The Middle Product Algorithm, I. Speeding up the division and square root of power series.*

Guillaume Hanrot[†], Michel Quercia[‡], Paul Zimmermann[*]

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. These results apply both to series and polynomials.

Keywords : middle product, inversion, division, square root, Newton's method.

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 its evaluation. The present paper shows how to evaluate only the meaningful part of $f(x_k)$, *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].

*Cet article est pariu dans Appl. Alg. Eng. Comm. Comp.**14** (2004), 415–438. Cf. `www.springerlink.com` pour la version de l'éditeur.

†INRIA-Lorraine/LORIA, 615, rue du jardin botanique, F-54602 Villers-lès-Nancy Cedex, e-mail : `{hanrot, zimmerma}@loria.fr`

‡23 rue de Montchapet, 21000 Dijon, e-mail : `michel.quercia@prepas.org`

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: §1 describes the basic trick in the two most usual nontrivial multiplication models, namely Karatsuba and FFT. §2 and §3 show how this trick can be used to compute the square and the inverse of a power series; §4 and §5 study the problem of computing the quotient and square root of power series.

In §6, 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, square and square root operations, following work of Brent [1], Mulders [9], Burnikel and Ziegler [2].

Finally, an appendix gives a formal description of how to transform any multiplication algorithm of a certain type into a middle product algorithm with the same complexity; this is linked to the so-called *transposition principle*.

1 Newton's Inverse Iteration 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

[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.

[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.

— from position $n/2$ to n — of the product Ax_k? We give in this section a positive answer under the Karatsuba and FFT models, and show this result 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 Karatsuba Model for n a Power of 2

We first describe our trick and the corresponding algorithm in the simple case when n is a power of 2, and when the underlying multiplication algorithm 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 $A = a_0 + a_1t + a_2t^2$ and $x = x_0 + x_1t$. 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

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

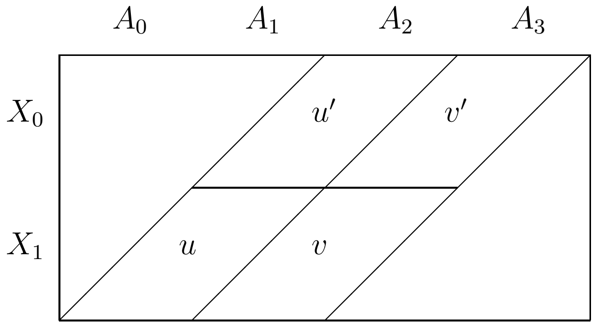
The algorithm works as follows (see Fig. 1):

Algorithm `MiddleProduct`.
Input: $x = x_0 + x_1t, A = a_0 + a_1t + a_2t^2$
Output: $h = a_0x_1 + a_1x_0$ and $l = a_1x_1 + a_2x_0$
1. $\alpha \leftarrow (a_0 + a_1)x_1$
2. $\beta \leftarrow a_1(x_1 - x_0)$
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 §1.2 for the general case.

Algorithm $\mathtt{MP}([x_0, \ldots, x_{n-1}], [a_0, \ldots, a_{2n-2}])$.
0. If $n = 1$, return $[a_0x_0]$
1. $p \leftarrow n/2$.
2. $\alpha \leftarrow \mathtt{MP}([x_p, \ldots, x_{2p-1}], [a_0 + a_p, \ldots, a_{2p-2} + a_{3p-2}])$
3. $\beta \leftarrow \mathtt{MP}([x_p - x_0, \ldots, x_{2p-1} - x_{p-1}], [a_p, \ldots, a_{3p-2}])$
4. $\gamma \leftarrow \mathtt{MP}([x_0, \ldots, x_{p-1}], [a_p + a_{2p}, \ldots, a_{3p-2} + a_{4p-2}])$
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 $c_k = \sum_{i+j=k} a_ix_j$, using exactly $K(n) = n^{\log_2 3}$ ring multiplications.*



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

Figure 1: `MiddleProduct` recursion

1.2 Karatsuba Model: The Odd Case

In this section, we generalize the trick described above to deal with the case when n is not a power of 2. If R is the coefficient ring of our power series, and $y := [y_0, \ldots, y_{n-1}] \in R^n$, $z := [z_0, \ldots, z_{2n-2}] \in R^{2n-1}$, we define $\mathtt{MP}(y, z) = [\sum_{i+j=k} y_i z_j]_{n-1 \le k \le 2n-2}$.

Theorem 2 *Karatsuba's algorithm can be adapted so as to yield an algorithm computing* $\mathtt{MP}(y, z)$ *in* $K(n)$ *multiplications.*

Proof. Consider the following algorithm:

Algorithm $\mathtt{MP\text{-}Karatsuba}([y_0, \ldots, y_{n-1}], [z_0, \ldots, z_{2n-2}])$.
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. $\alpha \leftarrow \mathtt{MP\text{-}Karatsuba}([y_{n_0}, \ldots, y_{n-1}], [z_0 + z_{n_1}, \ldots, z_{2n_1-2} + z_{3n_1-2}])$
3. If n is even
    3.1 $\beta \leftarrow \mathtt{MP\text{-}Karatsuba}([y_{n_1} - y_0, \ldots, y_{n-1} - y_{n_0-1}], [z_{n_1}, \ldots, z_{3n_1-2}])$
else
    3.2 $\beta \leftarrow \mathtt{MP\text{-}Karatsuba}([y_{n_0}, y_{n_1} - y_0, \ldots, y_{n-1} - y_{n_0-1}], [z_{n_1}, \ldots, z_{3n_1-2}])$
4. $\gamma \leftarrow \mathtt{MP\text{-}Karatsuba}([y_0, \ldots, y_{n_0-1}], [z_{n_1} + z_{2n_1}, \ldots, z_{n_1+2n_0-2} + z_{2n-2}])$
5. Return $[\alpha_0 - \beta_0, \ldots, \alpha_{n_1-1} - \beta_{n_1-1}, \gamma_0 + \beta_0, \ldots, \gamma_{n_0-1} + \beta_{n_0-1}]$.

