Relaxed Multiplication Using the Middle Product

Joris van der Hoeven
Département de Mathématiques (bât. 425)
Université Paris-Sud
91405 Orsay Cedex
France
joris@texmacs.org

ABSTRACT

In previous work, we have introduced the technique of relaxed power series computations. With this technique, it is possible to solve implicit equations almost as quickly as doing the operations which occur in the implicit equation. In this paper, we present a new relaxed multiplication algorithm for the resolution of linear equations. The algorithm has the same asymptotic time complexity as our previous algorithms, but we improve the space overhead in the divide and conquer model and the constant factor in the F.F.T. model.

Categories and Subject Descriptors

G. Mathematics of Computing [**G.1. Numerical Analysis**]: G.1.0. General—*Multiple Precision Arithmetic*

General Terms

Algorithms

Keywords

power series, relaxed multiplication, middle product

1. INTRODUCTION

Let $\mathcal{R}$ be an effective ring and consider two power series $f = f_0 + f_1 z + \cdots$ and $g = g_0 + g_1 z + \cdots$ in $\mathcal{R}[[z]]$. In this paper we will be concerned with the efficient computation of the first n coefficients of the product $h = fg = h_0 + h_1 z + \cdots$.

If the first n coefficients of f and g are known beforehand, then we may use any fast multiplication for polynomials in order to achieve this goal, such as divide and conquer multiplication [6, 7], which has a time complexity $K(n) = O(n^{\log 3/\log 2})$, or F.F.T. multiplication [2, 9, 1, 11], which has a time complexity $M(n) = O(n \log n \log \log n)$.

For simplicity, "time complexity" stands for the required number of operations in $\mathcal{R}$. Similarly, "space complexity" will stand for the number of elements of $\mathcal{R}$ which need to

be stored. The required number of multiplications $K_\times(n)$ in the divide and conquer algorithm satisfies the following recurrence relations:

$$
\begin{aligned}
K_\times(1) &= 1 \\
K_\times(n) &= 2K_\times(\lceil n/2 \rceil) + K_\times(\lfloor n/2 \rfloor)
\end{aligned}
$$

When performing computing only the product truncated at order n, then the number of multiplications $K_\times^*$ needed by the divide and conquer algorithm becomes

$$
\begin{aligned}
K_\times^*(1) &= 1 \\
K_\times^*(n) &= K_\times(\lceil n/2 \rceil) + 2K_\times^*(\lfloor n/2 \rfloor)
\end{aligned}
$$

For certain computations, and most importantly the resolution of implicit equations, it is interesting to have so called "relaxed algorithms" which output the first i coefficients of h as soon as the first i coefficients of f and g are known for each $i \leqslant n$. This allows for instance the computation of the exponential $g = \exp f$ of a series f with $f_0 = 0$ using the formula

$$
g = \int f'g. \tag{1}
$$

In [10, 11], we proved the following two theorems:

THEOREM 1. *There exists a relaxed multiplication algorithm of time complexity $K(n)$ and space complexity $O(\log n)$, and which uses $K_\times(n)$ multiplications.*

THEOREM 2. *There exists a relaxed multiplication algorithm of time complexity $O(M(n) \log n)$ and space complexity $O(n)$.*

Although these theorems are satisfactory from a theoretical point of view, they can be improved in two directions: by removing the logarithmic space overhead in the divide and conquer model and by improving the constant factor in the F.F.T. model.

In this paper, we will present such an improved algorithm in the case of relaxed multiplication with a fixed series. More precisely, let f and g be power series, such that g is known up to order n. Then our algorithm will compute the product $h = fg$ up to order n and output $(fg)_i$ as soon as $f_0, \ldots, f_i$ are known, for all $i < n$. We will prove the following:

THEOREM 3. *There exists a relaxed multiplication algorithm with fixed series of time complexity $O(K(n))$, of space complexity $O(n)$, and which uses $K_\times^*(n)$ multiplications.*
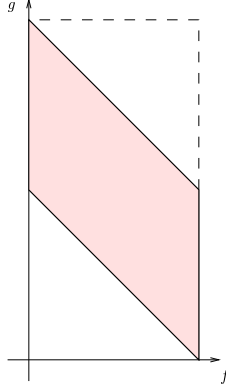
Figure 1: Illustration of the middle product.

