Short Division of Long Integers

David Harvey and Paul Zimmermann

New York University and Inria

25 July 2011

Cambridge Monographs on Applied and Computational Mathematics

Modern Computer Arithmetic focuses on arbitrary-precision algorithms for efficiently performing arithmetic operations such as addition, multiplication and division, and their connections to topics such as modular arithmetic, greatest common divisors, the Fast Fourier Transform (FFT), and the computation of elementary and special functions. Brent and Zimmermann present algorithms that are ready to implement in your favourite language, while keeping a high-level description and avoiding too low-level or machine-dependent details.

The book is intended for anyone interested in the design and implementation of efficient high-precision algorithms for computer arithmetic, and more generally efficient multiple-precision numerical algorithms. It may also be used in a graduate course in mathematics or computer science, for which exercises are included. These vary considerably in difficulty, from easy to small research projects, and expand on topics discussed in the text. Solutions are available from the authors..

Richard Brent is a Professor of Mathematics and Computer Science at the Australian National University, Canberra.

Paul Zimmermann is a Researcher at the Institut National de Recherche en Informatique et en Automatique (INRIA).

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org
ISBN 978-0-521-19469-3

Brent and Zimmermann

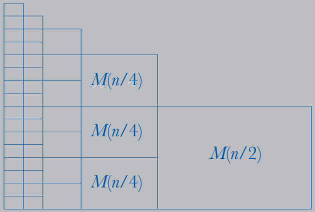
Modern Computer Arithmetic

CAMBRIDGE

Cambridge Monographs on Applied and Computational Mathematics

Modern Computer Arithmetic

Richard Brent and Paul Zimmermann



The problem to be solved

Divide efficiently

a p-bit floating-point number

by another p-bit f-p number

in the 100-10000 digit range

From `www.mpfr.org/mpfr-3.0.0/timings.html` (ms) :

digits	Maple 12.00	Mathematica 6.0.1	Sage 4.5.2	GMP MPF 5.0.1	MPFR 3.0.0
100					
mult	0.0020	0.0006	0.00053	**0.00011**	0.00012
div	0.0029	0.0017	0.00076	**0.00031**	0.00032
sqrt	0.032	0.0018	0.00132	0.00055	**0.00049**
1000					
mult	0.0200	0.007	0.0039	0.0036	**0.0028**
div	0.0200	0.015	0.0071	**0.0040**	0.0058
sqrt	0.160	0.011	0.0064	0.0049	**0.0047**
10000					
mult	0.80	0.28	0.11	0.107	**0.095**
div	0.80	0.56	0.28	**0.198**	0.261
sqrt	3.70	0.36	0.224	0.179	**0.176**

What is GMP (GNU MP) ?

- the most popular library for arbitrary-precision arithmetic
- distributed under a free license (LGPL) from `gmplib.org`
- main developer is Torbjörn Granlund
- contains several layers : `mpn` (arrays of words), `mpz` (integers), `mpq` (rationals), `mpf` (floating-point numbers)
- `mpn` is the low-level layer, with optimized assembly code for common hardware, and provides optimized implementations of state-of-the-art algorithms

Can we do better than GMP ?

An anonymous reviewer said :

What are the paper's weaknesses ?

The resulting performance, in the referee's opinion, is only marginally better a standard exact-quotient algorithm in GMP. One can expect about 10% improvement. It seems to be a weak result for the sophisticated recursive algorithm with the big error analysis effort.

What is GNU MPFR ?

- a widely used library for arbitrary-precision floating-point arithmetic
- distributed under a free license (LGPL) from `mpfr.org`
- main developers are Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Philippe Théveny and Paul Zimmermann
- contrary to GMP `mpf`, implements *correct rounding* and *mathematical functions* (exp, log, sin, ...)
- implements Sections 3.7 (Extended and extendable precisions) and 9.2 (Recommended correctly rounded functions) of IEEE 754-2008
- aims to be (at least) as efficient than other arbitrary-precision floating-point without correct rounding

The problem to be solved (binary fp division)

Assume we want to divide $a > 0$ of p bits by $b > 0$ of p bits, with a quotient c of p bits.

