The Middle Product Algorithm, I.

Guillaume Hanrot, Michel Quercia, Paul Zimmermann

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The Middle Product Algorithm, I.

Guillaume Hanrot , Michel Quercia and Paul Zimmermann

N° 4664

December 2, 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4664--FR+ENG

INRIA
LORRAINE

The Middle Product Algorithm, I.

Guillaume Hanrot* †, Michel Quercia‡ and Paul Zimmermann*

Thème 2 — Génie logiciel
et calcul symbolique
Projet SPACES

Rapport de recherche n° 4664 — December 2, 2002 — 23 pages

Abstract: We present new algorithms for the inverse, division, and square root of power series. The key trick is a new algorithm — `MiddleProduct` or, for short, `MP` — computing the n middle coefficients of a $(2n-1)\times n$ full product in the same number of multiplications as a full $n\times n$ product. This improves previous work of Brent, Mulders, Karp and Markstein, Burnikel and Ziegler. A forthcoming paper will study the floating-point case.

Key-words: Middle product, inversion, division, square root, Newton's method.

* e-mail : {hanrot, zimmerma}@loria.fr
† corresponding author
‡ 23 rue de Montchapet, 21000 Dijon, e-mail : michel.quercia@prepas.org

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

L'algorithme du produit médian, I.

Résumé : Nous présentons de nouveaux algorithmes pour l'inverse, le quotient et la racine carrée de séries formelles. L'ingrédient clé de ces algorithmes est un nouvel algorithme — appelé `MiddleProduct`, ou plus brièvement `MP` — qui calcule les n coefficients médians d'un produit $(2n-1) \times n$ en le même nombre de multiplications qu'un produit complet $n \times n$. Ces résultats améliorent des travaux de Brent, Mulders, Karp et Markstein, Burnikel et Ziegler. Un travail en préparation étudiera le cas des nombres flottants.

Mots-clés : Produit médian, inversion, division, racine carrée, méthode de Newton

Introduction

One of the major tools for performing arithmetic or computing special functions is Newton's iteration. Indeed its quadratic rate of convergence (so that the number of correct digits or terms doubles at each step) and its self-correcting character make it especially suitable for this kind of computations.

In full generality, Newton's rule for computing a root of a differentiable function $f(x)$ can be written as $x_{k+1} = x_k - f(x_k)/f'(x_k)$. As a rule of thumb, the term x_k should be seen as the main term, whereas $f(x_k)/f'(x_k)$ should be seen as the correcting term. In particular, not all the digits of $f(x_k)/f'(x_k)$ are significant, since the low weight ones will be corrected in the next iteration. Furthermore, since $f(x_k)$ is supposed to tend to zero quickly, one can expect important cancellations in the evaluation of $f(x_k)/f'(x_k)$. The present paper shows how to evaluate only the meaningful part of $f(x_k)/f'(x_k)$, i.e., the "middle" digits, in some situations where the evaluation of f is mainly based on a multiplication. As a consequence, new algorithms for inversion, division and square root are derived.

Another application of the "middle product" is the efficient computation of some transposed multiplications [11].

We will make use of the following notations: $M(n)$ denotes the complexity of the underlying multiplication algorithm on two polynomials[1] of degree n; $K(n)$ denotes the complexity of Karatsuba's algorithm, given by $K(1) = 1$, $K(n) = 2K(\lceil n/2 \rceil) + K(\lfloor n/2 \rfloor)$; $FFT(n)$ denotes the complexity of the Schönhage-Strassen multiplication algorithm [10].

The paper is organized as follows. Section 1 describes the basic trick in the specific case of Karatsuba's algorithm, proves its correctness and analyzes its complexity. Section 2 proves that the trick is in fact functorial and exists for any underlying multiplication algorithm. Sections 3 and 4 show how this trick can be used to compute the square and the inverse of a power series of order n in respectively $\frac{1}{2}K(n)$ and $K(n)$ (or $2FFT(n)$) ring operations. Section 5 studies the applications of these results to the computation of the quotient of two power series; we give two algorithms, one based on the inverse and Karp-Markstein trick that needs $\sim \frac{4}{3}K(n)$ (or $\frac{5}{2}FFT(n)$) ring operations and another one which needs $K(n)$ (or $\frac{1}{2}FFT(n)\log_2(n)$) operations.

In section 6, we study the problem of computing the square root of a power series. We shall give an algorithm of complexity $\frac{3}{4}K(n)$ based on Newton's iteration, and an algorithm of complexity $\frac{3}{4}K(n)$ in the worst case (roughly $0.66K(n)$ on average), based on the square root with remainder.

In section 7, an implementation of these algorithms is presented, which confirms the complexity results in practice.

We conclude the paper by a table summarizing the previous and new worst-case complexities under the Karatsuba and FFT models, for the middle product, inverse, quotient

[1] It may seem more natural to express the complexities of operations on power series in term of multiplications on power series (short products) instead of multiplications on polynomials (full products). For example, if $M^*(n)$ denotes the complexity of a short product, Brent gives in [1, Table 7.1] upper bounds of $3M^*(n)$, $4M^*(n)$, and $5.5M^*(n)$ for the inverse, quotient, and square root respectively. However all known subquadratic algorithms for short products are based on full products, and in the FFT case, it is not even known whether a short product can be computed faster than a full product! This explains our choice of the full product as basic operation.

and square root operations, following work of Brent [1], Mulders [9], Burnikel and Ziegler [2].

1 Newton's iteration for the inverse and the basic trick

Newton's iteration for computing $1/A$ — where A is a number, a polynomial, or a series — follows the recurrence:

$$x_{k+1} = x_k + x_k(1 - Ax_k). \tag{1}$$

Suppose we are looking for an approximation of $1/A$ to precision n. In the last step of Newton's iteration, x_k is then accurate to precision $n/2$, and we have to compute Ax_k to precision n, where the $n/2$ most significant[2] coefficients — or bits — of Ax_k vanish with 1. To get an approximation x_{k+1} to precision n of $1/A$, we multiply x_k by the $n/2$ most significant coefficients of $1 - Ax_k$. The cost of this last step was $3M(n/2)$ so far: $2M(n/2)$ for the product Ax_k which splits into two products of $n/2$ coefficients, and $M(n/2)$ for the product of x_k by $1 - Ax_k$. The total cost of Newton's iteration to compute $1/A$ to precision n is thus $3FFT(n)$ for FFT multiplication and $\frac{3}{2}K(n)$ for Karatsuba multiplication [6].

As we know in advance that the upper $n/2$ bits of Ax_k will vanish with 1, a natural question is the following: is there a faster way to compute directly the $n/2$ required bits — from position $n/2$ to n — of the product Ax_k? We give in this section a positive answer under the Karatsuba model. In the next section, we will show that the full answer lies deeper but applies to any multiplication algorithm. Note that other papers already investigated the computation of only part (usually the most significant bits) of the product of power series or floating-point numbers. See e.g. [8], [9].

1.1 Example: Karatsuba model when n is a power of 2

We first start by describing our trick and then the corresponding algorithm in the simple case when n is a power of 2, and when the underlying multiplication algorithm used is the Karatsuba method.

Basic trick. For the sake of clarity, we suppose A is a Taylor series in the variable t (at $t = 0$). Assume we have $A = a_0 + a_1 t + a_2 t^2$ and $x = x_0 + x_1 t$. We have

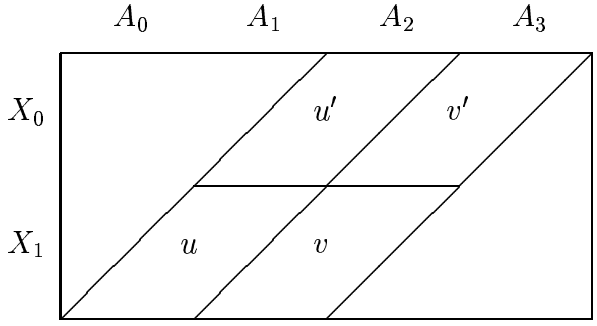
$$Ax = a_0x_0 + (a_0x_1 + a_1x_0)t + (a_1x_1 + a_2x_0)t^2 + a_2x_1t^3$$

and we want to compute

$$(a_0x_1 + a_1x_0)t + (a_1x_1 + a_2x_0)t^2.$$

The algorithm works as follows (see Fig. 1):

[2] By most significant, we mean the low order terms in the case of power series, and the leftmost digits in the usual notation for floating-point numbers.



$$\begin{array}{lcll} u+v & = & \alpha & = \texttt{MP}(X_1,\ [A_0,A_1]+[A_1,A_2]) \\ u'-v & = & \beta & = \texttt{MP}(X_0-X_1,\ [A_1,A_2]) \\ u'+v' & = & \gamma & = \texttt{MP}(X_0,\ [A_1,A_2]+[A_2,A_3]) \end{array}$$

Figure 1: `MiddleProduct` recursion

Algorithm `MiddleProduct`.
Input: $x = x_0 + x_1 t, A = a_0 + a_1 t + a_2 t^2$
Output: $h = a_0 x_1 + a_1 x_0$ and $l = a_1 x_1 + a_2 x_0$
1. $\alpha \leftarrow (a_0 + a_1)x_1$
2. $\beta \leftarrow a_1(x_0 - x_1)$
3. $\gamma \leftarrow (a_1 + a_2)x_0$
4. $h \leftarrow \alpha + \beta$
5. $l \leftarrow \gamma - \beta$.