A straightforward calculation or using the general formulation developed in the appendix then allows one to complete the proof. ∎

1.3 The FFT Model

Theorem 3 *Schönhage-Strassen's FFT algorithm can be adapted so as to yield an algorithm computing* $\mathtt{MP}$*(y, z) in* $FFT(n)$ *multiplications.*

PROOF. We give the algorithm and refer to the general framework described in the appendix for a detailed proof.

Algorithm $\texttt{MP-FFT}([y_0, \ldots, y_{n-1}], [z_0, \ldots, z_{2n-2}])$.
0. $\omega \leftarrow$ primitive $(2n)$-th root of unity
1. $\alpha \leftarrow \texttt{FFT}(\omega, [y_{n-1}, \ldots, y_0, \underbrace{0, \ldots, 0}_{n}])$
2. $\beta \leftarrow \texttt{FFT}(\omega^{-1}, [z_0, \ldots, z_{2n-2}, 0])$
3. $\gamma \leftarrow \texttt{FFT}(\omega, [\alpha_0\beta_0, \ldots, \alpha_{2n-1}\beta_{2n-1}])$
4. Return $[\gamma_0/2n, \ldots, \gamma_{n-1}/2n]$).

$\texttt{FFT}(a, v)$ computes $[\sum_{i=0}^{2n-1} a^{mi} v_i]_{0 \le m \le 2n-1}$, where a is a root of unity of order $2n$. At step 3, $\gamma = [c_{n-1}, \ldots, c_{2n-2}, c_{2n-1}, c_{2n} + c_0, \ldots, c_{3n-2} + c_{n-2}]$, where $c_k = \sum_{i+j=k} y_i z_j$. ■

1.4 General Situation

The two algorithms described in that section can be generalized: to any multiplication algorithm of a certain — quite general — kind, we can associate a middle product algorithm of the same complexity. (See [5, Def. 8] for the definition of a bilinear algorithm.)

Theorem 4 *From any bilinear (n, n) multiplication algorithm, one can derive an algorithm computing a $(n, 2n-1)$ middle product with exactly the same number of ring multiplications.*

Since the proof of that theorem is quite technical, and not required in the rest of the paper, we postpone it in an appendix.

In the following sections, we consider an algorithm $\texttt{MP}(y, z)$ taking y with n terms, z with $2n - 1$ terms, and returning their middle product, with the same complexity — $K(n)$ or $FFT(n)$ depending on the computation model — than multiplication. We then use this algorithm to build new algorithms for basic operations on power series or polynomials: square, inverse, quotient, square root. Sometimes to simplify the algorithms we allow z to have more terms; in such a case we consider only the first $2n-1$ terms from z, and we write $\texttt{MP}(y, z, n)$ to avoid confusion. We also write $\texttt{mul}(a, b)$ for the full product of two series or polynomials of n terms (giving $2n - 1$ terms), $\texttt{mul\_high}(a, b)$ for the (short) product of two series of n terms, giving the n most significant terms, of order 0 to $n-1$, and $\texttt{mul\_low}(a, b)$ for the short product giving the $n - 1$ least significant terms, of order n to $2n - 2$.

2 Squaring Power Series

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

Theorem 5 *Using the MP algorithm, one can compute the square of a power series of order n in $R(n) \le \frac{K(n)+1}{2}$ ring multiplications under the Karatsuba model.*

Proof. We use the following algorithm. 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 — where the second argument of MP should read $[2a_2]$ for $n = 3$ — 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 $a_p^2 t^{2p}$ 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 $[\alpha_0, \ldots, \alpha_{n-p-1}, \beta_0, \ldots, \beta_{p-1}]$.

The number $R(n)$ of ring multiplications satisfies the recurrence $R(n) = R(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor) + (n \bmod 2)$, with $R(1) = 1$. In the Karatsuba model, 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 $R(n) = R(k+1) + K(k) + 1 \leq \frac{K(k+1)+2K(k)+3}{2} \leq \frac{2K(k+1)+K(k)+1}{2} \leq \frac{K(n)+1}{2}$, using the fact that $K(k) + 2 \leq K(k+1)$.[3] ∎

3 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 6 *Using the MP algorithm, one can inverse a power series of order n in $I(n) = K(n) - 1$ ring multiplications under Karatsuba's model, and $I(n) \sim 2FFT(n)$ using FFT.*

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 $FFT(n+1) \sim FFT(n)$.

Proof. We describe the algorithm, which simply rests on Newton's iteration, with the usual product replaced by an MP call.

