Algorithms Seminar 1999–2000,
F. Chyzak (ed.), INRIA, (2000), pp. 43–46.

Available online at the URL
`http://algo.inria.fr/seminars/`.

Efficient Algorithms on Numbers, Polynomials, and Series

Paul Zimmermann
Polka Project, INRIA Lorraine, F–54600 Villers-lès-Nancy, France

January 24, 2000

Summary by Frédéric Chyzak

Abstract

For a computer algebra system, it is crucial to optimize the arithmetical operations on basic objects—numbers, polynomials, series, ... In fact, two classes of objects can be distinguished: integers and polynomials, which require exact operations; floating-point numbers and series, for which only the most significant part of the exact result is needed. The best algorithms currently known for multiplication, division, and square root on integers and floating-point numbers are mostly recent. We present and analyse them using complexity models based on three different multiplication algorithms (naive, Karatsuba, and FFT).

The MPFR library developed by Guillaume Hanrot and Paul Zimmermann is a C library for multiprecision floating-point computations with exact rounding [6]. Its main purpose is to achieve efficiency with a well-defined semantics. Beside the elementary operations $+$, $-$, $\times$, and $/$, it provides routines for square root (with remainder in the integer case, without remainder in the floating-point case), logarithm and exponential. The longer-term goal is to integrate routines for the numerical evaluation of other elementary and special functions as well.

Paul Zimmermann's algorithm for square roots [8] originates in this work. It is reported on here, as well as other recent fast algorithms for multiplications, divisions, and square roots. They all base

Operation	Naive		Karatsuba		FFT	
Method	exact	truncated	exact	truncated	exact	truncated
Multiplication	1	1/2	1	1	1	1
Mulders				0.808		
Division	1	1/2				
Newton			7/2	5/2	5	4
Karp–Markstein			17/6	11/6	9/2	7/2
Jebelean, Burnikel–Ziegler			2	3/2		
Mulders				1.397		
Square root	1/2	1/4				
Newton			7/2	5/2	5	4
Karp–Markstein			17/6	11/6	$9/2^{\dagger}$	$7/2^{\dagger}$
Jebelean, Burnikel–Ziegler			$3/2^{\ddagger}$	$1^{\ddagger}$		
Mulders				$0.966^{\ddagger}$		

FIGURE 1. Complexity of division and square root algorithms in terms of exact multiplications for the three usual multiplication models. Algorithms marked '†', resp. '‡', were analysed, resp. designed and analysed, by Paul Zimmermann in [8].

on Newton's method, which essentially reduces division and square root to a few multiplications. Conversely, division cannot be performed faster than multiplication, for $ab = a/(1/b)$. Thus, once a model for multiplication is chosen, the best to hope is to lessen the constant in the computational complexity of inversion and square rooting. Several approaches to reduce this constant are described and combined in the following sections. To simplify the exposition, carries and their propagation are not taken into account, although they could be accomodated with no conceptual difficulty and no essential change of the complexities.

1. **The Three Classical Multiplication Models**

The naive multiplication algorithm computes a product by convolution between coefficients. Its arithmetical complexity is $N(n) = O\big(n^2\big)$. Karatsuba's recursive algorithm bases on the formula

$$uv = (u_1 b + u_0)(v_1 b + v_0) = u_1 v_1 b^2 + \big((u_1 + v_1)(u_0 + v_0) - u_1 v_1 - u_0 v_0\big)b + u_0 v_0, \tag{1}$$