First write $a = m_a \cdot 2^{e_a}$ and $b = m_b \cdot 2^{e_b}$ such that :

- m_b has exactly p bits
- $2^{p-1} \le m_a/m_b < 2^p$ (m_a has $2p - 1$ or $2p$ bits)

The problem reduces to finding the p-bit correct rounding of m_a/m_b with the given rounding mode.

We do not assume that the divisor b is invariant, thus we do not allow precomputations involving b.

Division routine `mpfr_div` in MPFR 3.0.x

The MPFR division routine relies on the (GMP) low-level division with remainder `mpn_divrem`.

`mpn_divrem` computes q and r such that

$$m_a = qm_b + r \qquad \text{with } 0 \le r < m_b.$$

Since $2^{p-1} \le m_a/m_b < 2^p$, q has exactly p bits.

The correct rounding of the quotient is q or $q+1$ depending on the rounding mode.

For rounding to nearest, if $r < m_b/2$, the correct rounding is q; if $r > m_b/2$, the correct rounding is $q+1$.

What's new with GMP 5 ?

In GMP 5, the floating-point division (`mpf_div`) calls `mpn_div_q`, which only computes the (exact) quotient, and is faster (on average) than `mpn_divrem` or its equivalent `mpn_tdiv_qr`.

This is based on an approximate Barrett's algorithm, presented at ICMS 2006.

In most cases computing one more word of the quotient is enough to decide the correct rounding :

- pad the dividend with two zero low words
- pad the divisor with one zero low word
- one will obtain one extra quotient low word

Our goal

Design an approximate division routine for arrays of n words

An array of n words $[a_{n-1}, ..., a_1, a_0]$ represents the integer

$$a_{n-1}\beta^{n-1} + \cdots + a_1\beta + a_0$$

with $\beta = 2^{64}$

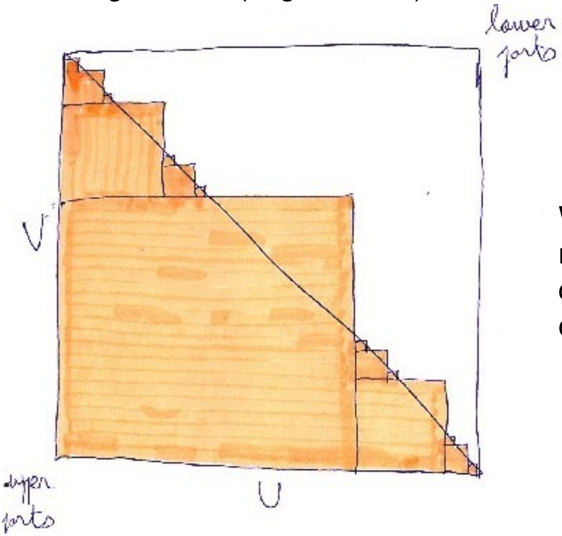
We want a rigorous error analysis and a $O(n)$ error

Plan of the talk

- Mulders' short product
- Mulders' short division
- Barrett's algorithm
- ℓ-fold Barrett's algorithm (cf Hasenplaugh, Gaubatz, Gopal, Arith'18)

Mulders' short product for polynomials (2000)

Short product : compute the upper half of $U \cdot V$, U and V having n terms (degree $n-1$)



With Karatsuba's multiplication, can save 20% over a full product.

Our variant of Mulders's algorithm for integers

Algorithm ShortMul.

Input: $U = \sum_{i=0}^{n-1} u_i \beta^i$, $V = \sum_{i=0}^{n-1} v_i \beta^i$, integer n
Output: an integer approximation W of $UV\beta^{-n}$

if $n < n_0$ **then**
  $W \leftarrow \mathrm{ShortMulNaive}(U, V, n)$
else
  choose a parameter k, $n/2 + 1 \leq k < n$, $\ell \leftarrow n - k$
  write $U = U_1 \beta^\ell + U_0$, $V = V_1 \beta^\ell + V_0$
  write $U = U_1' \beta^k + U_0'$, $V = V_1' \beta^k + V_0'$
  $W_{11} \leftarrow \mathrm{Mul}(U_1, V_1, k)$ $\triangleright$ $2k$ words
  $W_{10} \leftarrow \mathrm{ShortMul}(U_1', V_0, \ell)$ $\triangleright$ ℓ most significant words
  $W_{01} \leftarrow \mathrm{ShortMul}(U_0, V_1', \ell)$ $\triangleright$ ℓ most significant words
  $W \leftarrow \lfloor W_{11} \beta^{2\ell - n} \rfloor + W_{10} + W_{01}$