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`($[\alpha_0, \ldots, \alpha_{n-p-1}], [a_1, \ldots, a_{n-1}, 0], n-p$)
4. $\gamma \leftarrow$ `mul`($[\alpha_0, \ldots, \alpha_{p-1}], [\beta_0, \ldots, \beta_{p-1}]$)
5. Return $[\alpha_0, \ldots, \alpha_{n-p-1}, -\gamma_0, \ldots, -\gamma_{p-1}]$.

[3] 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.

The algorithm performs just one division. The number $I(n)$ of ring multiplications is given by $I(1) = 0$, $I(n) = I(\lceil n/2 \rceil) + M(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor)$. Under Karatsuba's framework, we have $I(n) = K(n) - 1$, since $I(n) + 1$ satisfies the same recurrence relation as $K(n)$. In the FFT case, one gets $I(n) \sim 2FFT(n)$. ∎

Remark. – The `mul` call at step 4 of `MP-inv` may be replaced by a call to Mulders' short product. This yields no improvement if n is a power of 2, but can otherwise be significantly better. See [9] for more details.

4 Power Series Division

4.1 Newton's Method

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

Theorem 7 *The quotient of two order n power series — Mulders' short division — can be computed in $\frac{4}{3}K(n)$ under the Karatsuba model, and $\frac{5}{2}FFT(n)$ under the FFT model.*

As a consequence, the division with remainder of a polynomial of degree $2n-1$ by a polynomial of degree n can be performed in $\frac{7}{2}FFT(n)$.

Proof. We use Karp and Markstein's trick [7] to incorporate the dividend in the last Newton's iteration. 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`$([\beta_0, \ldots, \beta_{n-p-1}], [a_1, \ldots, a_{n-1}, 0], n-p)$;
5. $\delta \leftarrow$ `mul`$([\alpha_0, \ldots, \alpha_{p-1}], [b_{n-p} - \gamma_0, \ldots, b_{n-1} - \gamma_{p-1}])$;
6. Return $[\beta_0, \ldots, \beta_{n-p-1}, \delta_0, \ldots, \delta_{p-1}]$.

The total cost is equivalent to that of one inversion plus one short multiplication of size $n/2$, *i.e.*, $I(n) + M(n/2)$, or $\frac{4}{3}K(n)$ with Karatsuba, and $\frac{5}{2}FFT(n)$ using the FFT. ∎

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 §4.2.

4.2 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.

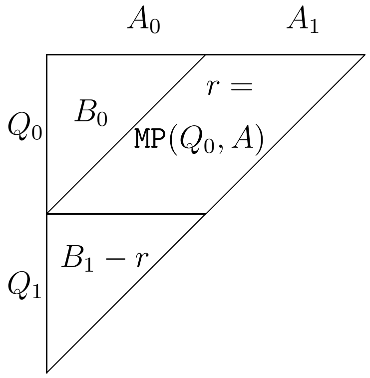


Figure 2: Algorithm `MP-divide`.

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 $Q = (Q_0, Q_1)$. The algorithm is illustrated by Figure 2.

Namely, the upper part of the quotient Q_0 is obtained by dividing B_0 by A_0. The lower part is then 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. $r \leftarrow \mathtt{MP}(\alpha, [a_1, \ldots, a_{n-1}, 0], n - p)$.
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 $[\alpha, \beta]$.

Theorem 8 *Algorithm* `MP-divide` *with inputs of n terms correctly computes the first n terms of their quotient, using $D(n) = K(n) - n$ ring multiplications under Karatsuba model.*

Proof. The correctness follows easily from the explanations of the picture above. Let $D(n)$ 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 $D(n)$, we have $D(n) = D(\lceil n/2 \rceil) + D(\lfloor n/2 \rfloor) + M(\lceil n/2 \rceil)$, $D(1) = 0$. Under Karatsuba model, we get $D(n) = K(n) - n$ 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" $D(n) \sim \frac{1}{2} FFT(n) \log_2(n)$ for n large enough. As a consequence, Newton's method should be preferred in that case.

4.3 Division With Remainder

Consider two series A and B of n terms. Since Algorithm `MP-divide` computes their quotient Q in $K(n)$ operations, and the remainder can be obtained from $B - AQ$ 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]. (Considering series at infinity instead of zero, all results transpose to polynomials.)

We show in this section how to compute the remainder, *i.e.*, the low part from AQ, 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).

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 9 *Algorithm* `ShortRem` *correctly computes the low half of AQ, and uses $\frac{2}{3}K(n)$ operations under the Karatsuba model, when A and Q have n terms.*

PROOF. The idea is that the remainder comes from the computation AQ, which Karatsuba decomposes as 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 $A = A_0 + t^pA_1$, $Q = Q_0 + t^pQ_1$, and write $A_1Q_1 = T_0 + t^{n-p}T$, $A_0Q_0 = Z_0 + t^pZ_1$, $(A_0 + A_1)(Q_0 + Q_1) = U_0 + t^{n-p}U_1$, $S = S_0 + t^{n-p}S_1$.

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

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

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

The high part of the remainder is the high part of the low part of (2), *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 $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$ product, which costs $K(\lfloor n/2 \rfloor)$; that of $U = (A_0 + A_1)\cdot(Q_0 + Q_1)$ is a full $\lceil \frac{n}{2} \rceil \times \lceil \frac{n}{2} \rceil$ product, which costs $K(\lceil n/2 \rceil)$. 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 $R[t]/(t^k - 1) \times R[t]/(t^k + 1)$ onto $R[t]/(t^p - 1)$, when $p = 2k$, if the characteristic of the ring R is odd.