Now we can use this idea recursively (note that here and in the sequel we use `MP` as a shortcut for `MiddleProduct`). This yields the following algorithm, described in the case when n is a power of 2. See the next section for the general case.

Algorithm $\texttt{MP}([x_0, \ldots, x_{n-1}], [a_0, \ldots, a_{2n-2}])$
0. If $n = 1$, return $[a_0 x_0]$
1. $p \leftarrow n/2$.
2. $\alpha \leftarrow \texttt{MP}([x_p, \ldots, x_{2p-1}], [a_0 + a_p, \ldots, a_{2p-2} + a_{3p-2}])$
3. $\beta \leftarrow \texttt{MP}([x_0 - x_p, \ldots, x_{p-1} - x_{2p-1}], [a_p, \ldots, a_{3p-2}])$
4. $\gamma \leftarrow \texttt{MP}([x_0, \ldots, x_{p-1}], [a_p + a_{2p}, \ldots, a_{3p-2} + a_{4p-2}])$
5. Return $[\alpha + \beta, \gamma - \beta]$.

Theorem 1 *When n is a power of two, Algorithm* `MP` *returns $[c_{n-1}, \ldots, c_{2n-2}]$ where $c_k = \sum_{i+j=k} a_i x_j$, using exactly $K(n) = n^{\log_2 3}$ ring multiplications.*

2 The general situation

In this section, we give a more general and formal presentation of the trick described above. This presentation is suitable for any multiplication algorithm, and gives, in the case of Karatsuba and FFT multiplication, corresponding algorithms of the same complexity for the middle product.

Lemma 1 *Let A be a commutative ring. There is a natural isomorphism*

$$\begin{aligned} \phi_{n,p,q} : \ ((A^n)^* \otimes_A (A^p)^*)^q &\longrightarrow (A^n)^* \otimes_A (A^p)^* \otimes_A (A^q)^* \\ (u_0, \ldots, u_{q-1}) &\longmapsto \sum_{k=0}^{q-1} (u_k \otimes e_k^*), \end{aligned}$$

where (e_k^*) is the dual basis of the canonical basis of A^q. Furthermore, there is also an isomorphism

$$\begin{aligned} \rho_{n,p,q} : \ (A^n)^* \otimes_A (A^p)^* \otimes_A (A^q)^* &\longrightarrow (A^p)^* \otimes_A (A^q)^* \otimes_A (A^n)^* \\ u &\longmapsto (Y, Z, X) \mapsto u(X, \tilde{Y}, Z), \end{aligned}$$

where $Y = (y_0, \ldots, y_{p-1})$ and $\tilde{Y} = (y_{p-1}, \ldots, y_0)$.

Proof. Obvious. ∎

Theorem 2 *Let (f_i^*), (g_j^*), (e_k^*) be the canonical bases of $(A^n)^*$, $(A^p)^*$, $(A^q)^*$.*

$$\textit{Let } \Pi_{n,p,q} = \left(\sum_{\substack{0 \le i < n \\ 0 \le j < p \\ i+j=k}} f_i^* \otimes g_j^* \right)_{0 \le k < q} \qquad \textit{and } M_{p,q,n} = \left(\sum_{\substack{0 \le j < p \\ 0 \le k < q \\ j+k=i+p-1}} g_j^* \otimes e_k^* \right)_{0 \le i < n}.$$

Then

$$M_{p,q,n} = \phi_{p,q,n}^{-1} \circ \rho_{n,p,q} \circ \phi_{n,p,q}(\Pi_{n,p,q}). \tag{2}$$

Proof. This is a matter of a simple calculation:

$$\begin{aligned} \phi_{p,q,n}(M_{p,q,n}) &= \sum_{\substack{0 \le i < n \\ 0 \le j < p \\ 0 \le k < q \\ j+k=i+p-1}} g_j^* \otimes e_k^* \otimes f_i^*, \\ \rho_{n,p,q} \circ \phi_{n,p,q}(\Pi_{n,p,q}) &= \sum_{\substack{0 \le i < n \\ 0 \le j < p \\ 0 \le k < q \\ i+j=k}} g_{p-1-j}^* \otimes e_k^* \otimes f_i^* = \sum_{\substack{0 \le i < n \\ 0 \le j < p \\ 0 \le k < q \\ i+p-1-j=k}} g_j^* \otimes e_k^* \otimes f_i^*. \end{aligned}$$

∎

Corollary 1 *With the notations of Theorem 2, let $X \in A^n$, $Y \in A^p$ and $Z \in A^q$. Then we have*

$$(X \mid M_{p,q,n}(Y, Z)) = (\Pi_{n,p,q}(X, \tilde{Y}) \mid Z), \tag{3}$$

where $(\ \mid\ )$ denotes the canonical inner product of two vectors of the same length.

PROOF. On the one hand we have

$$(X \mid M_{p,q,n}(Y,Z)) = \phi_{p,q,n}(M_{p,q,n})(Y,Z,X),$$

and on the other hand we have

$$(\Pi_{n,p,q}(X,\tilde{Y}) \mid Z) = \phi_{n,p,q}(\Pi_{n,p,q})(X,\tilde{Y},Z) = \rho_{n,p,q} \circ \phi_{n,p,q}(\Pi_{n,p,q})(Y,Z,X).$$

Thus, both sides are equal according to (2). ∎

The $\Pi_{n,p,q}$ and $M_{p,q,n}$ bilinear mappings are related to the product of power series in the following way. Let $X = (x_0, \ldots, x_{n-1}) \in A^n$, $Y = (y_0, \ldots, y_{p-1}) \in A^p$ and $Z = (z_0, \ldots, z_{q-1}) \in A^q$. We have

$$\begin{aligned}
\Pi_{n,p,q}(X,Y) &= \Big(\sum_{i+j=0} x_i y_j, \quad \ldots, \quad \sum_{i+j=q-1} x_i y_j \Big) \\
&= \Big(\text{coefficients of } t^0, \ldots, t^{q-1} \text{ in } (\textstyle\sum x_i t^i)(\sum y_j t^j) \Big), \\
M_{p,q,n}(Y,Z) &= \Big(\sum_{j+k=p-1} y_j z_k, \quad \ldots, \quad \sum_{j+k=n+p-2} y_j z_k \Big) \\
&= \Big(\text{coefficients of } t^{p-1}, \ldots, t^{n+p-2} \text{ in } (\textstyle\sum y_j t^j)(\sum z_k t^k) \Big).
\end{aligned}$$

From a practical point of view, Theorem 2 has the following meaning. One way of seeing usual algorithms for computing products rest on finding a set of products of linear forms in the multiplicands and expressing the coefficients to be computed as linear combinations of those products[3]. Suppose that we have a formula

$$\Pi_{n,p,q}(X,Y) = \sum_{m=1}^{\ell} (a_m \mid X)(b_m \mid Y) c_m \tag{4}$$

with $a_m \in A^n$, $b_m \in A^p$ and $c_m \in A^q$. Such a formula will be called "a decomposition of $\Pi_{n,p,q}$ into ℓ ring multiplications". Then we have

$$\begin{aligned}
(X \mid M_{p,q,n}(Y,Z)) &= (\Pi_{n,p,q}(X,\tilde{Y}) \mid Z) \\
&= \sum_{m=1}^{\ell} (a_m \mid X)(b_m \mid \tilde{Y})(c_m \mid Z) \\
&= \sum_{m=1}^{\ell} (a_m \mid X)(\tilde{b}_m \mid Y)(c_m \mid Z),
\end{aligned}$$

X being an arbitrary vector in A^n. Therefore

$$M_{p,q,n}(Y,Z) = \sum_{m=1}^{\ell} (\tilde{b}_m \mid Y)(c_m \mid Z) a_m, \tag{5}$$

[3]Note that for the algorithm to be efficient, not only has the set of products to be small, but the linear forms and reconstruction formulas have to be evaluated with the smallest possible number of multiplications, see e.g. the case of the FFT.

and $M_{p,q,n}$ can also be decomposed into ℓ ring multiplications.

Matrix multiplication – With the above notations, let us write $a_m = (a_{m,1}, \ldots, a_{m,n})$, $b_m = (b_{m,1}, \ldots, b_{m,p})$, $c_m = (c_{m,1}, \ldots, c_{m,q})$ and let A, B, C be the (ℓ, n), (ℓ, p) and (ℓ, q) corresponding matrices. Then formulas (4)–(5) read:

$$\Pi_{n,p,q}(X, Y) = C^T \times ((A.X^T) * (B.Y^T)), \qquad M_{p,q,n}(Y, Z) = A^T \times ((B.\tilde{Y}^T) * (C.Z^T)),$$

where $*$ denote the component-wise product of two vectors of length ℓ. From the "transposition principle" [5, Def. 8 and Th. 4], we conclude that one can transform a bilinear algorithm $\mathcal{P}$ computing $\Pi_{n,p,q}(X, Y)$ into a bilinear algorithm $\mathcal{M}$ computing $M_{p,q,n}(Y, Z)$. If $\alpha(\mathcal{X})$, $\beta(\mathcal{X})$ and $\gamma(\mathcal{X})$ denote the number of additions, multiplications by a known scalar and multiplications between two indeterminates for algorithm $\mathcal{X}$ then we have

$$\alpha(\mathcal{M}) = \alpha(\mathcal{P}) + q - p, \qquad \beta(\mathcal{M}) = \beta(\mathcal{P}), \qquad \gamma(\mathcal{M}) = \gamma(\mathcal{P}). \tag{6}$$

This means, in the case $p = n$, $q = 2n-1$, that from any multiplication bilinear algorithm of complexity $M(n)$ we can derive an algorithm for `MP` of complexity $M(n)$. The number of additions in the second algorithm exceeds the corresponding number for the first one by $n - 1$.