where only three multiplications are required instead of four by the naive method, yielding the better complexity $K(n) = O\big(n^{\lg 3}\big) = O\big(n^{1.585\cdots}\big)$. A refinement of this idea, splitting each term of the product into more and more parts as n goes to infinity, is the Toom–Cook approach [5]. The improved complexity is $O\left(n^{1+\sqrt{2}/\sqrt{\lg n}} \ln n\right)$. However this algorithm is only a theoretical one. Finally, the fastest known multiplication algorithm relies on FFT (fast Fourier transform) to achieve the complexity $F(n) = O(n \ln n \ln\ln n)$. FFT is a fast recursive method to compute the DFT (discrete Fourier transform) of a polynomial (i.e., its evaluation at each of the nth roots of unity, also called its Fourier coefficients). DFT exchanges product of polynomials—convolution of the coefficients—and point-wise product of the Fourier coefficients. A product of polynomials is thus essentially computed by two direct DFT, mulplication of the Fourier coefficients, and one reverse DFT. Note the following asymptotic relations between arithmetical complexities:

$$N(2n) \sim 4N(n), \quad K(2n) \sim 3K(n), \quad \text{and} \quad F(2n) \sim 2F(n). \tag{2}$$

2. **Newton's Scheme for Inverses and Square Roots**

Newton's schemes respectively given by $\iota(x) = x(2 - ax)$ and $\rho(x) = x(3 - ax^2)/2$ converge to $1/a$ and $1/\sqrt{a}$. This entails that inverses and square roots can be computed by additions and multiplications only, using $b/a = b \times (1/a)$ and $\sqrt{a} = a \times (1/\sqrt{a}\,)$. Both methods have a quadratic convergence rate since

$$\iota\left(\frac{1+\epsilon}{a}\right) = \frac{1-\epsilon^2}{a} \quad \text{and} \quad \rho\left(\frac{1+\epsilon}{\sqrt{a}}\right) = \frac{1 - 3\epsilon^2/2 - \epsilon^3/2}{\sqrt{a}}.$$

This means that the number of correct digits doubles at each step of the iteration.

For a of size n and x of size $n/2$, a naive calculation of $\iota(x)$ would take $5M(n/2)$ arithmetical operations, returning an output of size $2n$. The method is optimized by writing $\iota(x) = x + x(1 - ax)$ and noting that if the $n/2$ digits of x are correct, $1 - ax$ starts with $n/2$ zeroes and ends with a correction of size n, whose first $n/2$ digits only are useful. Thus, only the middle $n/2$ digits of ax are computed in $2M(n/2)$ arithmetical operations, then multiplied with x, then added to x by merely appending them. The overall cost $I(n)$ for inverting a of size n is therefore given by the recurrence $I(n) = 3M(n/2) + I(n/2)$. Unfolding it using (2) yields the asymptotics $2N(n)$ (no improvement), $3K(n)/2$, and $3F(n)$, depending on the multiplication model. Adding 1 for the final multiplications, this gives the constants for the truncated case. In the case of inversion with remainder, the latter is computed after the division as a correcting term, so that another 1 has to be added to the constant.

The same trick works to compute square roots, after writing $\rho(x) = x + x(1 - ax^2)/2$: x^2 is computed in $M(n/2)$ arithmetical operations, then $1 - ax^2$ in $M(n)$ arithmetical operations; the first $n/2$ digits are zero, and only the next $n/2$ ones are multiplied with x in $M(n/2)$ arithmetical operations. The overall cost $S(n)$ to compute $1/\sqrt{a}$ for a of size n is therefore given by the recurrence $S(n) = M(n) + 2M(n/2) + S(n/2)$, which once unfold yields the asymptotics $2N(n)$ (no improvement), $5K(n)/2$, and $4F(n)$, whence the constants for the truncated and exact cases.

3. **Karp and Markstein's Modification of Newton's Method**

Karp and Markstein's improvement is to incorporate the final multiplications $b \times (1/a)$ and $a \times (1/\sqrt{a}\,)$, respectively, into the last step of Newton's method in the corresponding calculation [4].

In the case of the inverse, this corresponds to replacing the last step of the iteration with the computation of $y = bx$, then of $y + x(b - ay)$. Only the first $n/2$ digits of y are kept, and the convergence remains quadratic. As to the complexity, only $M(n/2)$ has been added to the iteration as a replacement for the arithmetical complexity $M(n)$ of a multiplication outside of it. The gain is thus $2K(n)/3$ or $F(n)/2$, depending on the multiplication model.

