frac{n-1}{2}\beta^n < W_{00} < \beta^\ell$.

We have $W_{01} \geq -T\beta^k \geq -U_1' V_0 \beta^{k-\ell} \geq -2\beta^n$, so the adjustment in step 12 executes at most four times. Note that Eq. (3) continues to hold after step 12, as $U_1 V + W_{01}$ is left unchanged. Furthermore after step 12 we have: (i) in case $W_{01} \geq 0$ in step 10, $W_{01} \leq R_1 \beta^\ell + (W_0 \text{ div } \beta^\ell) \leq (V_1 - 1)\beta^\ell + \beta^\ell - 1 < V < \beta^n$, (ii) otherwise $W_{01} < V < \beta^n$, thus in both cases $W_{01} \text{ div } \beta^{k-\ell} < \beta^{2\ell}$, permitting the recursive call to ShortDiv.

Eq. (3) yields

$$\left\lfloor \frac{W}{V} \right\rfloor \leq \left\lfloor U_1 \beta^\ell + \frac{W_{01}\beta^\ell}{V} + \frac{W_0 \bmod \beta^\ell}{V} \right\rfloor = U_1 \beta^\ell + \left\lfloor \frac{W_{01}'}{V} \right\rfloor,$$

where $W_{01}' = W_{01}\beta^\ell + (W_0 \bmod \beta^\ell)$. Applying Lemma 2 on $W := W_{01}'$, $V := V$ and $B := \beta^k$ yields:

$$\left\lfloor \frac{W}{V} \right\rfloor \leq U_1 \beta^\ell + \left\lfloor \frac{W_{01}' \text{ div } \beta^k}{V \text{ div } \beta^k} \right\rfloor.$$

Since $W_{01}' \text{ div } \beta^k = W_{01} \text{ div } \beta^{k-\ell}$ and $V \text{ div } \beta^k = V_1'$, this proves by induction that the quotient returned by Algorithm ShortDiv is always larger or equal to the true quotient Q.

For the lower bound we have still from Eq. (3):

$$\begin{aligned} \frac{W}{V} &\geq U_1 \beta^\ell + \left\lfloor \frac{W_{01}\beta^\ell}{V} \right\rfloor + \frac{W_{00}}{V} \\ &\geq U_1 \beta^\ell + \left\lfloor \frac{W_{01} \text{ div } \beta^{k-\ell}}{V_1'} \right\rfloor - 2 - (n-1), \end{aligned}$$

using again[1] Lemma 2 and $W_{00} > -\frac{n-1}{2}\beta^n$. Thus $Q = \lfloor W/V \rfloor \geq U_1 \beta^\ell + \lfloor (W_{01} \text{ div } \beta^{k-\ell})/V_1' \rfloor - (n+1)$. Let $\varepsilon(n)$ be the maximal over-estimation of the quotient in Algorithm ShortDiv, i.e., $Q \leq U \leq Q + \varepsilon(n)$, then we have $\varepsilon(n) \leq \varepsilon(\ell) + n + 1$, thus since $\ell \leq (n-3)/2$, $\varepsilon(n) \leq 2n$. ■

REMARK: If β is even, the normalization condition $v_{n-1} \geq \beta/2$ is equivalent to $V \geq \beta^n/2$. In general the former

[1] We use here Lemma 2 with $W := W_{01}\beta^\ell$, $V := V$ and $B := \beta^k$. We have seen that $W_{01} < V$, thus with the notations of Lemma 2: $Q = \lfloor W_{01}\beta^\ell / V \rfloor \leq \lfloor (V-1)\beta^\ell / V \rfloor \leq \beta^\ell - 1 < 2(V \text{ div } B)$, thus $\lambda = 2$ applies.

condition is stronger; for example if $\beta = 3$, $n = 2$, $V = 5$, then $V \geq \beta^n/2$ but $v_1 = 1 < \beta/2$. The former condition is more natural, since if V is normalized, then any truncation $V \div \beta^k$ is normalized too, which is what we need in our algorithms.