We illustrate this theorem with two corollaries showing how to derive formulas for computing `MP` under the two most usual non-trivial multiplication models.

Corollary 2 *The Karatsuba algorithm can be adapted so as to yield an algorithm computing* $\texttt{MP}(x, y)$ *in* $K(n)$ *multiplications.*

Proof. Let $X = (x_i)_{0 \le i < n} \in A^n$, $Y = (y_j)_{0 \le j < n} \in A^n$ and $Z = (z_k)_{0 \le k < 2n-1} \in A^{2n-1}$. Write $n = n_0 + n_1$, with $0 < n_0 \le n_1$ and split X, Y in the following way.

$X = [X_0, X_1, X_2]$ with $X_0 = (x_0, \ldots, x_{n_0-1})$, $X_1 = (x_{n_0}, \ldots, x_{n_1-1})$, $X_2 = (x_{n_1}, \ldots, x_{n-1})$,
$Y = [Y_0, Y_1, Y_2]$ with $Y_0 = (y_0, \ldots, y_{n_0-1})$, $Y_1 = (y_{n_0}, \ldots, y_{n_1-1})$, $Y_2 = (y_{n_1}, \ldots, y_{n-1})$.

Note that the subvectors X_1 and Y_1 are empty when $n_0 = n_1$. Furthermore, we define Z_0, Z_{10}, Z_{11} and Z_2 as the following subvectors of Z:

$$\begin{aligned} Z_0 &= (z_0, \ldots, z_{2n_1-2}), \\ Z_{10} &= (z_{n_1}, \ldots, z_{3n_1-2}),\ Z_{11} = (z_{n_1}, \ldots, z_{n_1+2n_0-2}), \\ Z_2 &= (z_{2n_1}, \ldots, z_{2n-2}). \end{aligned}$$

Then, using equation (3) and Karatsuba's method, we can write

$$\begin{aligned} (X \mid M_{n,2n-1,n}(Y, Z)) &= (\Pi_{n,n,2n-1}(X, \tilde{Y}) \mid Z) \\ &= (\Pi_{n_1,n_1,2n_1-1}([X_0, X_1], [\tilde{Y}_2, \tilde{Y}_1]) \mid Z_0 + Z_{10}) \\ &\quad - (\Pi_{n_1,n_1,2n_1-1}([X_0 - X_2, X_1], [\tilde{Y}_2 - \tilde{Y}_0, \tilde{Y}_1]) \mid Z_{10}) \\ &\quad + (\Pi_{n_0,n_0,2n_0-1}(X_2, \tilde{Y}_0) \mid Z_2 + Z_{11}) \\ &= ([X_0, X_1] \mid M_{n_1,2n_1-1,n_1}([Y_1, Y_2], Z_0 + Z_{10})) \end{aligned}$$

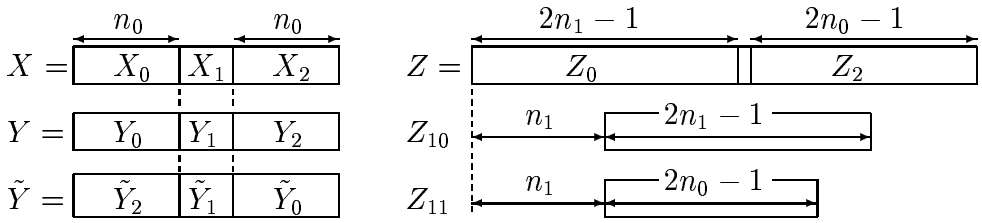


Figure 2: `Karatsuba` and `MP` splitting.

$$\begin{aligned} & - ([X_0 - X_2, X_1] \mid M_{n_1,2n_1-1,n_1}([Y_1, Y_2 - Y_0], Z_{10})) \\ & + (X_2 \mid M_{n_0,2n_0-1,n_0}(Y_0, Z_2 + Z_{11})) \\ = \; & (X \mid [\alpha - \beta, \gamma + \mathrm{low}(\beta)]) \end{aligned}$$

where

$$\begin{aligned} \alpha &= M_{n_1,2n_1-1,n_1}([Y_1, Y_2], Z_0 + Z_{10}), \\ \beta &= M_{n_1,2n_1-1,n_1}([Y_1, Y_2 - Y_0], Z_{10}), \\ \mathrm{low}(\beta) &= (\beta_0, \ldots, \beta_{n_0-1}), \\ \gamma &= M_{n_0,2n_0-1,n_0}(Y_0, Z_2 + Z_{11}). \end{aligned}$$

X being an arbitrary vector of A^n, we infer:

$$M_{n,2n-1,n}(Y, Z) = [\alpha - \beta, \gamma + \mathrm{low}(\beta)]. \tag{7}$$

From a practical viewpoint, this formula yields optimal results for $n_0 = \lfloor n/2 \rfloor$, $n_1 = n - n_0$ (but see the first remark below). Finally, we get a direct Karatsuba-type algorithm for `MP`:

Algorithm `MP-Karatsuba`$([y_0, \ldots, y_{n-1}], [z_0, \ldots, z_{2n-2}])$
0. If $n = 1$, return $[y_0 z_0]$.
1. $n_0 \leftarrow \lfloor n/2 \rfloor$, $\quad n_1 \leftarrow \lceil n/2 \rceil$.
2. $\alpha \leftarrow$ `MP-Karatsuba`$([y_{n_0}, \ldots, y_{n-1}], [z_0 + z_{n_1}, \ldots, z_{2n_1-2} + z_{3n_1-2}])$
3. If n is even
   3.1 $\beta \leftarrow$ `MP-Karatsuba`$([y_{n_1} - y_0, \ldots, y_{n-1} - y_{n_0-1}], [z_{n_1}, \ldots, z_{3n_1-2}])$
   else
   3.2 $\beta \leftarrow$ `MP-Karatsuba`$([y_{n_0}, y_{n_1} - y_0, \ldots, y_{n-1} - y_{n_0-1}], [z_{n_1}, \ldots, z_{3n_1-2}])$
4. $\gamma \leftarrow$ `MP-Karatsuba`$([y_0, \ldots, y_{n_0-1}], [z_{n_1} + z_{2n_1}, \ldots, z_{n_1+2n_0-2} + z_{2n-2}])$
5. Return $[\alpha_0 - \beta_0, \ldots, \alpha_{n_1-1} - \beta_{n_1-1}, \gamma_0 + \beta_0, \ldots, \gamma_{n_0-1} + \beta_{n_0-1}]$.

■

Corollary 3 *Schönhage's FFT algorithm can be adapted so as to yield an algorithm computing* `MP`*(x, y) in* $FFT(n)$ *multiplications.*

Proof. Let ω_n be a fixed primitive $(2n)$-th root of unity, $X \in A^n$, $Y \in A^n$ and $Z \in A^{2n}$. The FFT algorithm becomes, in our formalism,

$$\Pi_{n,n,2n}(X, Y) = \frac{1}{2n} \sum_{m=0}^{2n-1} ((\omega_n^{mi})_{0 \le i < n} | X)((\omega_n^{mj})_{0 \le j < n} | Y)(\omega_n^{-mk})_{0 \le k < 2n}.$$

Hence we obtain from (4)–(5) the formula

$$M_{n,2n,n}(Y,Z) = \frac{1}{2n}\sum_{m=0}^{2n-1}((\omega_n^{mj})_{0\le j<n}|\tilde{Y})((\omega_n^{-mk})_{0\le k<2n}|Z)(\omega_n^{mi})_{0\le i<n}.$$

In more classical notations, if $f_i^{(2n)}$ is the direct Fourier transform and $g_i^{(2n)}$ the inverse one, we get

$$\sum_{\substack{0\le j<n\\0\le k<2n\\j+k=i+n-1}} y_j z_k = \frac{1}{2n}\sum_{m=0}^{2n-1}\omega_n^{mi} f_m^{(2n)}(\tilde{y},0,\ldots,0)g_m^{(2n)}(z), \quad 0\le i<n.$$

This means that $\texttt{MP}(x,y)$ can be computed in two Fourier transforms of size $2n$ and one inverse Fourier transform of size $2n$, which is exactly the same as for a full $n\times n$ product. We leave the task of writing the detailed `MP-FFT` algorithm to the reader.