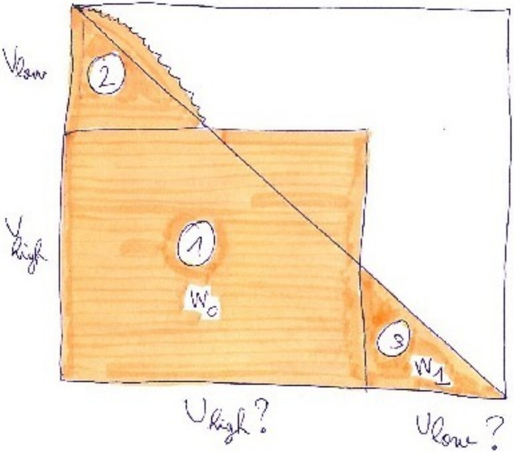
Lemma

The output of Algorithm ShortMul satisfies

$$UV\beta^{-n} - (n-1) < W \leq UV\beta^{-n}.$$

(In other words, the error is less than n ulps.)

Mulders' short division (2000)



U is unknown
V is known
$W = UV$ is known

1. estimate U_{high} from V_{high} and W_{high}, subtract $U_{\text{high}} V_{\text{high}}$ from W
2. compute $U'_{\text{high}} V_{\text{low}}$ and subtract from W
3. estimate U_{low} from V'_{high} and the remainder W

Our variant of Mulders' short division for integers

Algorithm ShortDiv.

Input: $W = \sum_{i=0}^{2n-1} w_i \beta^i$, $V = \sum_{i=0}^{n-1} v_i \beta^i$, with $V \geq \beta^n/2$

Output: an integer approximation U of $Q = \lfloor W/V \rfloor$

if $n < n_1$ **then**
  $U \leftarrow \mathrm{Div}(W, V)$ ▷ Returns $\lfloor W/V \rfloor$
else
  choose a parameter k, $n/2 < k \leq n$, $\ell \leftarrow n - k$
  write $W = W_1 \beta^{2\ell} + W_0$, $V = V_1 \beta^{\ell} + V_0$, $V = V_1' \beta^k + V_0'$
  $(U_1, R_1) \leftarrow \mathrm{DivRem}(W_1, V_1)$
  write $U_1 = U_1' \beta^{k-\ell} + S$ with $0 \leq S < \beta^{k-\ell}$
  $T \leftarrow \mathrm{ShortMul}(U_1', V_0, \ell)$
  $W_{01} \leftarrow R_1 \beta^{\ell} + (W_0 \operatorname{div} \beta^{\ell}) - T\beta^k$
  while $W_{01} < 0$ **do** $(U_1, W_{01}) \leftarrow (U_1 - 1, W_{01} + V)$
  $U_0 \leftarrow \mathrm{ShortDiv}(W_{01} \operatorname{div} \beta^{k-\ell}, V_1', \ell)$
  return $U_1 \beta^{\ell} + U_0$

Lemma

The output U of Algorithm ShortDiv satisfies, with $Q = \lfloor W/V \rfloor$:

$$Q \leq U \leq Q + 2(n-1).$$

(In other words, the error is less than $2n$ ulps.)

The optimal cutoff k in ShortMul and ShortDiv heavily depends on n. There is no simple formula. Instead, we determine the best $k(n)$ by tuning, for say $n < 1000$ words (about 20000 digits).

For ShortMul the best k varies between $0.5n$ and $0.64n$, for ShortDiv it varies between $0.54n$ and $0.88n$ (for a particular computer and a given version of GMP).