In the case of the square root, the last step of the iteration is replaced with the computation of $y = ax$, then of $y + x(a - y^2)/2$. Only the last $n/2$ digits of y are kept, the method remains quadratic, and the gains are the same as with inversion.

4. **Burnikel and Ziegler's Division with Remainder**

All the algorithms mentioned above base on Newton's method to reduce manipulations of objects of size $2n$ to manipulations of objects of size n. For a change, Burnikel and Ziegler's improvement of division [1, 3] consists of two mutually recursive algorithms for dividing an object of size $3n$ by an object of size $2n$ and for dividing an object of size $4n$ by an object of size $2n$. The division algorithm obtained in this way was then reused by Zimmermann for the computation of square roots [8].

Algorithm $D_{2/1}$ to divide $u_3b^3 + u_2b^2 + u_1b + u_0$ by $v_1b + v_0$ (where each u_i or v_i is a block of size n and where b is a suitable basis) first computes $(q_1, r_1b + r_0) = D_{3/2}(u_3b^2 + u_2b + u_1, v_1b + v_0)$, then $(q_0, s_1b + s_0) = D_{3/2}(r_1b^2 + r_0b + u_0, v_1b + v_0)$, to return $(q_1b + q_0, s_1b + s_0)$. The arithmetical complexity $D_{2/1}(n)$ to divide an object of size n by an object of size $n/2$ is thus twice the arithmetical complexity $D_{3/2}(n/2)$ to divide an object of size $3n/2$ by an object of size n. For its part, Algorithm $D_{3/2}$ to divide $u_2b^2 + u_1b + u_0$ by $v_1b + v_0$ first computes $(q, c) = D_{2/1}(u_2b + u_1, v_1)$, then $r = r_1b + r_0 = cb + u_0 - qv_0$; next, it decreases q by 1 while adding $v_1b + v_0$ to r until r is nonnegative, before returning (q, r). This 'while' loop is proved to cost little, so that the complexity $D_{3/2}(n)$ is just $D_{2/1}(n) + M(n)$.

Consequently, the complexity $D_{2/1}(n)$ is ruled by the recurrence $D_{2/1}(n) = 2D_{2/1}(n/2) + 2M(n/2)$. This makes no improvement in the case of FFT (complexity $2F(n)\ln n$), but provides a Karatsuba-based exact division of arithmetical complexity $2K(n)$, which is reduced to $3K(n)/2$ for truncated division. Indeed, the truncated variant of Algorithm $D_{2/1}$ calls the exact variant of Algorithm $D_{3/2}$ once, and its truncated variant once. Then, the exact $D_{3/2}$ only uses the exact $D_{2/1}$, while the truncated $D_{3/2}$ calls the truncated $D_{2/1}$. This variant saves as much as $M(n/2) + M(n/4) + \cdots$, that is to say $K(n)/2$ in the Karatsuba model.

Zimmermann's algorithm R to compute the square root of $u_3b^3 + u_2b^2 + u_1b + u_0$ first computes $(s', r') = R(u_3b + u_2)$, then $(q, u) = D_{2/1}(r'b + u_1, 2s')$; it next lets s and r be $s'b + q$ and $(ub + u_0) - q^2$, respectively; if r is nonnegative, it returns (s, r), else $(s, r + 2s - 1)$. The arithmetical complexity $R(n)$ to compute the square root of an object of size n is then given by the recurrence $R(n) = R(n/2) + D_{2/1}(n/2) + M(n/2)$. With multiplications by the Karatsuba algorithm, this reduces to $3K(n)/2$ for the exact case. In the truncated case, the algorithm is modified by calling the truncated variant of $D_{2/1}$ and by not substracting q^2 to define r. The recurrence becomes $R(n) = R(n/2) + D(n/2)$, which in the Karatsuba model delivers a complexity $K(n)$ for square roots without remainder.