Remark. – When using FFT multiplication, Algorithm `ShortRem` is of little interest, since it replaces a multiplication of size n by two multiplications of size $n/2$. However, the fact that the product $(A_0 + A_1)(Q_0 + Q_1)$ is needed only modulo $t^p - 1$ (whereas FFT might compute it modulo $t^{2p} - 1$) can probably be used to speed up things.

5 Square Root of Power Series

5.1 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 10 *The square root of a power series of order n can be computed in $S(n) \le K(n)$ ring multiplications under the Karatsuba model, $S(n) \sim \frac{7}{2}FFT(n)$ under the FFT model.*

Proof. 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. The number $S(n)$ of ring multiplications satisfies $S(1) = 0$, $S(n) = S(\lceil n/2 \rceil) + D(\lfloor n/2 \rfloor) + M(\lceil n/2 \rceil - 1)$. We get $S(n) \le K(n) - n$ under the Karatsuba model and $S(n) \sim \frac{7}{2}FFT(n)$ under the FFT model. ∎

Corollary 1 *The square root (resp. square root with remainder) of a power series can be computed in $\frac{3}{4}K(n)$ (resp. $\frac{5}{4}K(n)$) multiplications under the Karatsuba model, and $3FFT(n)$ (resp. $4FFT(n)$) 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 (see §2) , one gets $S(n) \le \frac{3}{4}K(n)$ 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 $n/2^j$ by three multiplications of size $n/2^j$ for $j \geq 2$, whence a total gain of $M(n/2)$ for $D(n) \sim \frac{5}{2}M(n)$.

To get the complexities for the square root with remainder, just add $\frac{1}{2}K(n)$ under the Karatsuba model (using Algorithm `MP-square`) and $FFT(n)$ under the FFT model. ■

5.2 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 precision p. Write $S' = S + \delta t^p$, with δ of order $n - p$. 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.*, $p > n/3$. Algorithm `MP-sqrt` 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-div-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-div-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-div-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-div-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. $\alpha \leftarrow$ `MP-sqrt-generic`$([a_0, \ldots, a_{p-1}])$;
2. $\beta \leftarrow$ `MP-square`(α);
3. `MP-div-inplace`$([a_p - \beta_{p-2}, \ldots, a_{2p-2} - \beta_0, a_{2p-1}, \ldots, a_{n-1}], 2\alpha, p+1, n-p)$;
4. Return $[\alpha_0, \ldots, \alpha_{p-1}, 2\alpha_0, \ldots, 2\alpha_{n-p-1}]$.

One can then try to optimize the choice of the splitting point p. A theoretical analysis in the Karatsuba model, taking $p = \lfloor \beta n \rceil$, gives $\beta_{\text{opt}} \approx 0.411$, with a cost $\approx 0.734K(n)$. 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 $2^{-25} \sum_{k=2^{25}}^{2^{26}-1} \frac{S(k)}{K(k)} \approx 0.6607...$.

6 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 architectures.

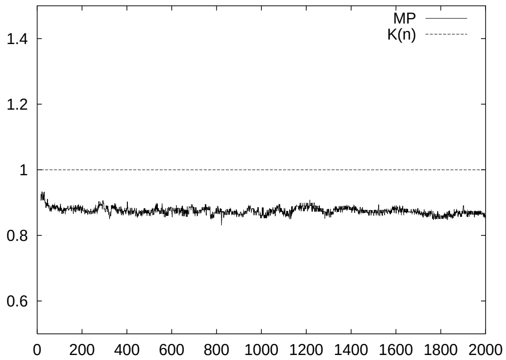
We choose to implement the algorithms on power series with coefficients in $\mathbb{Z}/p\mathbb{Z}$, with p the largest prime such that p^2 fits into a machine word ($p = 65521$ on 32-bit machines, $p = 4294967291$ on 64-bit machines).

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.

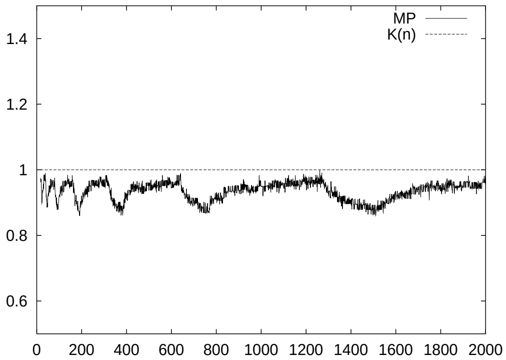
We discuss the results algorithm by algorithm.

6.1 Middle Product

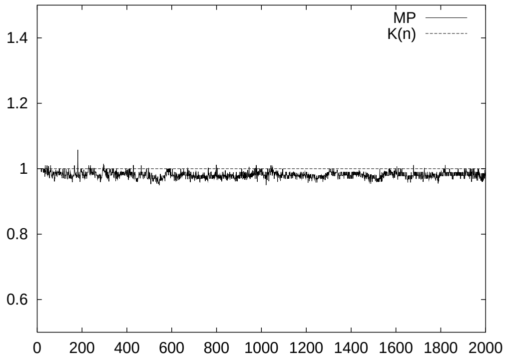
The middle product algorithm has been compared with Karatsuba's algorithm. The thresholds with the quadratic version are very similar on all architectures, though slightly smaller for the middle product. In practice, we recover the expected result $MP(n) \approx K(n)$. We sometimes get (ev6 and k7) a little better, roughly $0.9K(n)$.