Barrett's Algorithm (1987)

Goal : given W and V, compute quotient Q and remainder R :

$$W = QV + R$$

1. compute an approximation I of $1/V$
2. compute an approximation $Q = WI$ of the quotient
3. (optional) compute the remainder $R = W - QV$ and adjust if necessary

When V is not invariant, computing $1/V$ is quite expensive :

- ℓ-fold reduction from Hasenplaugh, Gaubatz, Gopal (Arith'18, 2007) (LSB variant)
- for $\ell = 2$, HGG is exactly Karp-Markstein division (1997)

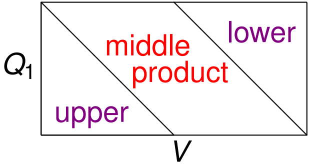
2-fold division (Karp-Markstein)

1. compute an approximation I of $1/V$ to $n/2$ words
2. deduce the upper $n/2$ words $Q_1 = \mathrm{ShortMul}(W, I, n/2)$
3. subtract $Q_1 V$ from W, giving W'
4. deduce the lower $n/2$ words $Q_0 = \mathrm{ShortMul}(W', I, n/2)$

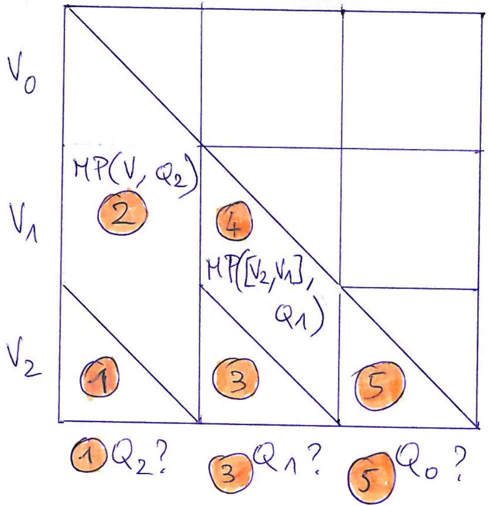
In step 3, $Q_1 V$ is a $(n/2) \times n$ multiplication, giving a $3n/2$ product.

However, we know the upper $n/2$ words match with W, and we are not interested in the lower $n/2$ words.

This is exactly a *middle product* (Hanrot, Quercia, Zimmermann, 2004) :



The 3-fold division algorithm



The integer middle product (Harvey 2009)

Input : X of m words and Y of n words, with $m \geq n$

$$X = \sum_{i=0}^{m-1} x_i \beta^i, \qquad Y = \sum_{j=0}^{n-1} y_j \beta^j$$

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

Lemma

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

Classical case : $m = 2n - 1$ with n^2 word-products.

Quadratic-time algorithms : n^2.

Karatsuba-like middle product : $O(n^{1.58...})$.

FFT-variant : $O(M(n))$.

ℓ-fold Barrett division

Algorithm FoldDiv(ℓ), $\ell \geq 2$.

Input: $W = \sum_{i=0}^{2n-1} w_i\beta^i$, $V = \sum_{i=0}^{n-1} v_i\beta^i$, with $V \geq \beta^n/2$, $W < \beta^n V$
Output: an integer approximation U of $Q = \lfloor W/V \rfloor$

if $n < n_2$ **then**
    return $U \leftarrow \mathrm{Div}(W, V)$
$k \leftarrow \lceil n/\ell \rceil$
write $V = V_1\beta^{n-(k+1)} + V_0$ ▷ V_1 has $k+1$ words
$I \leftarrow \lfloor (\beta^{2(k+1)} - 1)/V_1 \rfloor$
$r \leftarrow n$, $W_r \leftarrow W$, $U \leftarrow 0$
while $r > k+1$ **do** ▷ invariant : $0 \leq W_r < \beta^r V$
    $Q_r \leftarrow \mathrm{ShortMul}(W_r \operatorname{div} \beta^{n+r-(k+1)}, I, k+1)$
    $Q_r \leftarrow \min(Q_r, \beta^{k+1} - 1)$
    $T_r \leftarrow \mathrm{MP}_{r+1,k+1}(V \operatorname{div} \beta^{n-r}, Q_r)$
    $W_{r-k} \leftarrow (W_r - T_r\beta^{n-1}) \operatorname{mods} \beta^{n+r-k}$
    $U \leftarrow U + Q_r\beta^{r-(k+1)}$
    if $W_{r-k} < 0$ **then** $W_{r-k} \leftarrow W_{r-k} + \beta^{r-k}V$, $U \leftarrow U - \beta^{r-k}$
    $r \leftarrow r - k$
$Q_r \leftarrow \mathrm{ShortMul}(W_r \operatorname{div} \beta^{n+r-(k+1)}, I, k+1)$
$U \leftarrow U + (Q_r \operatorname{div} \beta^{k+1-r})$