REMARK: The $2n$ error bound in Theorem 1 can be improved slightly by more careful analysis of the recursion for $\varepsilon(n)$, taking into account that $\varepsilon(n) = 0$ for $n < n_1$. On the other hand, experiments suggest that the bound is asymptotically sharp, in the sense that the ratio of the error to $2n$ can be made arbitrarily close to 1 by taking n sufficiently large and by choosing W and V carefully. We have not attempted to prove this, but we give some examples. Let $\beta = 2^8$, $n_1 = 5$. We assume $k = \lceil (n+3)/2 \rceil$ at each recursion level, and that all ShortMul calls use ShortMulNaive. Then for $n = 61$, if V is given in hexadecimal by

```
80f7bc6ffe0000007fffffffff0000007fffffffffff
ffffffffffffff0000007fffffffffffffffffffffff
ffffffffffffffffffffffffffffffffff
```

and $W = \beta^{56} W'$ where W' is given by

```
407bde37ff0000003fffffffff8000003fffffffffff
fffffffffffffeff0844107dde37fe8000004000ffff
fe7f0844107dde37fdff0843d60e74d271d0e35f6bd4,
```

then the error is $93 \approx 1.52n$. We also constructed an example with $n = 1021$ where the error is $1965 \approx 1.92n$. The method of construction is to find values of U_1', V_0, W_{01} div $\beta^{k-\ell}$ and V_1' that elicit large errors in the recursive ShortMul and ShortDiv calls, and then reverse-engineer inputs that generate these calls at the next recursion level.

The theoretical running time analysis of Algorithm ShortDiv is not easy, since as visible on Fig. 2, the optimal cutoff value k depends heavily on n, and there is no simple formula giving k from n.

C. The Integer Middle Product

Before describing our algorithm using Barrett division with folding, we recall the definition of the integer middle product from [4]. Given two multiple-precision integers $X = \sum_{i=0}^{m-1} x_i \beta^i$ and $Y = \sum_{j=0}^{n-1} y_j \beta^j$ with $m \geq n$, their middle product is defined to be

$$\mathrm{MP}_{m,n}(X, Y) = \sum_{\substack{0 \leq i < m,\, 0 \leq j < n \\ n-1 \leq i+j \leq m-1}} x_i y_j \beta^{i+j-n+1}. \qquad (4)$$

The middle product $\mathrm{MP}_{m,n}(X, Y)$ takes into account the $n(m - n + 1)$ word-products $x_i y_j$ from weight β^{n-1} to weight β^{m-1}. In the "balanced" case $m = 2n - 1$, it corresponds to n^2 word-products. Assuming $n < \beta$, which holds for any reasonable value of n with $\beta = 2^{64}$, $\mathrm{MP}_{m,n}(X, Y) < \beta^{m-n+3}$ and thus fits into $m - n + 3$ words.

We will use the following lemma on the middle product:

Lemma 3: Let $m \geq n$, and suppose that $0 \leq X < \beta^m$ and $0 \leq Y < \beta^n$. Then

$$|\left(XY - \beta^{n-1}\mathrm{MP}_{m,n}(X, Y)\right) \bmod \beta^m| < (n - 1)\beta^n.$$