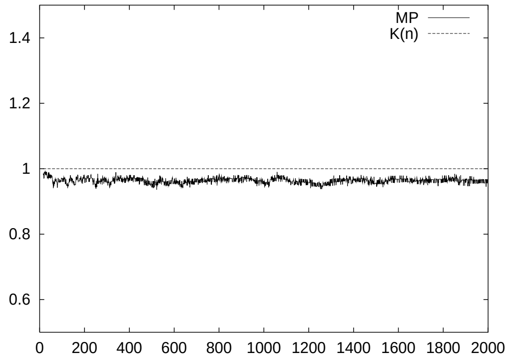
Middle product on a k7-550MHz



Middle product on a ev6-500MHz machine



Middle product on a sparcv9-296 MHz



Middle product on an R10K-194 MHz

6.2 Short Square

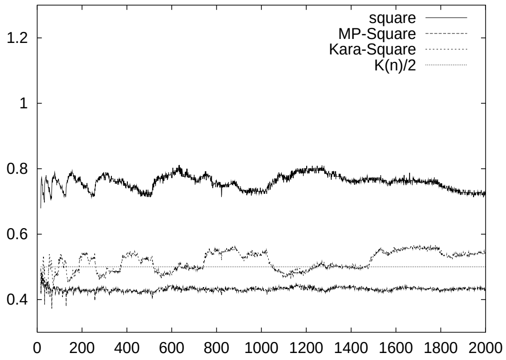
We have compared the `MP-square` algorithm, of theoretical complexity $\frac{1}{2}K(n)$ (see §2),[4] with the following algorithm:

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. $\alpha \leftarrow \texttt{mul}([a_0, \ldots, a_{n-p-1}], [a_0, \ldots, a_{n-p-1}])$;
3. $\beta \leftarrow \texttt{mul\_high}([a_0, \ldots, a_{n-p-1}], [a_{n-p}, \ldots, a_{n-1}])$;
4. Return $[\alpha_0, \ldots, \alpha_{n-p-1}, \alpha_{n-p} + 2\beta_0, \ldots, \alpha_{n-1} + 2\beta_{p-1}]$.

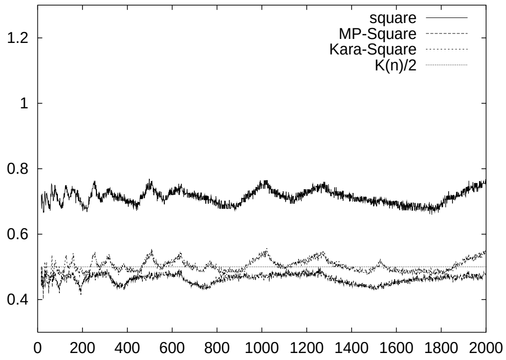
The theoretical complexity of this algorithm is $2M(\frac{n}{2})$, *i.e.*, $\frac{2}{3}K(n)$ with Karatsuba multiplication. However, a square is usually less expensive to compute than a product, and costs $\frac{2}{3}K(n)$ under reasonable assumptions. With the average gain due to Mulders' algorithm, we get an algorithm of practical complexity about $0.5K(n)$.

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

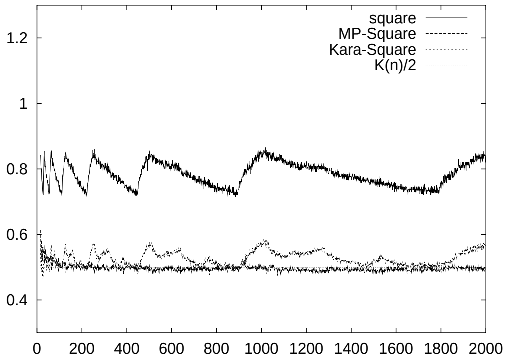
[4] A referee proposed the following Mulders-like trick to improve `MP-square`: take $p = \lfloor (1-\beta)n \rceil$ instead of $p = \lfloor n/2 \rfloor$, which leads to the recurrence $R(n) = R(\beta n) + K((1-\beta)n) + R((2\beta-1)n)$ under the Karatsuba model, with an optimal $\beta \approx 0.560$, and a theoretical average cost of about $0.481K(n)$. The same idea also applies to `Kara-square` with $p = \lfloor \beta n \rceil$, $\beta_{\mathrm{opt}} \approx 0.581$, and a cost of about $0.485K(n)$.