Theorem

Assuming $n + 8 < \beta/2$ *and* $\ell \leq \sqrt{n/2}$, *Algorithm FoldDiv*(ℓ) *returns an approximation* U *of* $Q = \lfloor W/V \rfloor$, *with error less than* $2n$.

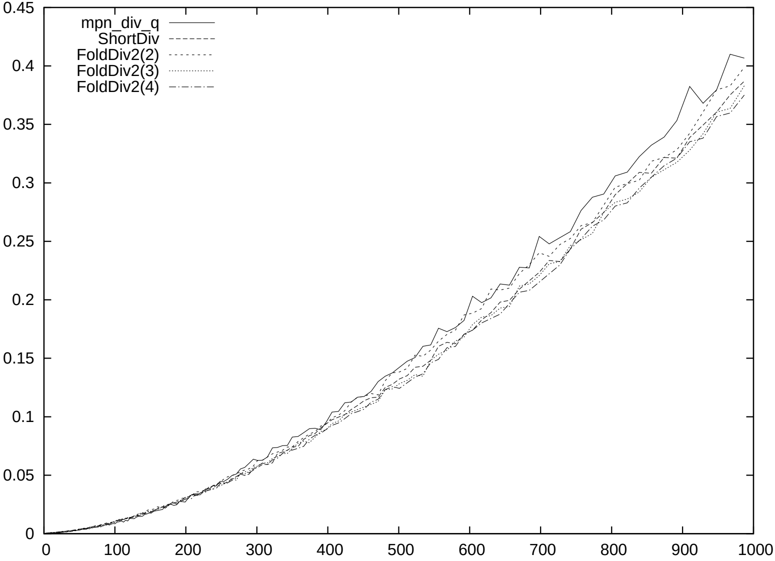
Experimental results

Hardware : `gcc16.fsffrance.org`, 2.2Ghz AMD Opteron 8354

GMP : changeset `131005cc271b` from 5.0 branch ($\approx$ 5.0.1)

`mulmid` patch from David Harvey (threshold 36 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


mpn_div_q
ShortDiv
FoldDiv2(2)
FoldDiv2(3)
FoldDiv2(4)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
100
200
300
400
500
600
700
800
900
1000

Algorithm ShortMul is implemented in GNU MPFR since version 2.2.0 (2005)

Extended to the MPFR squaring operation in 2010

Algorithm ShortDiv is available in GNU MPFR since revision 7191

Algorithm FoldDiv is not (yet) implemented since it requires a middle-product routine, which is not (yet) provided by GMP

Conclusion

Our contributions :

- two variants of Mulders' short product and short division for integers, with detailed description and rigorous error analysis
- a detailed description and rigorous error analysis of the ℓ-fold division for integers
- we get a 10% speedup, and more speedup can be obtained for FoldDiv, by using a Toom-3 middle product, a faster (approximate) inverse based on the same ideas, ...

Benchmarks are a good way to improve software tools !

Still to do : design an approximate inverse using the ℓ-fold algorithm

Adapt the FoldDiv algorithm for an approximate inverse and update the error analysis

ECC 2011

15th workshop on Elliptic Curve Cryptography
September 19-21, 2011
INRIA, Nancy, France

ECC Summer School 2011

September 12-16, 2011

`http://ecc2011.loria.fr/`