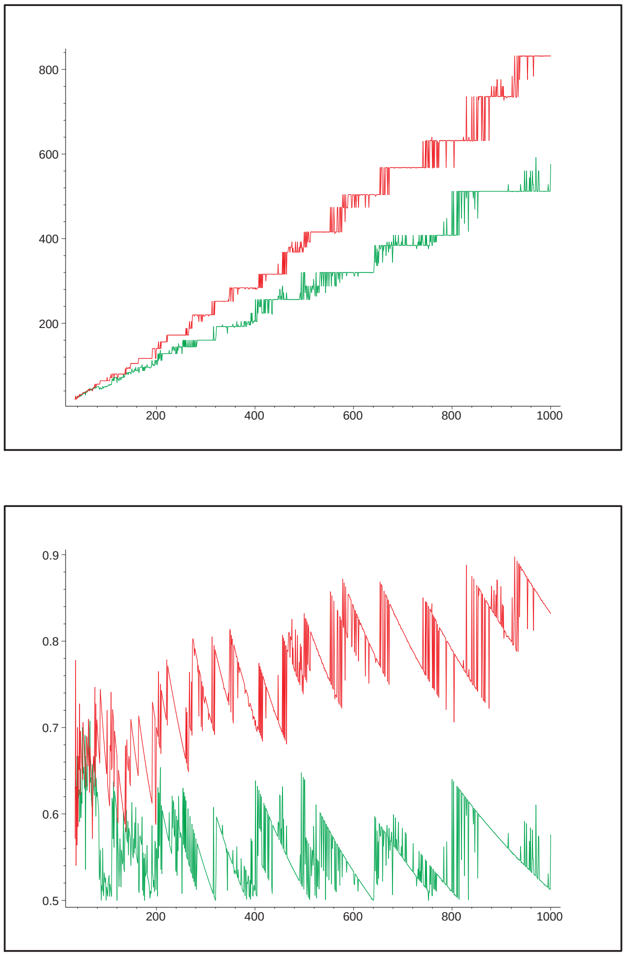

Figure 2. Up: optimal cutoff value k determined by tuning for ShortMul (top, red) and ShortDiv (bottom, green) for word size n; down: ratio k/n (experimental conditions described in §III).

Proof: From the definition (4) of $\mathrm{MP}_{m,n}(X, Y)$, we see that

$$\beta^{n-1}\mathrm{MP}_{m,n}(X, Y) = \sum_{\substack{0 \leq i < m,\, 0 \leq j < n \\ n-1 \leq i+j \leq m-1}} x_i y_j \beta^{i+j},$$

i.e., $\beta^{n-1}\mathrm{MP}_{m,n}(X, Y)$ takes into account exactly the word-products of weight $n - 1$ to $m - 1$ of the full product XY. The word-products of weight m or more are zero modulo β^m, therefore

$$\begin{aligned} &\left(XY - \beta^{n-1}\mathrm{MP}_{m,n}(X, Y)\right) \bmod \beta^m \\ &\quad = \sum_{\substack{0 \leq i < m,\, 0 \leq j < n \\ i+j < n-1}} x_i y_j \beta^{i+j} = \sum_{j=0}^{n-2} \sum_{\substack{0 \leq i < m \\ i+j < n-1}} x_i y_j \beta^{i+j}. \end{aligned}$$

For each j, $0 \leq j \leq n-2$, the sum over i is bounded strictly by β^n, which proves the Lemma. ■

D. ℓ-fold Barrett Division

Barrett division consists in first computing an approximation of $1/V$, then multiplying it by W to obtain an approximation of the quotient Q [1]. However, since computing n words of $1/V$ is expensive, a natural idea is to approximate n/ℓ upper words of $1/V$ for an integer $\ell \geq 2$, and to use this high inverse ℓ times. This folding technique is described in [5] in the context of modular reduction; for $\ell = 2$, it reduces to Karp-Markstein division [6]. The best known constant for power series division in the FFT domain, due to van der Hoeven [9], depends on the same sort of folding idea, with careful attention paid to FFT reuse. In this section, we give a precise algorithm based on ℓ-fold Barrett division, and analyze its error. We consider an input-independent algorithm which returns an approximation with error $O(n)$ ulps; a different approach would be to ask for an error of at most one ulp, and increase if necessary the precision inside the algorithm. (Since usually $n < \beta$, using Algorithm FoldDiv with $n + 1$ words, we obtain an error smaller than one ulp, moreover in most cases we can obtain a correct rounding.)

Algorithm II.4 FoldDiv(ℓ), $\ell \geq 2$

```
Input: $W = \sum_{i=0}^{2n-1} w_i\beta^i$, $V = \sum_{i=0}^{n-1} v_i\beta^i$
Require: $v_{n-1} \geq \beta/2$, $W < \beta^n V$
Output: an integer approximation $U$ of $Q = \lfloor W/V \rfloor$
 1: if $n < n_2$ then
 2:     return $U \leftarrow \mathrm{Div}(W, V)$
 3: $k \leftarrow \lceil n/\ell \rceil$
 4: write $V = V_1\beta^{n-(k+1)} + V_0$        ▷ $V_1$ has $k+1$ words
 5: $I \leftarrow \lfloor (\beta^{2(k+1)} - 1)/V_1 \rfloor$
 6: $r \leftarrow n$, $W_r \leftarrow W$, $U_n \leftarrow 0$
 7: while $r > k+1$ do        ▷ invariant: $0 \leq W_r < \beta^r V$
 8:     $Q_r \leftarrow \mathrm{ShortMul}(W_r \text{ div } \beta^{n+r-(k+1)}, I, k+1)$
 9:     $Q_r \leftarrow \min(Q_r, \beta^{k+1} - 1)$
10:     $T_r \leftarrow \mathrm{MP}_{r+1,k+1}(V \text{ div } \beta^{n-r}, Q_r)$
11:     $W_{r-k} \leftarrow (W_r - T_r\beta^{n-1}) \text{ mods } \beta^{n+r-k}$
12:     $U_{r-k} \leftarrow U_r + Q_r\beta^{r-(k+1)}$
13:     if $W_{r-k} < 0$ then
14:         $W_{r-k} \leftarrow W_{r-k} + \beta^{r-k}V$
15:         $U_{r-k} \leftarrow U_{r-k} - \beta^{r-k}$
16:     $r \leftarrow r - k$
17: $Q_r \leftarrow \mathrm{ShortMul}(W_r \text{ div } \beta^{n+r-(k+1)}, I, k+1)$
18: $U \leftarrow U_r + (Q_r \text{ div } \beta^{k+1-r})$
```