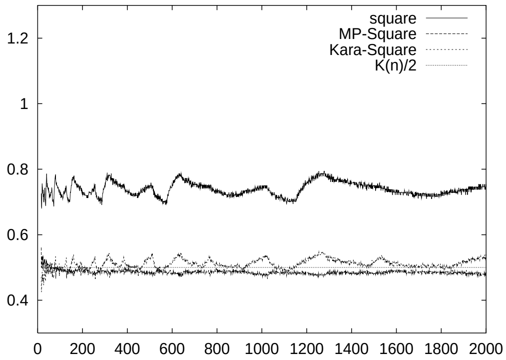
Short square on a k7-550MHz



Short square on a ev6-500MHz machine



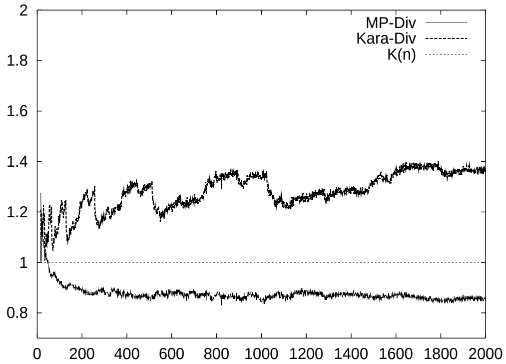
Short square on a sparcv9-296 MHz



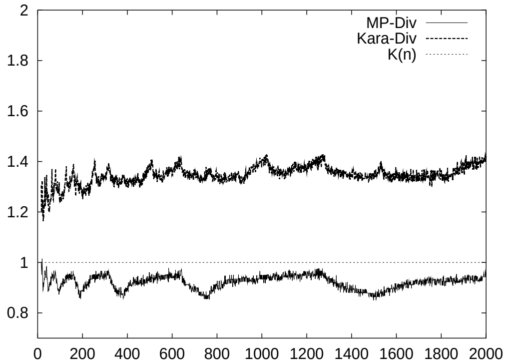
Short square on an R10K-194 MHz

6.3 Division

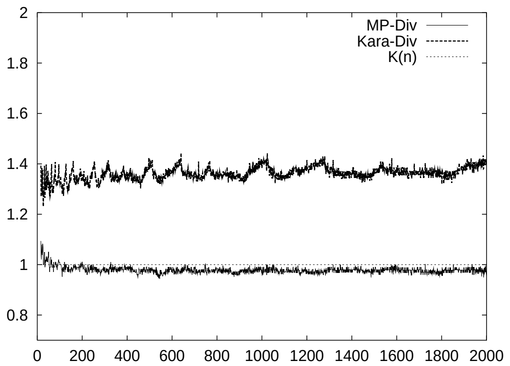
We have compared our `MP-divide` algorithm with Burnikel-Ziegler division [2], in which we have used short products to compute the remainder associated to the high part of the quotient. We get the following diagrams (lower curve is `MP-divide`):



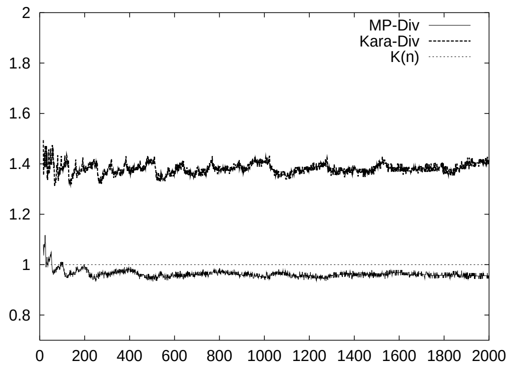
Division on a k7-550MHz



Division on a ev6-500MHz



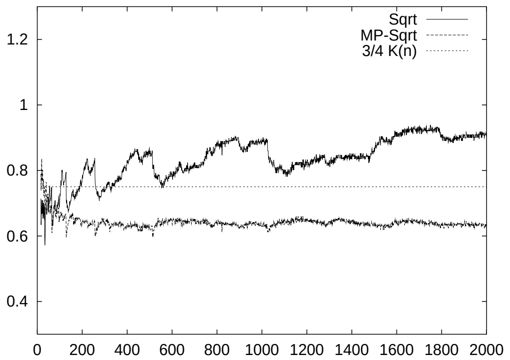
Division on a sparcv9-296 MHz



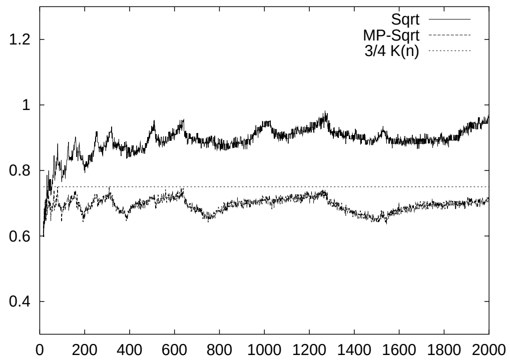
Division on an R10K-194 MHz

6.4 Square Root

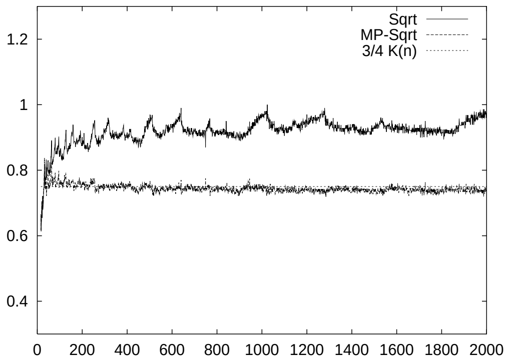
We have compared our `MP-Sqrt` algorithm (lower curve) with Karatsuba square root [13], in which we have used the `Kara-square` algorithm. We get the following diagrams:



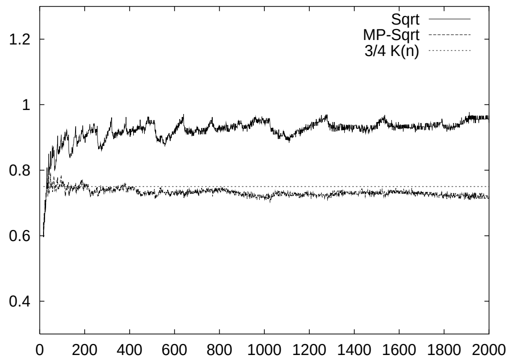
Square root on a k7-550MHz



Square root on a ev6-500MHz machine



Square root on a sparcv9-296 MHz



Square root on an R10K-194 MHz

6.5 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.