■

Remark. – The preceding formulas are still valid if we suppose X of length $n+1$:

$$\Pi_{n+1,n,2n}(X,Y) = \frac{1}{2n}\sum_{m=0}^{2n-1}((\omega_n^{mi})_{0\le i\le n}|X)((\omega_n^{mj})_{0\le j<n}|Y)(\omega_n^{-mk})_{0\le k<2n},$$

$$M_{n,2n,n+1}(Y,Z) = \frac{1}{2n}\sum_{m=0}^{2n-1}((\omega_n^{mj})_{0\le j<n}|\tilde{Y})((\omega_n^{-mk})_{0\le k<2n}|Z)(\omega_n^{mi})_{0\le i\le n},$$

therefore we can compute the $n+1$ middle coefficients of the product $(\sum y_j t^j)(\sum z_k t^k)$ with no additional operations (assuming no particular optimization).

3 Squaring power series

In this section, we show how to use the MP algorithm to compute the square of a power series. This algorithm can also be used to compute the upper half part (or lower half part) from the square of a polynomial.

Theorem 3 *From an algorithm computing the product of two power series of order n in $M(n)$ operations, one can deduce – using the MP algorithm – an algorithm computing the square of a power series of order n in $R(n)$ ring multiplications, where $R(n) = R(\lceil n/2\rceil) + M(\lfloor n/2\rfloor) + (n \bmod 2)$, $R(1) = 1$. Under the Karatsuba model, $R(n) \le \frac{K(n)+1}{2}$.*

Proof. We use the following algorithm (at step 4 the second argument of `MP` should read $[2a_2,\ldots,2a_{n-1}]$ for n odd, $p\le 2$). Let $A = \sum_{i=0}^{n-1} a_i t^i = A_0 + t^p A_1$, we have $A^2 = A_0^2 + 2A_0A_1t^p \bmod t^n$, plus an extra term $a_p^2 t^{2p}$ when n is odd. At step 3 we have $\alpha = A^2 \bmod t^{n-p}$, and at step 4 β gives the coefficients of degree $n-p$ to $n-1$ of $A_0^2 + 2A_0A_1t^p$, thus $\alpha + t^{n-p}\beta$, plus $a_p^2t^{2p}$ when n is odd, gives $A^2 \bmod t^n$.

Algorithm `MP-square`$([a_0, \ldots, a_{n-1}])$
1. If $n = 1$, return $[a_0^2]$.
2. $p \leftarrow \lfloor n/2 \rfloor$
3. $\alpha \leftarrow$ `MP-square`$([a_0, \ldots, a_{n-p-1}])$
4. $\beta \leftarrow$ `MP`$([a_0, \ldots, a_{p-1}], [a_{n-2p+1}, \ldots, a_{p-1}, 2a_p, \ldots, 2a_{n-1}])$
5. if $n \bmod 2 = 1$ then $\beta_{p-1} \leftarrow \beta_{p-1} + a_p^2$
6. Return $[\alpha_0, \ldots, \alpha_{n-p-1}, \beta_0, \ldots, \beta_{p-1}]$.

Assume that $R(k) \leq \frac{K(k)+1}{2}$ for $k < n$. If $n = 2k$, we have $R(n) = R(k) + K(k) \leq \frac{3K(k)+1}{2} \leq \frac{K(n)+1}{2}$. If $n = 2k+1$, then $R(n) = R(k+1) + K(k) + 1 \leq \frac{K(k+1)+2K(k)+3}{2} \leq \frac{2K(k+1)+K(k)+1}{2} = \frac{K(n)+1}{2}$ using the fact that the sequence $(K(n))$ is strictly increasing with odd values.[4]

4 Power series inversion

In this section, we explain how the MP algorithm can be applied, when plugged into Newton's iteration, to compute inverses.

Theorem 4 *From an algorithm computing the product of two power series of order n in $M(n)$ operations, one can deduce – using the MP algorithm – an algorithm computing the inverse of a power series of order n in one ring division and $I(n)$ ring multiplications, where $I(n) = I(\lceil n/2 \rceil) + M(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor)$, $I(1) = 0$.*

Proof. We describe the algorithm, which simply rests on Newton's iteration, with the usual product replaced by an MP call. According to the assumptions of the theorem, we are given a multiplication algorithm, which we denote by `mul`, making $M(n)$ operations on two power series of order n. We can thus compute a middle product $n \times (2n-1)$ in $M(n)$ multiplications also.

Algorithm `MP-Inv`$([a_0, \ldots, a_{n-1}])$
0. If $n = 1$, return $[1/a_0]$.
1. $p \leftarrow \lfloor n/2 \rfloor$.
2. $\alpha \leftarrow$ `MP-Inv`$([a_0, \ldots, a_{n-p-1}])$;
3. $\beta \leftarrow$ `MP`$([\alpha_0, \ldots, \alpha_{n-p-1}], [a_1, \ldots, a_{n-1}, 0])$
4. $\gamma \leftarrow$ `mul`$([\alpha_0, \ldots, \alpha_{p-1}], [\beta_0, \ldots, \beta_{p-1}])$
5. Return $[\alpha_0, \ldots, \alpha_{n-p-1}, -\gamma_0, \ldots, -\gamma_{p-1}]$.

Note that the code above assumes in the `MP` call that `MP` can deal with its second argument possibly larger than what is needed (when n is even). Otherwise one just needs to remove the trailing 0 when n is even.

The algorithm performs just one division. The number of multiplications is given by the relation $I(1) = 0$, $I(n) = I(\lceil n/2 \rceil) + M(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor)$. ∎

[4] It can be shown that $K(n+1) - K(n) = 2^{z(n)+1}$ where $z(n)$ is the number of zeroes in the binary expansion of n.

Corollary 4 *Using Karatsuba's algorithm, we get an algorithm of complexity* $I(n) = K(n) - 1$.

PROOF. Due to the fact that $I(n) + 1$ satisfies the same recurrence relation as $K(n)$. ∎

Remark. – The analysis of all the FFT variants from now on is biased by the fact that exact FFT complexity (i.e., exact formulas for $FFT(n)$) highly depends on the implementation. We shall thus content ourselves with "reasonable estimates", assuming mainly that $FFT(2n) \sim 2FFT(n)$ and $FFT(n+1) \sim FFT(n)$.

In the present case, one gets that way $I(n) \sim 2FFT(n)$.

Remark. – More generally, note that the `mul` call can be replaced by a call to Mulders' short product method. This yields no improvement if n is a power of 2, but can otherwise be significantly better. See [9] for more details.

5 Algorithms for power series division

5.1 Newton's method

From our inversion algorithm, we can mechanically deduce a division algorithm as follows:

Theorem 5 *The quotient of two degree* n *polynomials or two order* n *power series — called short division by Mulders — can be computed in* $I(n) + M(n/2)$ *operations.*

PROOF. We use Karp and Markstein's trick [7] to incorporate the dividend in the last Newton's iteration. This can be done by using a short multiplication. Thus the total cost is equivalent to that of one inversion plus one short multiplication of size $n/2$, i.e. $I(n)+M(n/2)$, i.e. $\frac{4}{3}K(n)$ under the Karatsuba model, and $\frac{5}{2}FFT(n)$ under the FFT model. ∎

Corollary 5 *The division with remainder of a polynomial of degree* $2n-1$ *by a polynomial of degree* n *can be performed in* $\frac{7}{2}FFT(n)$.

Remark. – The same remark as above applies concerning Mulders' short product, but the *worst-case* complexity $\frac{4}{3}K(n)$ already improves on the previous best-known algorithm of average complexity $\sim 1.397K(n)$ from Mulders [9, Table 6]. However, see also subsection 5.2.

The corresponding algorithm is as follows:

Algorithm `MP-Div-KM`$([b_0, \ldots, b_{n-1}], [a_0, \ldots, a_{n-1}])$
0. If $n = 1$, return $[b_0/a_0]$.
1. $p \leftarrow \lfloor n/2 \rfloor$.
2. $\alpha \leftarrow$ `MP-Inv`$([a_0, \ldots, a_{n-p-1}])$;
3. $\beta \leftarrow$ `mul`$(\alpha, [b_0, \ldots, b_{n-p-1}])$;
4. $\gamma \leftarrow$ `MP`$(\beta, [a_1, \ldots, a_{n-1}, 0])$;
5. $\delta \leftarrow$ `mul`$([\alpha_0, \ldots, \alpha_{p-1}], [b_{n-p} - \gamma_0, \ldots, b_{n-1} - \gamma_{p-1}])$;
6. Return $[\beta_0, \ldots, \beta_{n-p-1}, \delta_0, \ldots, \delta_{p-1}]$.

5.2 Direct division using MP

Algorithm MP may also be used directly — i.e. without Newton's iteration — to compute the quotient of polynomials or power series. This new algorithm is described and analyzed in this section.

Assume that we are dividing B by A, and split B as (B_0, B_1), and A as (A_0, A_1). Let the quotient be $Q = (Q_0, Q_1)$. Then the algorithm can be illustrated by Figure 3.

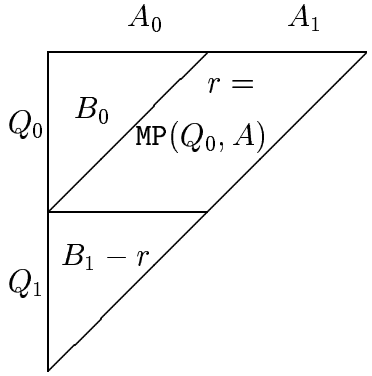


Figure 3: Algorithm MP-Divide.