5. Mulders' "Short Products"

Mulder's idea is a modification of Karatsuba's algorithm dedicated to the truncated case [7].

Each of the terms u_1v_1, $(u_1+v_1)(u_0+v_0)-u_1v_1-u_0v_0$, and u_0v_0 in Equation (1) has size $2n$ if the input u and v are of size $2n$. In view of a truncated product—or "short product"—, the same relation suggests to compute u_1v_1 exactly, only the most significant half of $(u_1+v_1)(u_0+v_0)-u_1v_1-u_0v_0$, and to save the calculation of u_0v_0. In fact, the simpler form $u_1v_0+u_0v_1$ is used: the product uv is thus reduced to an exact multiplication, u_0v_0, and two truncated multiplications, u_1v_0 and u_0v_1. Unfortunately, unfolding the recurrence $M(n) = K(n/2) + 2M(n/2)$ yields no optimization at all.

The idea is then to vary the sizes of the blocks in u and v: for blocks u_1 and v_1 of size βn, the recurrence becomes $M(n) = K(\beta n) + 2M\big((1-\beta)n\big)$, inducing $M(n) = cK(n)$ for $c = \beta^\alpha/\big(1 - 2(1-\beta)^\alpha\big)$, where $\alpha = \lg 3 = 1.585\ldots$ The optimum is obtained for $\beta \simeq 0.694$ and $c \simeq 0.808$.

The same idea applies to division, with an optimum for $\beta \simeq 0.542$ and $c \simeq 1.397$. Moreover, Zimmermann's algorithm reduces the computation of a truncated square root of an object of size n to an exact square root and a truncated division on objects of size $n/2$; this yields the arithmetical complexity $\simeq (3/2 + 1.397)K(n/2) \simeq 0.966K(n)$ for truncated square root.

6. Other Improvements

Other improvements for the Karatsuba model were announced in the talk: Hanrot and Zimmermann have obtained a better constant for inversion and division ($\simeq 1.212$), which was then used by Quercia to lessen the constant for division without remainder to roughly 1. These works have been further developed since then, with applications to square roots as well [2].

Bibliography

[1] Burnikel (Christoph) and Ziegler (Joachim). – *Fast recursive division.* – Research Report n° MPI-I-98-1-022, Max-Planck-Institut für Informatik, Saarbrücken, Germany, October 1998.

[2] Hanrot (Guillaume), Quercia (Michel), and Zimmermann (Paul). – *Speeding up the division and square root of power series.* – Research Report n° 3973, Institut National de Recherche en Informatique et en Automatique, July 2000. Available from `http://www.inria.fr/RRRT/RR-3973.html`.

[3] Jebelean (Tudor). – Practical integer division with Karatsuba complexity. In Küchlin (Wolfgang W.) (editor), *ISSAC'97 (July 21–23, 1997. Maui, Hawaii, USA).* pp. 339–341. – ACM Press, New York, 1997. Conference proceedings.

[4] Karp (Alan H.) and Markstein (Peter). – High-precision division and square root. *ACM Transactions on Mathematical Software*, vol. 23, n° 4, 1997, pp. 561–589.

[5] Knuth (Donald E.). – *The art of computer programming. Vol. 2.* – Addison-Wesley Publishing Co., Reading, Mass., 1981, second edition, xiii+688p. Seminumerical algorithms, Addison-Wesley Series in Computer Science and Information Processing.

[6] The MPFR library. – Available from `http://www.loria.fr/projets/mpfr/`.

[7] Mulders (Thom). – On short multiplications and divisions. *Applicable Algebra in Engineering, Communication and Computing*, vol. 11, 2000, pp. 69–88.

[8] Zimmermann (Paul). – *Karatsuba square root.* – Research Report n° 3805, Institut National de Recherche en Informatique et en Automatique, November 1999. 8 pages.