The idea of Algorithm FoldDiv is the following: at each step we have $W \approx U_rV + W_r$, we guess a partial quotient Q_r of W_r by $V\beta^{r-(k+1)}$, and update W_{r-k} and U_r accordingly. The crucial point is to check that $W_{r-k} \approx W_r - Q_rV\beta^{r-(k+1)}$, up to low order terms. The final steps are special since we don't have to estimate the new remainder.

Theorem 2: Assuming $n + 9 < \beta/2$ and $\ell \leq \sqrt{n/2}$, Algorithm FoldDiv(ℓ) returns an approximation U of $Q = \lfloor W/V \rfloor$, with error less than $2n$.

Proof: We will first prove that $|I - \beta^{n+k+1}/V| \leq 5$. Let $\varepsilon = \beta^{-(k+1)}$. We have $V_1 = V\beta^{-n+(k+1)}(1 - \delta_1)$ with $0 \leq \delta_1 = V_0/V < 2\varepsilon$. Now since $\beta^{2(k+1)} - 1 = IV_1 + R_1$ with $0 \leq R_1 < V_1$, this yields $\beta^{2(k+1)} = IV_1 + R_1'$ with $0 \leq R_1' \leq V_1$, thus $\beta^{2(k+1)}/V_1 = I + R_1'/V_1$ with $0 \leq R_1'/V_1 \leq 1$. Since $V_1 < \beta^{k+1}$, $I \geq \beta^{k+1}$, thus we can write $\beta^{2(k+1)}/V_1 = I(1 + \delta_2)$ with $0 \leq \delta_2 \leq \varepsilon$. We thus have $I = \beta^{2(k+1)}/(V_1(1 + \delta_2)) = \beta^{n+k+1}/(V(1 - \delta_1)(1 + \delta_2))$. Note that the errors due to δ_1 and δ_2 are of opposite signs, thus we can bound by the larger absolute error, which is due to δ_1. It can be shown that $|I - \beta^{n+k+1}/V| \leq 5$ as long as $\beta \geq 4$, which follows from our assumption.

We will prove the following by induction: $0 \leq W_r < \beta^r V$ at line 7, and $|W - (U_rV + W_r)| \leq (k+1)(n-r)/k \cdot \beta^n$.

This holds for $r = n$, since $W_n = W < \beta^n V$ by hypothesis, and $U_n = 0$, thus $|W - (U_nV + W_n)| = 0$.

Now assume the induction hypothesis holds for some $r > k+1$; we will prove it still holds for $r - k$. We will do this in three steps:

1) we first show that Q_r is close to $W_r\beta^{(k+1)-r}/V$;
2) we deduce that $\widehat{W}_{r-k} := W_r - Q_rV\beta^{r-(k+1)}$ is small;
3) finally we show that the value W_{r-k} computed at step 11 is close to $\widehat{W}_{r-k}$.