Namely, the upper part of the quotient Q_0 is obtained by dividing B_0 by A_0. The lower part is then just obtained by dividing B_1 minus the middle terms of the product AQ_0 (hence $\mathtt{MP}(Q_0, A)$) by A_0. This gives more formally the following algorithm:

Algorithm MP-Divide $([b_0, \ldots, b_{n-1}], [a_0, \ldots, a_{n-1}])$.
0. If $n = 1$, return $[b_0/a_0]$.
1. $p \leftarrow \lfloor n/2 \rfloor$.
2. $\alpha \leftarrow \mathtt{MP\text{-}Divide}([b_0, \ldots, b_{n-p-1}], [a_0, \ldots, a_{n-p-1}])$.
3. $r \leftarrow \mathtt{MP}(\alpha, [a_1, \ldots, a_{n-1}, 0])$.
4. $\beta \leftarrow \mathtt{MP\text{-}Divide}([b_{n-p} - r_0, \ldots, b_{n-1} - r_{p-1}], [a_0, \ldots, a_{p-1}])$.
5. Return $[\alpha, \beta]$.

Note that there, as in section 3, we assumed for the sake of simplicity that the second argument of MP can be larger than needed.

Theorem 6 *Algorithm* MP-Divide *with input B of degree n and A of degree n correctly computes the first n terms of their quotient.*

Proof. Follows easily from the explanations of the picture above. ∎

Let $D_m(n)$ be the number of multiplications in the base field, $D_d(n)$ be the number of divisions. Then one has $D_d(n) = D_d(\lfloor n/2 \rfloor) + D_d(\lceil n/2 \rceil)$, and $D_d(1) = 1$, hence $D_d(n) = n$. As for $D_m(n)$, we have $D_m(n) = D_m(\lceil n/2 \rceil) + D_m(\lfloor n/2 \rfloor) + M(\lceil n/2 \rceil)$, $D_m(1) = 0$.

Corollary 6 *If the underlying multiplication algorithm is Karatsuba's algorithm, then one has $D_m(n) = K(n) - n$.*

PROOF. This is obvious by a simple induction, since the recurrence relations are closely related to those that one obtains for Karatsuba method. ∎

In the case of the FFT, we get the "reasonable estimate" $D_m(n) \sim \frac{1}{2} FFT(n) \log_2(n)$ for n large enough. As a consequence, Newton's method should be prefered in that case.

5.3 Division with remainder

Since Algorithm `MP-Divide` computes the quotient of two polynomials in $K(n)$ operations, and the remainder can be obtained from $B - A \cdot Q$ in $K(n)$ additional operations, this yields a division with remainder in $2K(n)$ operations, which equals the best known complexity in the Karatsuba model [2].

We show in this section how to compute the remainder, *i.e.* the low part from $A \cdot Q$, in $\frac{2}{3}K(n)$ operations, which gives a division with remainder in $\frac{5}{3}K(n)$ operations.

In the following algorithm we assume that the high part S of AQ is known (which is the case in division, since this is just the high part of the dividend B). Note that `Mul_low`$([a_0, \ldots, a_{n-1}, [b_0, \ldots, b_{n-1})]$ for a and b of size n is defined as the $n-1$ low terms of the product $a \cdot b$, i.e., $[(\sum_{i=k-n+1}^{n-1} a_i b_{k-i})]_{n \leq k \leq 2n-2}$.

Algorithm `ShortRem`$([a_0, \ldots, a_{n-1}], [q_0, \ldots, q_{n-1}], [s_0, \ldots, s_{n-1}])$
0. $p \leftarrow \lceil n/2 \rceil$.
1. $T \leftarrow$ `Mul_low`$([a_p, \ldots, a_{n-1}], [q_p, \ldots, q_{n-1}])$.
2. If n is even
2.1. $U \leftarrow$ `Mul`$([a_0 + a_p, \ldots, a_{p-1} + a_{n-1}], [q_0 + q_p, \ldots, q_{p-1} + q_{n-1}])$;
2.2 Return $[u_0 + u_p - t_0 - s_0 - s_p, \ldots, u_{p-2} + u_{2p-2} - t_{p-2} - s_{p-2} - s_{n-2}, u_{p-1} - s_{p-1} - s_{n-1}, t_0, \ldots, t_{p-2}]$
else
2.3 $U \leftarrow$ `Mul`$([a_0 + a_p, \ldots, a_{p-2} + a_{n-1}, a_{p-1}], [q_0 + q_p, \ldots, q_{p-2} + q_{n-1}, q_{p-1}])$;
2.4 Return $[u_{p-1} - t_0 - s_{p-1}, u_0 + u_p - t_1 - s_0 - s_p, \ldots, u_{p-4} + u_{2p-4} - t_{p-3} - s_{p-4} - s_{2p-4}, u_{p-3} + u_{2p-3} - s_{p-3} - s_{2p-3}, u_{p-2} + u_{2p-2} - s_{p-2} - s_{n-1}, t_0, \ldots, t_{p-3}]$.

Theorem 7 *Algorithm* `ShortRem` *correctly computes the low half of $A \cdot Q$, and uses $\frac{2}{3}K(n)$ operations under the Karatsuba model, if A and Q have size n.*

PROOF. The idea is that the remainder comes from the computation $A \cdot Q$, which Karatsuba decomposes as the three multiplications A_0Q_0, A_1Q_1 and $(A_0 + A_1)(Q_0 + Q_1)$. But in the present case, the upper half is already known, and we can recover A_0Q_0 from it and the two other products.

More precisely, put $A = A_0 + t^p A_1$, $Q = Q_0 + t^p Q_1$, and write $A_1Q_1 = T_0 + t^{n-p-1}T$, $A_0Q_0 = Z_0 + t^p Z_1$, $(A_0 + A_1)(Q_0 + Q_1) = U_0 + t^{n-p}U_1$, $S = S_0 + t^{n-p}S_1$.

The low part of the remainder is simply the low part of A_1Q_1, i.e. T_1. By definition, S is the high part of

$$A_0Q_0 + t^p(A_0Q_1 + A_1Q_0) + t^{2p}A_1Q_1. \tag{8}$$

If n is even, $S_0 = Z_0$ and $S_1 = Z_1 + U_0 - T_0 - Z_0$. This implies that $Z_1 = S_1 + T_0 - U_0 + S_0$.

According to (8), the high part of the remainder is the low part of (8), i.e., the low part of $(A_0Q_1 + A_1Q_0)$ plus the high part of A_1Q_1, namely $T_0 + U_1 - T - Z_1 = U_0 + U_1 - T - S_0 - S_1$.

If now n is odd, we have $S_0 = Z_0$, $S_1 = Z_1 + t(U_0 - T_0 - Z_0)$, hence $Z_1 = S_1 + tS_0 + tT_0 - tU_0$, and the high part of the remainder is $U_1 - T - Z_1 - tT_0 = tU_0 + U_1 - T - tS_0 - S_1$, and the result follows.

The computation of T is a product of $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$, which costs $K(\lfloor n/2 \rfloor)$; that of $(A_0 + A_1) \cdot (Q_0 + Q_1)$ is a full product $\lceil \frac{n}{2} \rceil \times \lceil \frac{n}{2} \rceil$, which costs $K(\lceil n/2 \rceil)$ too. Thus `ShortRem` costs $K(\lfloor n/2 \rfloor) + K(\lceil n/2 \rceil) \sim \frac{2}{3}K(n)$. ∎

Remark. – since we only need the sum of U_0 or tU_0 and U_1, we may replace the full product $(A_0 + A_1) \cdot (Q_0 + Q_1)$ by a product mod $t^p - 1$, which would directly give the sum $U_0 + U_1$ or $tU_0 + U_1$. If p is even or, better, divisible by a large power of 2, the evaluation of such a product can be improved by simple FFT-like techniques, using the fact that $(x, y) \mapsto (x + y)/2 + t^k(x - y)/2$ is an isomorphism from $A[t]/(t^k - 1) \times A[t]/(t^k + 1)$ onto $A[t]/(t^p - 1)$, when $p = 2k$, if the characteristic of the ring A is odd.

Remark. – When using FFT multiplication, the `ShortRem` algorithm is of no interest, since it replaces a multiplication of size p by two multiplications of size $p/2$. However, the fact that the product $(A_0 + A_1) \cdot (Q_0 + Q_1)$ is needed only modulo $t^p - 1$ (whereas FFT might compute it modulo $t^{2p} - 1$) can probably be used to speed up things.

6 Algorithms for the square root of power series

6.1 Newton's method

From our division algorithm, we can derive a square root algorithm by the classical Newton's iteration $x_{k+1} = x_k + \frac{A - x_k^2}{2x_k}$.