7 Conclusion

This paper presents new algorithms for basic operations on polynomials or power series. Those algorithms are based on a new algorithm, called *middle product*, that computes the n middle coefficients of a $(2n-1) \times n$ product as efficiently as a full $n \times n$ product. Table 1 gives a detailed analysis of the number of coefficient operations used by those algorithms, compared to previously known algorithms.[5]

	Karatsuba model				FFT model		
	previous		this paper		previous	this paper	
Square	$\frac{2}{3}K(n)$		$\frac{1}{2}K(n)$	[Th. 5]	$\frac{2}{3}M(n)$	$\frac{2}{3}M(n)$	
Middle product	$2K(n)$		$K(n)$	[Th. 2]	$2M(n)$	$M(n)$	[Th. 3]
Inverse	$\frac{3}{2}K(n)$	[2]	$K(n)$	[Th. 6]	$3M(n)$	$2M(n)$	[Th. 6]
Quotient	$\frac{3}{2}K(n)$	[2]	$K(n)$	[Th. 8]	$\frac{7}{2}M(n)$	$\frac{5}{2}M(n)$	[Th. 7]
... with remainder	$2K(n)$	[2]	$\frac{5}{3}K(n)$	[Th. 9]	$\frac{9}{2}M(n)$	$\frac{7}{2}M(n)$	[Th. 7]
Square root	$K(n)$	[13]	$\frac{3}{4}K(n)$	[Co. 1]	$\frac{7}{2}M(n)$	$3M(n)$	[Co. 1]
... with remainder	$\frac{3}{2}K(n)$	[13]	$\frac{5}{4}K(n)$	[Co. 1]	$\frac{9}{2}M(n)$	$4M(n)$	[Co. 1]

Table 1: Previous and new best-known complexities for several operations on power series and polynomials, under the Karatsuba and FFT models, within $O(n)$ memory usage.

An implementation in Maple of these algorithms is available at the URL `http://www.loria.fr/~zimmerma/papers/MP.mpl`.

It is possible to obtain floating point versions of the middle product algorithm using only O(1) extra memory — with respect to the polynomial version — 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:

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

[5]For the sake of simplicity, we consider that a full square costs as much as a full product, and we don't reuse Fourier transforms when a given variable is used in several multiplications.

This is exactly $\texttt{MP}([b_{n-1},\ldots,b_0],[v_0,\ldots,v_{2n-2}])$, 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 $0\le i-j<n$, 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 $\texttt{MP}([x_0,\ldots,x_{n-1}],[a_0,\ldots,a_{2n-2}])$, in reversed order.

The algorithms of Table 1 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, the *relaxed* DAC-multiplication proposed by van der Hoeven in [12] can be adapted to yield a division with remainder in *exactly* $K(n)$ scalar operations, using $O(n\log n)$ memory. However the "fast variant" from [12, §4.3.1], which works within $O(n)$ memory, requires more than $2K(n)$ operations asymptotically.

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, 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 (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 Symposium on Symbolic and Algebraic Computation* (Vancouver, Canada), S. Dooley, Ed., ACM, pp. 53–58.

[12] van der Hoeven, J. Relax, but don't be too lazy. *Journal of Symbolic Computation 34*, 6 (2002), 479–542.

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

Appendix. The General Situation

In this section, we give a more general and formal presentation of the trick described in §1. This presentation is suitable for any bilinear multiplication algorithm, and gives, in the case of Karatsuba and FFT multiplication, corresponding algorithms of the same complexity for the middle product (see Theorems 2 and 3).

Theorem 11 *Let $M_{p,q,n} : R^p \times R^q \to R^n$ and $\Pi_{n,p,q} : R^n \times R^p \to R^q$ be the bilinear forms defined by*

$$M_{p,q,n}(y,z) = \left(\sum_{j+k=i+p-1} y_j z_k \right)_{0 \le i < n} \qquad \Pi_{n,p,q}(x,y) = \left(\sum_{i+j=k} x_i y_j \right)_{0 \le k < q}.$$

Then, for any $(X,Y,Z) \in R^n \times R^p \times R^q$, 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 and $\tilde{Y} = (y_{p-1-i})_{0 \le i < p}$

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 $X = (x_0, \ldots, x_{n-1}) \in R^n$, $Y = (y_0, \ldots, y_{p-1}) \in R^p$ and $Z = (z_0, \ldots, z_{q-1}) \in R^q$. We have

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

For $p = n$ and $q = 2n-1$, $\Pi_{n,p,q}$ and $M_{p,q,n}$ correspond respectively to the usual multiplication and the middle product.