Figure 2: The subdivision used for the new relaxed multiplication algorithm.

We also obtain a better constant factor in the asymptotic complexity in the F.F.T. model, but this result is harder to state in a precise theorem.

The algorithm is useful for the relaxed resolution of linear differential or difference equations. For instance, the exponential of a series can be computed using $\leqslant K_\times^*(n)$ multiplications in $\mathcal{R}$. Moreover, the new algorithm is very simple to implement, so it is likely to require less overhead than the algorithm from theorem 1.

Our algorithm is based on the recent middle product algorithm [3, 4], which is recalled in section 2. In section 3 we present our new algorithm and in section 5 we give some applications.

In our algorithms we will use the following notations: the data type $\mathsf{TPS}(n)$ stands for truncated power series of order n, like $f = f_0 + \cdots + f_{n-1}z^{n-1}$. Given $f \in \mathsf{TPS}(n)$ and $0 \leqslant i < j \leqslant n$, we will denote $f_{i\ldots j} = f_i + \cdots + f_{j-1}z^{j-i-1} \in \mathsf{TPS}(j-i)$. Given $f \in \mathsf{TPS}(m)$ and $g \in \mathsf{TPS}(n)$, we also denote $f \bowtie g = f + gz^m \in \mathsf{TPS}(m+n)$. We will denote by $\mathsf{Ref}(\mathsf{TPS}(n))$ the type, whose elements are references to elements of type $\mathsf{TPS}(n)$. If $f \in \mathsf{TPS}(n)$ and $0 \leqslant i < j \leqslant n$, then we assume that $f_{i\ldots j} \in \mathsf{Ref}(\mathsf{TPS}(n))$.

2. THE MIDDLE PRODUCT

Let $f = f_0 + \cdots + f_{n-1}z^{n-1}$ and $g = g_0 + \cdots + g_{2n-2}z^{2n-2}$ be two truncated power series at orders n resp. $2n-1$. The *middle product* f and g is defined to be the truncated power series $h = f * g = h_0 + \cdots + h_{n-1}z^{n-1}$ of order n, such that $h_i = \sum_{j=0}^{n-1} f_j g_{n-1+i-j}$ for all $i \in \{0, \ldots, n-1\}$. In figure 1, h corresponds to the colored region.

The middle product of $f = f_0 + f_1 z$ and $g = g_0 + g_1 z + g_2 z^2$ can be computed using only three multiplications, using the following trick:

$$
\begin{aligned}
\alpha &= f_1(g_0 + g_1) \\
\beta &= (f_1 - f_0)g_1 \\
\gamma &= f_0(g_1 + g_2) \\
h_0 &= \alpha - \beta \\
h_1 &= \gamma + \beta
\end{aligned}
$$

This trick may be applied recursively in order to yield an algorithm which needs exactly the same number of multiplications $K_\times(n)$ as the divide and conquer algorithm 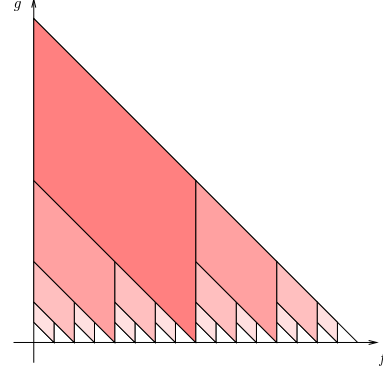for the computation of the product of two polynomials of de-gree $n-1$. More precisely, the following recursive algorithm comes from [3, 4].