Theorem 8 *The square root of a power series of order n can be computed in $S(n)$ multiplications, with $S(n) = S(\lceil n/2 \rceil) + D(\lfloor n/2 \rfloor) + M(\lceil n/2 \rceil - 1)$, i.e., $S(n) \leq K(n) - n$ under the Karatsuba model and $S(n) \sim \frac{7}{2}FFT(n)$ under the FFT model.*

The corresponding algorithm is as follows:

Algorithm `MP-sqrt`$([a_0, \ldots, a_{n-1}])$
0. If $n = 1$, return $[\sqrt{a_0}]$.
1. $p \leftarrow \lfloor n/2 \rfloor$.
2. $\alpha \leftarrow$ `MP-sqrt`$([a_0, \ldots, a_{n-p-1}])$;
3. $\beta \leftarrow$ `mul`$([\alpha_1, \ldots, \alpha_{n-p-1}], [\alpha_1, \ldots, \alpha_{n-p-1}])$;
4. $\gamma \leftarrow$ `MP-Divide`$([a_{n-p} - \beta_{n-p-2}, \ldots, a_{n-1} - \beta_{n-3}], 2\alpha)$;
5. Return $[\alpha_0, \ldots, \alpha_{n-p-1}, \gamma_0, \ldots, \gamma_{p-1}]$.

where at step 4 it is assumed that $\beta_{n-p-2} = 0$ if $n = 2$, $\beta_{n-3} = 0$ if n is even, and 2α is truncated to p terms.

Remark. – Using the inverse square root iteration $x_{k+1} = x_k + \frac{x_k}{2}(1 - Ax_k^2)$ with Karp/Markstein idea gives $\frac{5}{3}K(n)$ (using `MP` to evaluate $(Ax_k)x_k$, one needs $4K(n/2)$ to go from precision

$n/2$ to n; and the last step on its own with the computation of $y_k = Ax_k$ requires three short products of size $n/2$, i.e. $K(n)$ operations).

Corollary 7 *The square root of a power series can be computed in $\frac{3}{4}K(n)$ multiplications under the Karatsuba model, and $3FFT(n)$ multiplications under the FFT model.*

Proof. If one replaces the full product from step 3 by a short square giving the $n-p-1$ high order terms of α^2 (§3) , one gets $S(n) \leq \frac{3}{4}K(n)$ under the Karatsuba model. In practice, one just need to replace the call to `mul` by a call to `MP-square`.

For the FFT model, instead of dividing $A - x_k^2$ by x_k at step 4 of `MP-sqrt`, we multiply by y_k where $y_k = y_{k-1} + y_{k-1}(1 - x_k y_{k-1})$ is an approximation of $1/\sqrt{A}$, computed together with x_k. For the last step, this replaces one division of size $n/2$ by one multiplication of size $n/2$ — the last y_k is not needed — and for the preceding steps this replaces one division of size $n/2^k$ by three multiplications of size $n/2^k$, whence a total gain of $M(n/2)$ for $D(n) \sim \frac{5}{2}M(n)$. ∎

Corollary 8 *The square root of a polynomial (i.e., a square root with remainder) can be computed in $\frac{5}{4}K(n)$ multiplications under the Karatsuba model.*

Proof. Under the Karatsuba model, we can compute the square root S of a polynomial of degree $2n-2$ in $\frac{3}{4}K(n)$ operations and the $n-1$ low order terms of S^2 in $\frac{1}{2}K(n) + O(1)$ operations using Algorithm `MP-square`, thus we are able to perform a square root with remainder in $\frac{5}{4}K(n)$, improving the $\frac{3}{2}K(n)$ bound from [13].

6.2 A variant

We can also give a slightly more general algorithm. Assume that A is given to precision n, and we have an approximation $A = S^2 + t^p R$ to the precision p. Write $S' = S + \delta t^p$, with δ of size $n-p$. One has $A = S'^2 + Rt^p - \delta t^p(2S' - \delta t^p)$. To get the right S', we just need that $Rt^p - \delta t^p(2S' - \delta t^p)$ be 0 to precision n, hence $\delta = R/(2S + \delta t^p)$ to precision $n-p$. This identity is meaningful as soon as $p > 0$, since the quotient is performed "online": the term of degree k of δ only depends on the terms of δ of degree $\leq k-p$. However for our recursive algorithm to work, all the first half bits of $(2S + \delta t^p)$ are needed to compute the first half of δ. Hence we can give an "in place" formulation of `MP-Divide` finding δ as soon as $p > (n-p)/2$, i.e., $p > n/3$. The previous algorithm corresponds to the case $p = n/2$, hence we can ignore the term δt^p, and simply compute $R/(2S)$ using `MP-Divide`.

Precisely, `MP-Divide-inplace`(a, q, ℓ, n) computes the quotient of $[a_0, \ldots, a_{n-1}]$ by $[q_0, \ldots, q_{n-1}]$ and puts the result in $[q_\ell, \ldots, q_{\ell+n-1}]$.

Algorithm `MP-Divide-inplace` $([a_0, \ldots, a_{n-1}], [q_0, \ldots, q_{\ell+n-1}], \ell, n)$.
0. If $n = 1$ $q_\ell \leftarrow a_0/q_0$. Return.
1. $p \leftarrow \lfloor n/2 \rfloor$.
2. `MP-Divide-inplace`$(a, q, \ell, n-p)$;
3. $\gamma \leftarrow$ `MP`$([q_\ell, \ldots, q_{\ell+n-p-1}], [q_1, \ldots, q_{n-1}, 0])$;
4. `MP-Divide-inplace`$([a_{n-p} - \gamma_0, \ldots, a_{n-1} - \gamma_{p-1}], q, \ell + n - p, p)$.

Algorithm `MP-sqrt-generic`$([a_0, \ldots, a_{n-1}], p)$.
0. If $n = 1$ return $[\sqrt{a_0}]$.
1. $\alpha \leftarrow$ `Sqrt`$([a_0, \ldots, a_{2p-2}])$;
2. $\beta \leftarrow$ `MP-square`(α);
3. $\gamma \leftarrow 2\alpha$.
4. `MP-Divide-inplace`$([a_p - \beta_{p-2}, \ldots, a_{2p-2} - \beta_0, a_{2p-1}, \ldots, a_{n-1}], \gamma, p+1, n-p)$;
4. Return $[\alpha_0, \ldots, \alpha_{p-1}, \gamma_0, \ldots, \gamma_{n-p-1}]$.

One can then try to optimize the choice of the splitting point p. We have computed the value of p giving the optimal number of multiplications under the Karatsuba model. Experimental observations yield the following rule:

- for n between 2^k and $2^k + 2^{k-1}$, choose $p = 2^{k-1}$, *except for* $n = 2^k + 1$ *where one should choose* $p = 2^{k-1} + 1$;
- for n between $2^{k-1} + 2^k$ and 2^{k+1}, choose $p = n - 2^k$.

Using those simple rules, we seem to get the optimal number of multiplications up to a 5% loss; furthermore those rules seem to be asymptotically optimal. The asymptotic behaviour for $S(n)/K(n)$ is then $\limsup S(n)/K(n) = 3/4$ (eg. for $n = 2^k$ where we use in fact the first algorithm) and $\liminf S(n)/K(n)$ seems to be 17/28 (roughly 0.607... for n up to 2^{26}), and obtained for $n = 2^k + 2^{k-1}$. On average we get $2^{-25} \sum_{k=2^{25}}^{2^{26}-1} \frac{S(k)}{K(k)} \approx 0.6607....$

7 Implementation results

Since the algorithms we propose offer only a gain on the constant factor, it seemed to us that a realistic (by opposition with complexity estimates) implementation was required. We describe it shortly in the present section, and give experimental results on several different architectures.

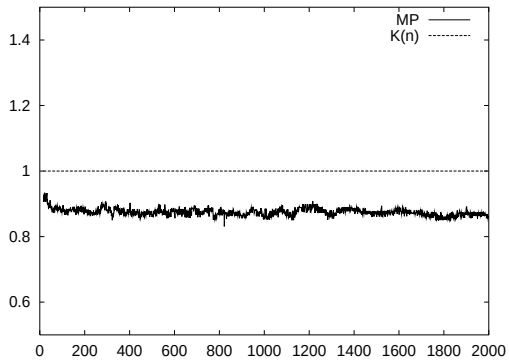
We chose to implement the algorithms on power series with coefficients in $\mathbb{Z}/p\mathbb{Z}$, with p the largest prime such that p^2 fits into a machine word ($p = 65521$ on 32-bit machines, $p = 4294967291$ on 64-bit machines).

For small sizes of arguments, quadratic algorithms are used. The optimal threshold for each algorithm is determined by a tuning program. Experiments have been done for series from degree 16 to 2000.

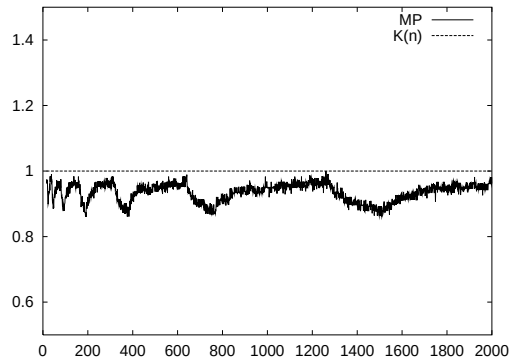
We discuss the results algorithm by algorithm.