The difference $|(W_r \text{ div } \beta^{n+r-(k+1)})I - W_r\beta^{2(k+1)-r}/V|$ is bounded by

$$\begin{aligned}
|(W_r \quad \text{div} \quad & \beta^{n+r-(k+1)})I - W_r\beta^{-n-r+(k+1)}I| \\
+ \quad & |W_r\beta^{-n-r+(k+1)}I - W_r\beta^{2(k+1)-r}/V| \\
\leq \quad & I + W_r\beta^{-n-r+(k+1)}|I - \beta^{n+k+1}/V| \\
\leq \quad & 2\beta^{k+1} + 5\beta^{k+1} \leq 7\beta^{k+1}.
\end{aligned}$$

The short product at line 8 induces an error of at most $k+1$ ulps, thus

$$|Q_r - W_r\beta^{(k+1)-r}/V| \leq k + 8. \quad (5)$$

(This holds for the value of Q_r in line 8, and is still correct after line 9, because if $Q_r \geq \beta^{k+1}$ on line 8, since $W_r < \beta^r V$, we have $W_r\beta^{(k+1)-r}/V < \beta^{k+1}$, and thus the new value $Q_r = \beta^{k+1} - 1$ is closer to $W_r\beta^{(k+1)-r}/V$.) It follows:

$$|\widehat{W}_{r-k}| \leq (k+8)V\beta^{r-(k+1)} \leq (k+8)\beta^{n+r-(k+1)}. \quad (6)$$

Let $V_r = V \text{ div } \beta^{n-r}$, considered as having $r + 1$ words (with zero most significant word). At line 10, T_r is an approximation of $Q_rV\beta^{r-n-k}$, thus $T_r\beta^{n-1}$ at step 11 approximates $Q_rV\beta^{r-(k+1)}$, with the following possible differences:

- a multiple of β^{n+r-k} from the symmetric modulo at step 11;
- the truncation of V in step 10;

- the high-order error on T_r from the middle product at step 10, which is a multiple of β^{r-k+1} from Lemma 3, and thus a multiple of β^{n+r-k} for $\beta^{n-1}T_r$ at step 11;
- the low-order error from the middle product at step 10.

More precisely:

$$|T_r - Q_r V \beta^{r-n-k}| \bmod \beta^{r-k+1} < Q_r\beta^{-k} + k\beta \leq (k+1)\beta,$$

where $Q_r\beta^{-k}$ takes into account the truncation of V into V_r, and $k\beta$ comes from Lemma 3. Thus $|\beta^{n-1}T_r - Q_rV\beta^{r-(k+1)}| \bmod \beta^{n+r-k} < (k+1)\beta^n$. It follows that $W_{r-k} = \widehat{W}_{r-k} + \alpha\beta^{n+r-k} + \varepsilon$, where $|\varepsilon| < (k+1)\beta^n$, and α is an integer. Since we know that $\widehat{W}_{r-k}$ is small from Eq. (6), it follows: $W_{r-k} = \alpha\beta^{n+r-k} + \varepsilon'$ where

$$|\varepsilon'| < (k+1)\beta^n + (k+8)\beta^{n+r-(k+1)} < \frac{1}{2}\beta^{n+r-k},$$

since $(k+1)\beta^n \leq (n+1)\beta^{n+r-(k+2)} < \beta^{n+r-(k+1)}$, and $(k+9)\beta^{n+r-(k+1)} \leq (n+9)\beta^{n+r-(k+1)} < \frac{1}{2}\beta^{n+r-k}$. Since the symmetric modulus at step 11 guarantees that $|W_{r-k}| \leq \frac{1}{2}\beta^{n+r-k}$, necessarily $\alpha = 0$, which proves that before step 13:

$$W_{r-k} = \widehat{W}_{r-k} + \varepsilon,$$

with $|\varepsilon| < (k+1)\beta^n$.

We are now ready to prove the induction hypothesis. If $W_{r-k} \geq 0$ after step 11, then $W_{r-k} < \frac{1}{2}\beta^{n+r-k} \leq \beta^{r-k}V$, which proves the first part of the induction hypothesis. If $W_{r-k} < 0$ after step 11, then $-\frac{1}{2}\beta^{n+r-k} \leq W_{r-k} < 0$, and after step 14, $0 \leq W_{r-k} < \beta^{r-k}V$, which also proves the first part of the induction hypothesis.

For the second part, we know by induction that $|W - (U_rV + W_r)| \leq (k+1)(n-r)/k \cdot \beta^n$. For $U_{r-k} = U_r + Q_r\beta^{r-(k+1)}$, and the new value of W_{r-k}, we have:

$$\begin{aligned}
|W \quad &- \quad [U_{r-k}V + W_{r-k}]| \leq |W - (U_rV + W_r)| \\
&+ \quad |(U_rV + W_r) - [(U_r + Q_r\beta^{r-(k+1)})V + W_{r-k}]| \\
&\leq \quad (k+1)(n-r)/k \cdot \beta^n + |\widehat{W}_{r-k} - W_{r-k}| \\
&\leq \quad (k+1)(n-r)/k \cdot \beta^n + (k+1)\beta^n \\
&\leq \quad (k+1)(n-(r-k))/k \cdot \beta^n,
\end{aligned}$$

which proves the second part of the induction hypothesis.