PROOF.[Theorem 11] We have

$$\begin{aligned}
(X \mid M_{p,q,n}(Y,Z)) &= \sum_{i=0}^{n} x_i \sum_{\substack{0\le j<p\\0\le k<q\\j+k=i+p-1}} y_j z_k = \sum_{\substack{0\le i<n\\0\le j<p\\0\le k<q\\i+p-1-j=k}} x_i y_j z_k = \sum_{\substack{0\le i<n\\0\le j<p\\0\le k<q\\i+j=k}} x_i y_{p-1-j} z_k \\
&= \sum_{0\le k<q} z_k \sum_{\substack{0\le i<n\\0\le j<p\\i+j=k}} x_i y_{p-1-j} = (\Pi_{n,p,q}(X,\tilde{Y}) \mid Z),
\end{aligned}$$

which concludes the proof. ∎

We are now able to prove our main result, Theorem 4:

PROOF OF THEOREM 4. 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 $a_m \in R^n$, $b_m \in R^p$ and $c_m \in R^q$. Such a formula will be called "a decomposition of $\Pi_{n,p,q}$ into ℓ ring multiplications". Then we have according to Theorem 11

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

Therefore, X being an arbitrary vector in R^n:

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

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

From a practical point of view, Theorem 11 has the following meaning. One way of seeing usual algorithms for computing products is based 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[6].

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:

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

[6] 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.

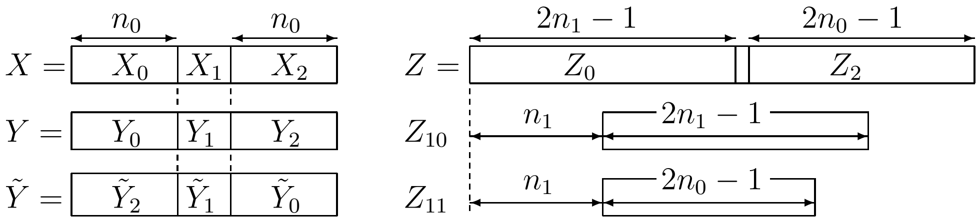


Figure 3: `Karatsuba` and `MP` splitting.

where $*$ denotes 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

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

This means, in the case $p = n$, $q = 2n - 1$, that from any multiplication bilinear algorithm using $M(n)$ multiplications in R we can derive an algorithm for `MP` using $M(n)$ multiplications in R, and $n - 1$ more additions.

We now illustrate how the algorithms of Theorems 2 and 3 can be obtained from Theorem 11.

The Karatsuba Model

Let $X = (x_i)_{0 \le i < n} \in R^n$, $Y = (y_j)_{0 \le j < n} \in R^n$ and $Z = (z_k)_{0 \le k < 2n-1} \in R^{2n-1}$. 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:

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

Karatsuba's method amounts to make a recursive call on low halves and high halves of the operands, and on the differences of those operands, *i.e.*, to compute

$$\begin{aligned} a &= \Pi_{n_1,n_1,2n_1-1}([X_0, X_1], [Y_0, Y_1]), \\ b &= \Pi_{n_0,n_0,2n_0-1}(X_2, Y_2), \\ c &= \Pi_{n_1,n_1,2n_1-1}([X_0 - X_2, X_1], [Y_0 - Y_2, Y_1]), \end{aligned}$$

and then the high part of the result is b, the low part is a and the middle part is $a-c+b$. In our formalism, we write

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

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

$$\begin{aligned}(X \mid M_{n,2n-1,n}(Y,Z)) &= (\Pi_{n,n,2n-1}(X,\tilde{Y}) \mid Z) \\ &= (\Pi_{n_1,n_1,2n_1-1}([X_0,X_1],[\tilde{Y}_2,\tilde{Y}_1]) \mid Z_0+Z_{10}) \\ &\quad - (\Pi_{n_1,n_1,2n_1-1}([X_0-X_2,X_1],[\tilde{Y}_2-\tilde{Y}_0,\tilde{Y}_1]) \mid Z_{10}) \\ &\quad + (\Pi_{n_0,n_0,2n_0-1}(X_2,\tilde{Y}_0) \mid Z_2+Z_{11}) \\ &= ([X_0,X_1] \mid M_{n_1,2n_1-1,n_1}([Y_1,Y_2],Z_0+Z_{10})) \\ &\quad - ([X_0-X_2,X_1] \mid M_{n_1,2n_1-1,n_1}([Y_1,Y_2-Y_0],Z_{10})) \\ &\quad + (X_2 \mid M_{n_0,2n_0-1,n_0}(Y_0,Z_2+Z_{11})) \\ &= (X \mid [\alpha-\beta,\gamma+\mathrm{low}(\beta)])\end{aligned}$$

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 R^n, we infer:

$$M_{n,2n-1,n}(Y,Z) = [\alpha-\beta,\gamma+\mathrm{low}(\beta)]. \tag{6}$$

This formula is exactly Algorithm `MP-Karatsuba` from §1.2.

The FFT Model

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

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

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

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

In more classical notations, if $f_m^{(2n)}$ is the m-th component of the direct Fourier transform, and $g_m^{(2n)}$ that of the inverse one, we get

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

which is exactly Algorithm `MP-FFT` from §1.3.

This means that $\mathtt{MP}(y,z)$ 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.

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

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

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 operation (assuming no particular optimization).