7.1 Middle product

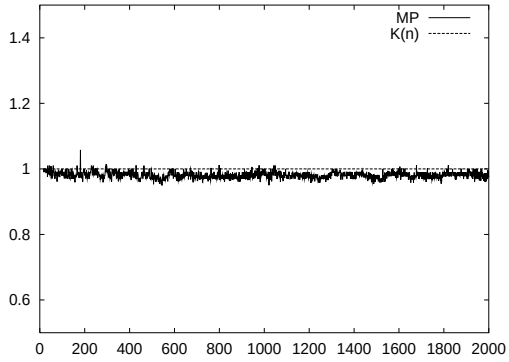
The middle product algorithm has been compared with Karatsuba's algorithm. The thresholds with the quadratic version are very similar on all architectures, though slightly smaller for the middle product. In practice, we recover the expected result $MP(n) \approx K(n)$. We sometimes get (ev6 and K7) a little better, roughly $0.9K(n)$.



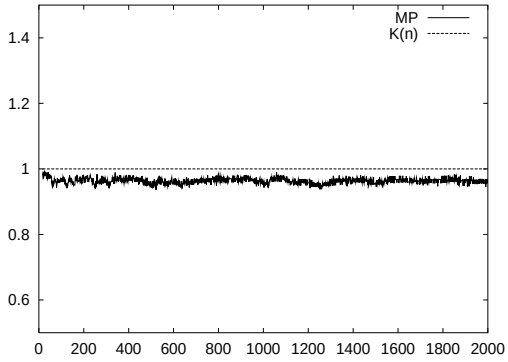
Middle product on a K7-550MHz



Middle product on a ev6-500MHz machine



Middle product on a sparcv9-296 MHz



Middle product on an R10K-194 MHz

7.2 Short square

We have compared the `MP-square` algorithm (see section 3) with the following algorithm:

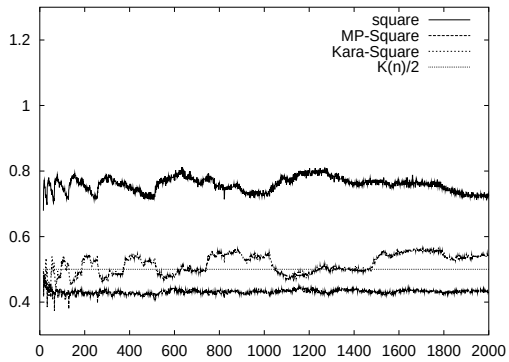
Algorithm `Kara-square`$([a_0, \ldots, a_{n-1}])$
0. If $n = 1$ return $[a_0^2]$.
1. $p \leftarrow \lfloor n/2 \rfloor$.
2. $\alpha \leftarrow$ `mul`$([a_0, \ldots, a_{n-p-1}], [a_0, \ldots, a_{n-p-1}])$;
3. $\beta \leftarrow$ `mul`$([a_0, \ldots, a_{n-p-1}], [a_{n-p}, \ldots, a_{n-1}])$.
4. Return $[\alpha, 2\beta]$.

where the `mul` call at step 3. is a call to a short product (see [9]).

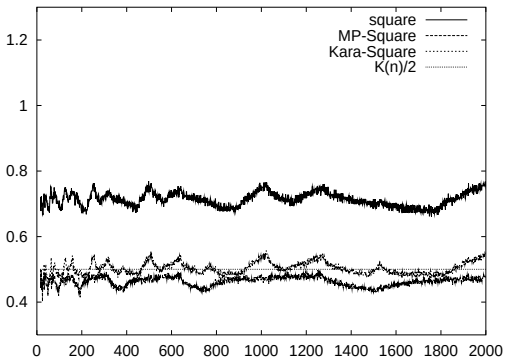
The theoretical complexity of this algorithm is $\frac{2}{3}K(n)$. However, a square is usually less expensive to compute than a product, i.e., the call at step 2 is on average of complexity $0.8K(n/2)$. With the average gain due to Mulders' algorithm, we get an algorithm of practical complexity $0.5K(n)$.

The `MP-square` code is of theoretical complexity $\frac{1}{2}K(n)$.

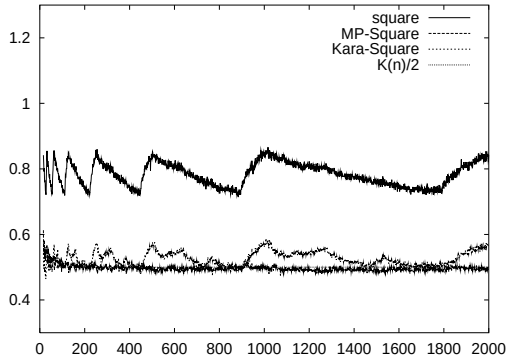
On the following diagrams, the upper curve is the time for execution of a full square using Karatsuba, the second one is the time for computing a short square using `Kara-square` and the third one the time for computing a short square using `MP-square`.



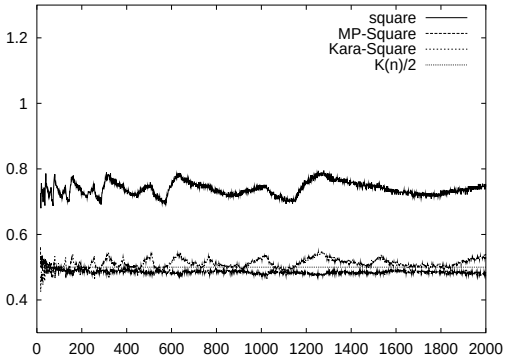
Short square on a K7-550MHz



Short square on a ev6-500MHz machine



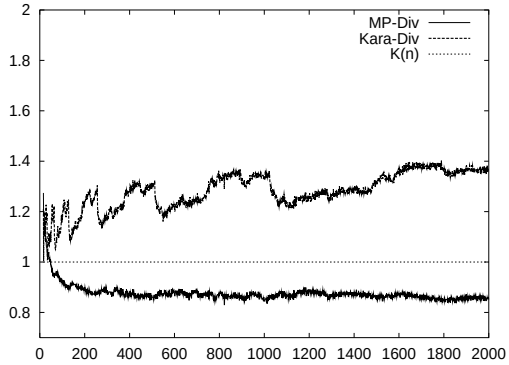
Short square on a sparcv9-296 MHz



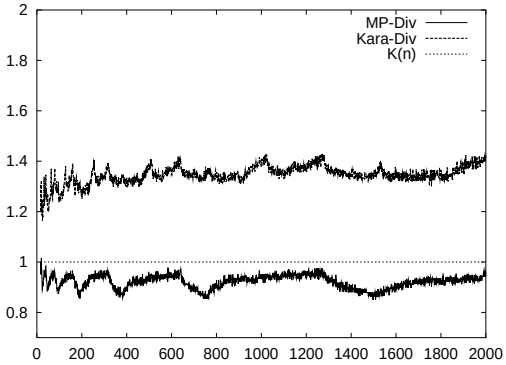
Short square on an R10K-194 MHz

7.3 Division

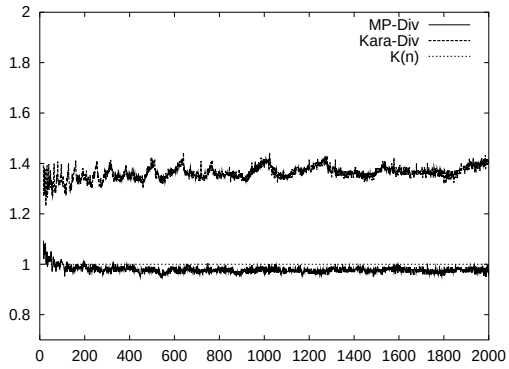
We have compared our `MP-Divide` algorithm with Burnikel-Ziegler division, in which we have used short products to compute the remainder associated to the high part of the quotient. We get the following diagrams:



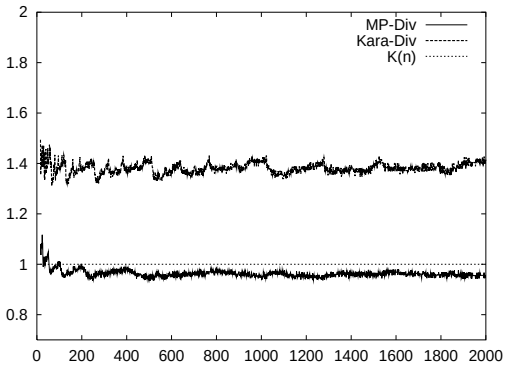
Division on a K7-550MHz



Division on a ev6-500MHz



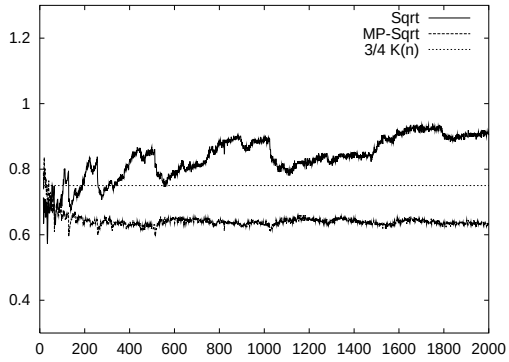
Division on a sparcv9-296 MHz



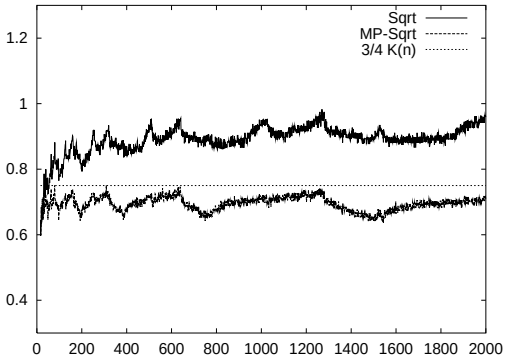
Division on an R10K-194 MHz

7.4 Square root

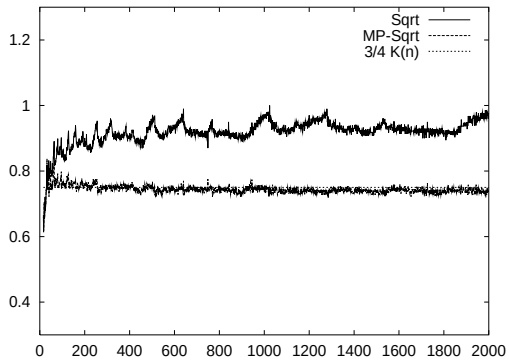
We have compared our `MP-Sqrt` algorithm with Karatsuba square root (see [13]), in which we have used the `Kara-square` algorithm. We get the following diagrams.