For the last steps, the analysis leading to Eq. (5) still holds, thus if U_r denotes the previous approximate quotient, and $U = U_r + (Q_r \operatorname{div} \beta^{k+1-r})$ is the final result:

$$\begin{aligned}
|W \quad &- \quad UV| \\
&\leq \quad |W - (U_rV + W_r)| + |(U_rV + W_r) - UV| \\
&\leq \quad (k+1)\frac{n-r}{k}\beta^n + |W_r - (Q_r \operatorname{div} \beta^{k+1-r})V| \\
&\leq \quad (k+1)\frac{n-r}{k}\beta^n + |W_r - Q_r\beta^{r-(k+1)}V| \\
&+ \quad |Q_r\beta^{r-(k+1)} - (Q_r \operatorname{div} \beta^{k+1-r})|V.
\end{aligned}$$

Dividing by V and using $V \geq \beta^n/2$ we get:

$$\begin{aligned}
|W/V \quad &- \quad U| \\
&\leq \quad 2(k+1)\frac{n-r}{k} + |W_r/V - Q_r\beta^{r-(k+1)}| + 1 \\
&\leq \quad 2(k+1)\frac{n-r}{k} + (k+8)\beta^{r-(k+1)} + 1.
\end{aligned}$$

Since $k = \lceil n/\ell \rceil$, we can write $n = k\ell - s$ with $0 \leq s < \ell$. We assumed that $\ell \leq \sqrt{n/2}$, then $k \geq n/\ell \geq \sqrt{2n} \geq 2\ell$. Thus $0 \leq s < k/2$. Since r equals n initially and decreases by k at each step, we have $r \equiv n \bmod k$ at the end, with $2 \leq r \leq k+1$. On the other hand we have $k/2 < k - s \leq k$, which proves that $r = k - s$ at the end ($\ell \leq \sqrt{n/2}$ implies $n \geq 8$, which in turn implies $k \geq 4$). Thus the case $r = k+1$ at the end is not possible. Therefore the term $(k+8)\beta^{r-(k+1)}$ is bounded by $(k+8)\beta^{-1} \leq 1$. We thus get since $s \leq (k-1)/2$:

$$\begin{aligned}
|W/V \quad &- \quad U| \leq 2(k+1)(n-k+s)/k + 2 \\
&= \quad 2(n-k+s) + 2(n-k+s)/k + 2 \\
&\leq \quad 2n - 2k + k - 1 + 2n/k - 2 + 2s/k + 2 \\
&\leq \quad 2n - k + 2\ell + 2s/k - 1 \\
&< \quad 2n - k + k + 1 - 1 = 2n.
\end{aligned}$$

■

Remark 1: In Algorithm FoldDiv, the variables W_r with $r = n,\ n-k,\ n-2k, \ldots$ might of course overwrite each other, similarly for T_r and U_r. We used indices to make the proof simpler.

Remark 2: Algorithm FoldDiv also works with $\ell = 1$, with the convention that line 4 yields $V_1 = \beta V$ and $V_0 = 0$; in that case $k = n+1$, we compute an approximate inverse on $n+1$ words on line 5, we skip the while-loop, and perform a short product between the $n+1$ most significant words of W_r and I at line 17, which is finally truncated by one word at line 18. This is exactly Barrett's algorithm, with a working precision of $n+1$ words.

1) Theoretical running time analysis of FoldDiv: Neglecting operations with linear cost, the cost of FoldDiv(ℓ) is mainly that of the initial inversion of size $n' = n/\ell$, followed by $\ell - 1$ loops, each one performing a short product of size n' and a middle product, and a final short product of size n'. In the ith iteration, $1 \leq i \leq \ell - 1$, the middle product can be computed with $\ell - i$ balanced middle products with parameters $2n' - 1$ and n'. The total cost is thus

$$I(n/\ell) + \frac{\ell(\ell-1)}{2}\mathrm{MP}(n/\ell) + \ell M^*(n/\ell),$$

where $\mathrm{MP}(n)$ denotes the cost of a balanced middle product $(2n-1) \times n$, and M^* denotes the cost of a short product. For $\ell = 2$ we get $I(n/2) + \mathrm{MP}(n/2) + 2M^*(n/2)$, and for $\ell = 3$ we get $I(n/3) + 3\mathrm{MP}(n/3) + 3M^*(n/3)$. It is not possible to find the optimal value of ℓ theoretically, since it depends on the relative values of $I(n)$, $\mathrm{MP}(n)$ and $M^*(n)$ and their growth with n (see §III).