Algorithm $f * g$
INPUT: $f \in \mathsf{TPS}(n)$ and $g \in \mathsf{TPS}(2n-1)$
OUTPUT: their middle product $f * g \in \mathsf{TPS}(n)$

if $n = 1$ **then return** $f_0 g_0$
$k := \lfloor n/2 \rfloor, l := \lceil n/2 \rceil$
$\alpha := f_{k\ldots n} * (g_{0\ldots 2l-1} + g_{l\ldots 3l-1})$
if n is even
    then $\beta := (f_{l\ldots n} - f_{0\ldots k}) * g_{l\ldots 3l-1}$
    else $\beta := [f_k \bowtie (f_{l\ldots n} - f_{0\ldots k})] * g_{l\ldots 3l-1}$
$\gamma := f_{0\ldots k} * (g_{l\ldots l+2k-1} + g_{2l\ldots 2n-1})$
return $(\alpha - \beta) \bowtie (\gamma + \beta_{0\ldots k})$

In [4] it is also shown that, in the F.F.T. model, the middle product can still be computed in essentially the same time as the product of two polynomials.

3. RELAXED MULTIPLICATION WITH A FIXED SERIES

Let f and g be power series, such that g is known up to order n. In this section, we present an algorithm which computes the product $h = fg$ up to order n. For each $i < n$, the algorithm outputs $(fg)_i$ as soon as $f_0, \ldots, f_i$ are known.

The idea of our algorithm is similar to the idea behind fast relaxed multiplication in [10, 11] and based on a subdivision of the triangular area which corresponds to the computation of the truncated power series product. This subdivision is shown in figures 2 and 3, where each parallelogram corresponds to the computation of a middle product.

More precisely, let $l = \lceil n/2 \rceil$ and assume that $f_0, \ldots, f_{l-1}$ are known. Then the contribution of $f_{0\ldots l} * g_{n+1-2l\ldots n}$ to fg may be computed using the middle product algorithm from the previous section. The relaxed truncated products $f_{0\ldots k}g_{0\ldots k}$ and $f_{l\ldots n}g_{0\ldots k}$ may be computed recursively.

In order to implement this idea, we will use an in-place algorithm, which adds the result of $h = fg$ to a reference φ to an element of $\mathsf{TPS}(n)$. Denote by φ_{init} the initial value of φ. Then the in-place algorithm should be called successively for $i = 1, \ldots, n$. After the last call, we have $\varphi = \varphi_{\mathrm{init}} + h$. Taking $\varphi_{\mathrm{init}} = 0$, the algorithm computes h.

Algorithm relaxed-muladd(f, g, φ, i)
INPUT: $f, g \in \mathsf{TPS}(n)$, $\varphi \in \mathsf{Ref}(\mathsf{TPS}(n))$, $i \leqslant n$.
ACTION: we have $\varphi_{0\ldots i} = \varphi_{\mathrm{init}} + f_{0\ldots i}g_{0\ldots i}$ on exit.

if $i = n = 1$ **then** $\varphi_0 \mathrel{+}= f_0 g_0$ and **return**
$k := \lfloor n/2 \rfloor, l := \lceil n/2 \rceil$
if $i \leqslant k$ **then** relaxed-muladd$(f_{0\ldots k}, g_{0\ldots k}, \varphi_{0\ldots k}, i)$
if $i = k + 1$ **then** $\varphi_{k\ldots n} \mathrel{+}= f_{0\ldots l} * g_{n+1-2l\ldots n}$
if $i > l$ **then** relaxed-muladd$(f_{l\ldots n}, g_{0\ldots k}, \varphi_{l\ldots n}, i - l)$

The number of multiplications $R_\times(n)$ used by relaxed-muladd is determined by the relations

$$
\begin{aligned}
R_\times(1) &= 1 \; ; \\
R_\times(n) &= K_\times(\lceil n/2 \rceil) + 2R_\times(\lfloor n/2 \rfloor).
\end{aligned}
$$

By induction, it follows that $R_\times(n) = K_\times^*(n)$. The overall time complexity satisfies

$$R(n) \leqslant K(\lceil n/2 \rceil) + 2R(\lfloor n/2 \rfloor) + O(n),$$

so $R(n) = O(K(n))$. The algorithm being in-place, its space complexity is clearly $O(n)$. This proves theorem 3.