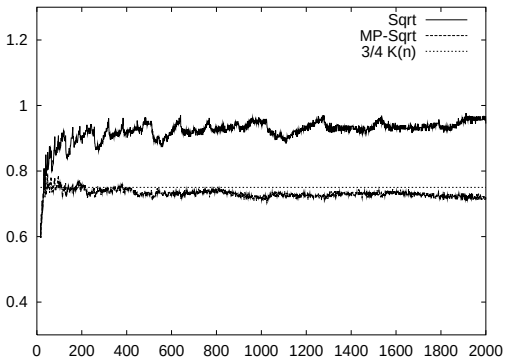
Square root on a K7-550MHz



Square root on a ev6-500MHz machine



Square root on a sparcv9-296 MHz



Square root on an R10K-194 MHz

7.5 Synthesis

The complexity results obtained in this paper, which might be subject to caution since they deal with constants, are reliable. Our experiments confirm that we really obtain those constants in practice. We even get slightly better since the middle product seems to perform

slightly better than Karatsuba on a variety of machines. We do not have an explanation for this phenomenon.

8 Conclusion

This paper presents a new algorithm to compute a short division of two degree n polynomials or series in at most the same number $K(n)$ of arithmetic operations as a full product using Karatsuba's algorithm. An implementation in Maple of these algorithms is available at the URL `http://www.loria.fr/~zimmerma/papers/MP.mpl`.

In addition a new square root algorithm without remainder is presented, with an asymptotic complexity of $\frac{3}{4}K(n)$ in the worst case, roughly $0.66K(n)$ on average. Both algorithms use a new algorithm to compute the n middle coefficients of a $(2n-1) \times n$ product. A detailed analysis of the number of coefficient operations used by those algorithms with respect to previously known algorithms is given.

Our algorithms need only a $O(n)$ memory space, i.e. proportional to the input size. If one enables a larger memory usage, then other algorithms exist. In particular, Joris van der Hoeven showed that a division with remainder can be performed in exactly $K(n)$ operations with so-called *relaxed* algorithms [12].

It is possible to obtain floating point versions of the middle product algorithm *using only O(1) extra memory at each recursion step.* This, and the applications to multiprecision floating point arithmetic, is a work in progress and will be presented in a forthcoming paper [3].

In [11], Victor Shoup asks if a matrix/vector product of the following form can be reduced to a single multiplication of polynomials of degree less than n:

$$\begin{pmatrix} v_0 & v_1 & \cdots & v_{n-1} \\ v_1 & v_2 & \cdots & v_n \\ & & \vdots & \\ v_{n-1} & v_n & \cdots & v_{2n-2} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

This is exactly $\mathtt{MP}([b_{n-1}, \ldots, b_0], [v_0, \ldots, v_{2n-2}])$, whence the "middle product" algorithm partially answers to that open question: it does not reduce the matrix/vector product to a single $n \times n$ product, but to the same number of ring multiplications, provided the product algorithm uses linear forms. This is a particular instance of the "transposition principle" problem [4, Open Problem 6] in the case of Toeplitz matrices: if M is the $(2n-1) \times n$ matrix with $M_{i,j} = x_{i-j}$ for $0 \leq i - j < n$, and $M_{i,j} = 0$ otherwise, y is the vector $(a_0, \ldots, a_{n-1})^T$, and z is the vector $(a_{2n-2}, \ldots, a_0)^T$, then $M \cdot y$ gives the full product of $[x_0, \ldots, x_{n-1}]$ and $[a_0, \ldots, a_{n-1}]$, whereas $M^T \cdot z$ gives $\mathtt{MP}([x_0, \ldots, x_{n-1}], [a_0, \ldots, a_{2n-2}])$, in reversed order.

Table 1 is a synthetic presentation of the results obtained in the present paper. All the complexities from Table 1 are worst-case complexities. Furthermore (cf. Table 2) we improve the complexity of division with remainder to $\frac{7}{2}M(n)$ under the FFT model, that of the square root with remainder to $\frac{5}{4}K(n)$ under the Karatsuba model, and $4M(n)$ under the FFT model.

		Middle prod.	Inverse	Quotient	Square root
Karatsuba model	previous	$2K(n)$	$\frac{3}{2}K(n)$ [2]	$\frac{3}{2}K(n)$ [2]	$K(n)$ [13]
	this paper	$K(n)$ [Co. 2]	$K(n)$ [Co. 4]	$K(n)$ [Co. 6]	$\frac{3}{4}K(n)$ [Co. 7]
FFT model	previous	$2M(n)$	$3M(n)$	$\frac{7}{2}M(n)$	$\frac{7}{2}M(n)$
	this paper	$M(n)$ [Co. 3]	$2M(n)$ [Co. 4]	$\frac{5}{2}M(n)$ [Th. 5]	$3M(n)$ [Co. 7]

Table 1: Previous and new best-known complexities for several operations on power series, under the Karatsuba and FFT models.

		Division with remainder	square root with remainder
Karatsuba model	previous	$2K(n)$ [2]	$\frac{3}{2}K(n)$ [13]
	this paper	$\frac{5}{3}K(n)$ (Th. 7)	$\frac{5}{4}K(n)$ (Co. 8)
FFT model	previous	$\frac{9}{2}M(n)$	$\frac{9}{2}M(n)$
	this paper	$\frac{7}{2}M(n)$ (Co. 5)	$4M(n)$ (Co. 7)

Table 2: Previous and new best-known complexities for several operations on polynomials, under the Karatsuba and FFT models.

Acknowledgements. The authors wish to thank one of the anonymous referees for his clever comments that greatly helped to improve the presentation of the results, and Éric Schost who pointed out the link with the transposition principle.

References

[1] Brent, R. P. The complexity of multiple-precision arithmetic. In *The Complexity of Computational Problem Solving* (1976), R. S. Anderssen and R. P. Brent, Eds., University of Queensland Press, pp. 126–165.

[2] Burnikel, C., and Ziegler, J. Fast recursive division. Research Report MPI-I-98-1-022, MPI Saarbrücken, Oct. 1998.

[3] Hanrot, G., Quercia, M. and Zimmermann, P. The Middle Product Algorithm, II. Floating point arithmetic. In preparation.

[4] Kaltofen, E. Challenges of symbolic computation: My favorite open problems. *Journal of Symbolic Computation 29*, 6 (2000), 891–919.

[5] Kaminski M., Kirpatrick D. G., and Bshouty N. H.. Addition requirements for matrix and transposed matrix products. *J. Algorithms*, 9 (1988), 354–364.

[6] Karatsuba, A. A., and Ofman, Y. P. Multiplication of multiplace numbers by automata. Dokl. Akad. Nauk SSSR **145**, 2, 293–294 (1962).

[7] Karp, A. H., and Markstein, P. High-precision division and square root. *ACM Trans. Math. Softw. 23*, 4 (Dec. 1997), 561–589.

[8] Krandick, W., and Johnson, J. R. Efficient multiprecision floating point multiplication with exact rounding RISC-Linz Report Series, 93-76, 1993, RISC-Linz, Johannes Kepler University.

[9] Mulders, T. On short multiplications and division. AAECC **11**, 1, 69–88 (2000).

[10] Schönhage, A., and Strassen, V. Schnelle Multiplikation großer Zahlen. *Computing 7* (1971), 281–292.

[11] Shoup, V. Efficient computation of minimal polynomials in algebraic extension of finite fields. In *Proceedings of the 1999 International Symposiumon Symbolic and Algebraic Computation* (Vancouver, Canada, July 1999), S. Dooley, Ed., ACM, pp. 53–58.

[12] van der Hoeven, J. Relax, but don't be too lazy. Tech. Rep. 78, Université de Paris-Sud, Mathématiques, Nov. 1999.

[13] Zimmermann, P. Karatsuba square root. Research Report 3805, INRIA, Nov. 1999.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhˆone-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399