III. Experimental Results

We now describe experiments done on `gcc16.fsffrance.org`, a 2.2Ghz AMD Opteron 8354 from the GCC Compile Farm. For those experiments, we used the changeset `131005cc271b` from the 5.0 branch of the GMP repository, which is the one corresponding most closely with the GMP 5.0.1 release, and the corresponding `mulmid` patch from the first author, which implements the integer middle product (the threshold between quadratic and subquadratic middle product was 36 words). We also used the svn revision 7229 from MPFR.

In Figure 3, `mpn_mul_n` is the full $n \times n$ product routine from GMP, `ShortMul` is Mulders' short product from GNU MPFR (svn revision 7191), `mpn_invert` is GMP's internal inversion routine, `mpn_mulmid_n` is Harvey's $(2n-1) \times n$ middle product routine, `mpn_tdiv_qr` is GMP's $(2n)/n$ division routine with remainder, `mpn_div_q` is GMP's internal division routine without remainder (with exact quotient), `ShortDiv` is Algorithm ShortDiv (available in GNU MPFR, svn revision 7191), `FoldDiv(2)` is Algorithm FoldDiv(2), and so on.

words n	100	200	500	1000
`mpn_mul_n`	7.52	22.4	80.8	225
`ShortMul`	0.76	0.81	0.89	0.85
`mpn_invert`	1.21	1.32	1.59	1.57
`mpn_mulmid_n`	1.12	1.20	1.45	1.59
`mpn_tdiv_qr`	1.74	1.86	2.35	2.46
`mpn_div_q`	**1.22**	1.34	1.79	1.87
`ShortDiv`	1.34	**1.32**	1.62	1.75
`FoldDiv(2)`	1.37	1.36	1.62	1.74
`FoldDiv(3)`	1.34	1.35	**1.61**	**1.73**
`FoldDiv(4)`	1.35	**1.32**	1.63	1.76

Figure 3. Comparison of different routines: n is the number of words, the second row gives the time for `mpn_mul_n` in microseconds, the other rows give the ratio of the corresponding routine with respect to `mpn_mul_n`.

Several comments can be made on Fig. 3. As n increases, the gain of the short product (`ShortMul`) over the full product decreases. This is expected, since in the FFT range it is not known how to compute a short product faster than a full product, thus the asymptotic ratio is 1. The middle product ratio also increases with n, whereas we might expect in theory a ratio of 1, by analogy with the polynomial case. There are two explanations for this. Firstly there is a small linear overhead in the subquadratic middle product, due to the computation of the error terms (see [4]). In the context of FoldDiv, it may be possible to reduce this overhead by skipping some of the error term computations and absorbing the resulting errors into the error estimates in the proof of Theorem 2. Secondly, for $n \geq 81$ (on the computer we used), GMP switches to the Toom-3 algorithm for `mpn_mul_n`, which is asymptotically faster than the Karatsuba-like algorithm used by `mpn_mulmid_n`. Thus an implementation of a Toom-3 variant of the middle product may exhibit better timings for `mpn_mulmid_n`, and thus for FoldDiv(ℓ). Remark that one could also implement a middle product using two short products: if X has $2n$ words and Y has n words, their middle product can be obtained from the (high) short product of X_0 by Y and from the (low) short product of X_1 by Y, where X_0 (resp. X_1) denotes the lower (resp. higher) n words from X. Finally `mpn_invert` is still quite expensive, although it has been much improved in GMP 5.0.1; in particular for $n = 200$ `mpn_invert` is not faster than `FoldDiv(4)`! An approximate inverse routine using the middle and short products, along the lines of FoldDiv, should give better timings for FoldDiv(ℓ).