In it is also interesting to use the above algorithm in the F.F.T. model. We then have the estimation

$$R(n) \leqslant M(\lceil n/2 \rceil) + 2R(\lfloor n/2 \rfloor) + O(n)$$

for the asymptotic complexity $R(n)$. If

$$M(n) \sim cn \log n \log \log n,$$

this yields

$$R(n) \sim \frac{1}{2} M(n) \log_2 n.$$

This should be compared with the complexity

$$R(n) \sim M(n) \log_2 n$$

of the previously best algorithm and with the complexity $L(n) \sim 2M(n) \log_2 n$ of the standard fast relaxed multiplication algorithm.

Notice that we rarely obtain the complexity

$$M(n) \sim cn \log n \log \log n$$

in practice. In the range where $M(n) \sim cn^\alpha$, we obtain

$$R(n) \sim \frac{1}{2^\alpha - 2} M(n).$$

4. A WORKED EXAMPLE

Let us consider the computation of $f = \mathrm{e}^{z/(1-z)}$ up till order $7 = n + 1$ using our algorithm and the formula

$$f = \int fg,$$

with $f_0 = 1$ and $g = 1 + 2z + 3z^2 + \cdots$. We start with $\varphi = 0$ in relaxed-muladd and perform the following computations at successive calls for $i = 1, \ldots, 6$:

1. We set $\varphi_0 \mathrel{+}= f_0 g_0 = 1$, so that

$$\varphi := 1$$

   and $f_1 = 1$.

2. We recursively apply relaxed-muladd to $f_{0\ldots 3}$, $g_{0\ldots 3}$, $\varphi_{1\ldots 3}$ and $i = 2$. This requires the computation of
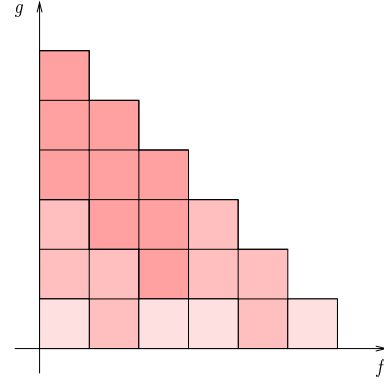

Figure 3: Illustration of an order 6 relaxed multiplication.

$f_{0\ldots 2} * g_{0\ldots 3} = (1 + z) * (1 + 2z + 3z^2) = 3 + 5z$. We thus increase $\varphi_{1\ldots 3} \mathrel{+}= 3 + 5z$, so that

$$\varphi := 1 + 3z + 5z^2$$

   and $f_2 = \frac{3}{2}$.

3. The two nested recursive calls to relaxed-muladd now lead to the increase of φ_2 by $f_2 g_0 = \frac{3}{2}$, so that

$$\varphi := 1 + 3z + \frac{13}{2}z^2$$

   and $f_3 = \frac{13}{6}$.

4. We now both have $i = k + 1 = 4$ and $i > l = 3$. So we first compute $f_{0\ldots 3} * g_{1\ldots 6} = 10 + \frac{27}{2}z + 17z^2$ and set $\varphi_{3\ldots 6} \mathrel{+}= 10 + \frac{27}{2}z + 17z^2$. We next recursively apply relaxed-muladd to $f_{3\ldots 6}$, $g_{0\ldots 3}$, $\varphi_{3\ldots 6}$ and $i = 1$, which leads to an increase of φ_3 by $f_3 g_0 = \frac{13}{6}$. Alltogether, we obtain

$$\varphi := 1 + 3z + \frac{13}{2}z^2 + \frac{73}{6}z^3 + \frac{27}{2}z^4 + 17z^5$$

   and $f_4 = \frac{73}{24}$.

5. We recursively apply relaxed-muladd to $f_{3\ldots 6}$, $g_{0\ldots 3}$, $\varphi_{3\ldots 6}$ and $i = 2$. This leads to the increase $\varphi_{4\ldots 6} \mathrel{+}= f_{3\ldots 5} * g_{0\ldots 3} = \frac{59}{8} + \frac{151}{12}z$, so that

$$\varphi := 1 + 3z + \frac{13}{2}z^2 + \frac{73}{6}z^3 + \frac{167}{8}z^4 + \frac{355}{12}z^5$$

   and $f_5 = \frac{167}{40}$.