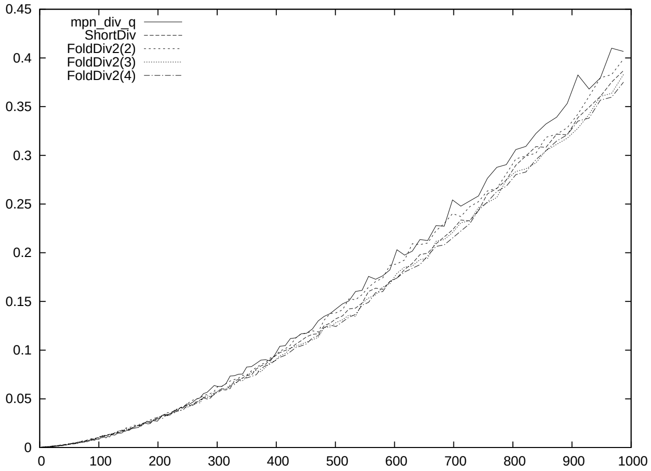

Figure 4. Graphical comparison of different short division routines up to 1000 words (timings in microseconds).

Algorithm ShortMul is implemented in GNU MPFR since version 2.2.0 from September 2005. It is trivial to extend it to the squaring operation, however up to MPFR 3.0.x it was not used in that case, this will be done in the future as a consequence of this work. Another motivation for this work was the new `mpn_div_q` routine in GMP 5.0.x, which is used in GMP's floating-point division `mpf_div`, and as a consequence `mpf_div` was faster than `mpfr_div`. Algorithm ShortDiv is available in GNU MPFR since revision 7191.

IV. Concluding Remarks

In this article, we describe in detail two short division algorithms for multiple-precision integers, namely ShortDiv and FoldDiv. Those algorithms are based on known techniques for polynomials, our contribution being to adapt those techniques to the integer case (dealing with carries), and to provide a rigorous error analysis. As mentioned in the introduction, short division algorithms with rigorous error bounds are very useful as building blocks for floating-point division with correct rounding. The ShortDiv algorithm is easy to implement, since it only depends on a short product routine, and already gives very good timings, in particular it beats GMP 5.0.1 `mpn_div_q` routine up from 200 words. For $n = 500$, `FoldDiv(3)` gives a 31% speedup with respect to `mpn_tdiv_qr`, and a 10% speedup with respect to `mpn_div_q`. The FoldDiv(ℓ) algorithms are more difficult to implement since they rely on the integer middle product, which is not yet available as a basic routine in multiple-precision libraries like GMP, and their error analysis is more involved. However they are slightly faster than ShortDiv, and using $\ell = 3$ seems to be near to optimal. Moreover the efficiency of FoldDiv can still be improved in two ways: on the one hand with a faster middle product routine in the Toom-3 range, on the other hand with an approximate inversion routine using itself short and middle products.

Acknowledgements. The authors thank Torbjörn Granlund for several discussions about the k-fold Barrett division, and Laurent Guerby for maintaining the GCC Compile Farm Project. They also thank the four anonymous reviewers for their careful reading, in particular a reviewer found and corrected an error in Algorithm ShortMulNaive.

References

[1] Barrett, P. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology, Proceedings of Crypto'86* (1987), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 311–323.

[2] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software 33*, 2 (2007), 13:1–13:15.

[3] *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.1 ed., 2010. http://gmplib.org/.

[4] Harvey, D. The Karatsuba middle product for integers. http://www.cims.nyu.edu/~harvey/papers/mulmid/, 2009. Preprint, 15 pages.

[5] Hasenplaugh, W., Gaubatz, G., and Gopal, V. Fast modular reduction. In *Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH-18)* (Montpellier, France, 2007), IEEE Computer Society Press, pp. 225–229.

[6] Karp, A. H., and Markstein, P. High-precision division and square root. *ACM Trans. on Mathematical Software 23*, 4 (1997), 561–589.

[7] Knuth, D. E. *The Art of Computer Programming*, third ed., vol. 2 : Seminumerical Algorithms. Addison-Wesley, 1998. http://www-cs-staff.stanford.edu/~knuth/taocp.html.

[8] Mulders, T. On short multiplications and divisions. *Applicable Algebra in Engineering, Communication and Computing 11*, 1 (2000), 69–88.

[9] van der Hoeven, J. Newton's method and FFT trading. *Journal of Symbolic Computation 45*, 8 (2010), 857–878.