6. The two nested recursive calls lead to the increase $\varphi_5 \mathrel{+}= f_5 g_0 = \frac{167}{40}$, so that

$$\varphi := 1 + 3z + \frac{13}{2}z^2 + \frac{73}{6}z^3 + \frac{167}{8}z^4 + \frac{4051}{120}z^5$$

   and $f_6 = \frac{4051}{720}$.

The entire computation is represented schematically in figure 3.

5. APPLICATIONS

First of all, let us consider the problem of relaxed division by a fixed power series. In other words, we are given two power series f and g, where g is known up to order n and $g_0 = 1$. We want an algorithm for the computation of $h = f/g$ up to order n, such that h_i is computed as soon as $f_0, \ldots, f_i$ are known for each $i < n$. Now we have

$$h = f - z(\varphi h),$$

where $\varphi = (g-1)/z \in \mathcal{R}[[z]]$. We may thus compute h in a relaxed way using the algorithm from the previous section. Computing h up till n terms will then necessitate $\leqslant K_\times(n)$ multiplications in $\mathcal{R}$.

Let us next consider a linear differential equation

$$L_r f^{(r)} + \cdots + L_0 f = 0, \qquad (2)$$

with $L_0, \ldots, L_r \in \mathcal{R}[[z]]$ and $L_r(0) = 1$. Given initial conditions for $f_0, \ldots, f_{r-1}$, there exists a unique solution to this equation. We may compute this solution using the relaxed algorithm from the previous section, the above algorithm for relaxed division, and the formula

$$f = L_r^{-1} \int \overset{r \times}{\cdots} \int (L_r f^{(r)} + \cdots + L_0 f).$$

In order to compute n coefficients, we need to perform $(r+1)K_\times(n)$ multiplications in $\mathcal{R}$ and $O(n)$ multiplications and divisions by integers. If $L_r = 1$, then we only need $rK_\times(n)$ multiplications.

For instance, the exponential g of a series f with $f_0 = 0$ satisfies the equation

$$g' - f'g = 0,$$

so g can be computed using $K_\times(n)$ multiplications, using the formula (1).

More generally, consider the solution to (2) with the prescribed initial conditions, and let g be another series with $g_0 = 0$. Then the composition $h = f \circ g$ again satisfies a linear differential equation. Indeed, we have the relations

$$
\begin{aligned}
f \circ g &= h \\
f' \circ g &= \frac{h}{g'} \\
f'' \circ g &= \frac{h''}{g'^2} - \frac{h'g''}{g'^3} \\
&\;\;\vdots
\end{aligned}
$$

Postcomposing (2) with g and using these relations, we obtain a linear differential equation for h.

In fact, our algorithm may be used to solve far more general linear equations, such as linear partial differential equations, or linear differential-difference equations. In the case of difference equations, we notice that the relaxed multiplications in the algorithms from [11] for relaxed right composition with a fixed series all have one fixed argument. So we may indeed apply the algorithm from section 3.

We finally notice that our algorithm can even be used in a non-linear context. Indeed, after computing $\lceil n/2 \rceil$ coefficients of a truncated relaxed product, the computation of the remaining products reduces to the computation of two truncated relaxed products with one fixed argument. Actually, this corresponds to an implicit application of Newton's method.
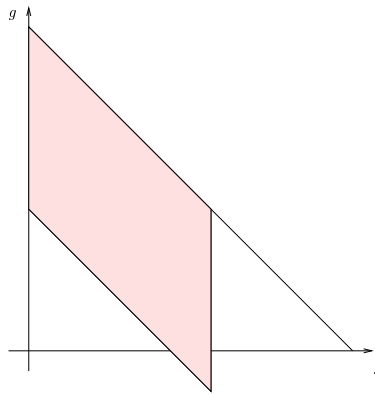

Figure 4: Using Mulders' trick in combination with the middle product.

6. CONCLUSION AND OPEN QUESTIONS

We have presented a new algorithm for relaxed multiplication. Although the new algorithm does not yield a significant improvement from the asymptotic complexity point of view, we do expect it to be very useful for practical applications, such as the exponentiation of power series.

First of all, the algorithm is easy to implement. Secondly, it only needs a linear amount of memory in the range where divide and conquer multiplication is appropriate. In combination with F.F.T. multiplication, the algorithm yields a better constant factor in the asymptotic complexity.

When implementing a library for power series computations, it is interesting to incorporate a mechanism to automatically detect relaxed and fixed multiplicands in a complex computation. This is possible by examining the dependency graph. With such a mechanism, one may use the new algorithm whenever possible.

Some interesting questions remain open in the divide and conquer model: can we apply Mulders' trick [8, 5] for the computation of "short products" in our setting while maintaining the linear space complexity (see figure 4)? In that case, we might improve the number of multiplications in theorem 3 to $\sim 0.808 \cdots K(n)$.

In a similar vein, does there exist a relaxed multiplication algorithm of time complexity $\leqslant K(n)$ and linear space complexity? This would be so, if the middle product algorithm could be made relaxed in an in-place way (the algorithm is already "essentially relaxed" in the sense of [10, 11] in the divide and conquer model).

As it stands now, with the above questions still unanswered, the original relaxed multiplication algorithm from theorem 1 remains best from the time complexity point of view in the divide and conquer model. Moreover, Mulders' trick can be applied in this setting, so as to yield a short relaxed multiplication algorithm of complexity $\sim 0.808 \cdots K(n)$, or even better [5].

This has surprising consequences for the complexities of several operations like short division and square roots: we obtain algorithms of time complexities $\sim 0.808 \cdots K(n)$ and $\sim \frac{1}{2}K(n)$ when using $O(n \log n)$ space, while the best known algorithms which use linear space have time complexities $\sim K(n)$ and $\sim \frac{3}{4}K(n)$. In order to obtain the complexity of $\sim \frac{1}{2}K(n)$ in the case of square roots, one should use

a relaxed version of the fast squaring algorithm from [4], which is based on middle products.

We finally remark that this relaxed version of squaring using middle products is also interesting in the F.F.T. model. In this case, the relaxed middle product corresponds to a full relaxed product with one fixed argument. Such products can be computed in time $\sim 2R(n)$, so that we obtain a relaxed squaring algorithm of time complexity $\sim 2R(n)$. This is twice as good as general relaxed multiplication. In the non-relaxed setting, squares can be computed in a time between $\frac{1}{2}M(n)$ and $\frac{2}{3}M(n)$, depending on whether most time is spent on inner multiplications or fast Fourier transforms respectively.

7. REFERENCES

[1] Cantor, D., and Kaltofen, E. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica 28* (1991), 693–701.

[2] Cooley, J., and Tukey, J. An algorithm for the machine calculation of complex Fourier series. *Math. Computat. 19* (1965), 297–301.

[3] Hanrot, G., Quercia, M., and Zimmermann, P. Speeding up the division and square root of power series. Research Report 3973, INRIA, July 2000. Available from `http://www.inria.fr/RRRT/RR-3973.html`.

[4] Hanrot, G., Quercia, M., and Zimmermann, P. The middle product algorithm I. speeding up the division and square root of power series. Submitted, 2002.

[5] Hanrot, G., and Zimmermann, P. A long note on mulders' short product. Research Report 4654, INRIA, Dec. 2002. Available from `http://www.loria.fr/~hanrot/Papers/mulders.ps`.

[6] Karatsuba, A., and Ofman, J. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady 7* (1963), 595–596.

[7] Knuth, D. *The Art of Computer Programming*, 3-rd ed., vol. 2: Seminumerical Algorithms. Addison-Wesley, 1997.

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

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

[10] van der Hoeven, J. Lazy multiplication of formal power series. In *Proc. ISSAC '97* (Maui, Hawaii, July 1997), W. W. Küchlin, Ed., pp. 17–20.

[11] van der Hoeven, J. Relax, but don't be too lazy. *JSC 34* (2002), 479–542.