A simple and fast online power series multiplication and its analysis

Romain Lebreton, Éric Schost

A simple and fast online power series multiplication and its analysis*

Romain Lebreton
LIRMM
UMR 5506 CNRS
Université Montpellier II
Montpellier, France
Email: lebreton@lirmm.fr

Éric Schost
Computer Science Department
Western University
London, Ontario
Canada
Email: eschost@uwo.ca

Abstract

This paper focuses on *online* (or *relaxed*) algorithms for the multiplication of power series over a field and their complexity analysis. We propose a new online algorithm for the multiplication using middle and short products of polynomials as building blocks, and we give the first precise analysis of the arithmetic complexity of various online multiplications. Our algorithm is faster than Fischer and Stockmeyer's by a constant factor; this is confirmed by experimental results.

Keywords: Online algorithm, relaxed algorithm, multiplication of power series, arithmetic complexity

1 Introduction

Let $\mathbb{A}$ be a commutative ring with unity, and let x be an indeterminate over $\mathbb{A}$ (the commutativity hypothesis may not be required but greatly simplifies the setting of this paper). Given two power series $a = \sum_{i \geqslant 0} a_i\, x^i$ and $b = \sum_{i \geqslant 0} b_i\, x^i$ in $\mathbb{A}[[x]]$, we are interested in computing the coefficients c_i of the product $c = a\, b$ under the following constraint: we cannot use the coefficients a_i or b_i before we have computed $c_0, ..., c_{i-1}$. This condition is useful to model situations where the inputs a, b and the output c are related by a feedback loop, *i.e.* where $c_0, ..., c_{i-1}$ are needed in order to determine a_i and b_i (see the discussion below).

Previous work. Algorithms that satisfy such a constraint were introduced by Fischer and Stockmeyer in (Fischer and Stockmeyer, 1974); following that reference, we will call them *online* (the notion of an online algorithm extends beyond this question of power series multiplication, see for instance (Hennie, 1966)). Still following Fischer and Stockmeyer, we will also consider *half-line* multiplication, where one of the arguments, say b, is assumed to be known in advance at arbitrary precision; in other words, the only constraint for such algorithms is that we cannot use the coefficient a_i before we have computed $c_0, ..., c_{i-1}$.

*. This work has been partly supported by the ANR grant HPAC (ANR-11-BS02-013), NSERC and the CRC program.

It seems that few applications of online power series multiplication were given at the time (Fischer and Stockmeyer, 1974) was written. Recently, van der Hoeven rediscovered Fischer and Stockmeyer's half-line and online multiplication algorithms, which he respectively called *semi-relaxed* and *relaxed* (van der Hoeven, 1997; van der Hoeven, 2002). In addition, as alluded to above, he showed that online multiplication is the key to computing power series solutions of large families of differential equations or of more general functional equations; this result was extended in (Berthomieu and Lebreton, 2012) to further families of linear and polynomial equations, showing the fundamental importance of online multiplication.

We complete this brief review of online multiplication by mentioning its adaptation to real numbers in (Schröder, 1997) and its extension to the multiplication of p-adic integers in (Berthomieu et al., 2011).

The results of the papers (Fischer and Stockmeyer, 1974; Schröder, 1997; van der Hoeven, 1997; van der Hoeven, 2002; Berthomieu et al., 2011) can be summarized by saying that online multiplication is slower than "classical" multiplication by at most a logarithmic factor. More precisely, let us denote by $\mathsf{M}(n)$ a function such that polynomials of degree at most $n-1$ in $\mathbb{A}[x]$ can be multiplied in $\mathsf{M}(n)$ ring operations in $\mathbb{A}$. For instance, using the naive algorithm gives $\mathsf{M}(n) = \mathcal{O}(n^2)$, Karatsuba's algorithm gives $\mathsf{M}(n) = \mathcal{O}(n^{\log_2(3)})$ and Fast Fourier Transform (FFT) techniques allow us to take $\mathsf{M}(n)$ quasi-linear: in the presence of roots of unity in $\mathbb{A}$ of orders 2^ℓ for any $\ell \geqslant 0$, FFT gives $\mathsf{M}(n) = 9 \cdot 2^\ell\, \ell + \mathcal{O}(2^\ell)$ with $\ell = \lceil \log_2(n) \rceil$ (hence the behavior of a "staircase" function, see (von zur Gathen and Gerhard, 2003, Chapter 8.2)).

Then, the results in (Fischer and Stockmeyer, 1974) and (van der Hoeven, 1997; van der Hoeven, 2002) show that half-line multiplication to precision n, *i.e.* with input and output modulo x^n, can be done in time

$$\mathsf{H}(n) = \mathcal{O}\left(\sum_{k=0}^{\lfloor \log_2(n) \rfloor} \frac{n}{2^k}\, \mathsf{M}(2^k) \right)$$

and that online multiplication to precision n can be done in time $\mathsf{O}(n) = \mathcal{O}(\mathsf{H}(n))$. In all cases, if $\mathsf{M}(n)/n$ is increasing, $\mathsf{H}(n)$ is $\mathcal{O}(\mathsf{M}(n)\log(n))$, since all terms in the sum are bounded from above by $\mathsf{M}(n)$; for naive or Karatsuba's multiplication, $\mathsf{H}(n)$ is actually $\mathcal{O}(\mathsf{M}(n))$. The algorithm introduced by van der Hoeven in (van der Hoeven, 2003) for half-line multiplication improves on the one reported above by a constant factor.

Recent progress has been made on online multiplication (van der Hoeven, 2007; van der Hoeven, 2014): these papers give an online algorithm that multiplies power series on a wide range of rings in time $\mathsf{M}(n)\log(n)^{o(1)}$, which improves on the costs given here. However, this algorithm is significantly more complex; we believe that there is still an interest in developing simpler and reasonably fast algorithms, such as the one given here.

Our contribution. In this paper, we introduce a simple and fast algorithm for online multiplication, based on the ideas from (van der Hoeven, 2003). We compare it to previous algorithms by giving the first precise analysis of the arithmetic complexity of the various online and half-line multiplication algorithms mentioned up to now. For this complexity measure, our algorithm is faster than Fischer and Stockmeyer's by a constant factor; this is confirmed by experimental results. This paper is based on the PhD. thesis (Lebreton,

2012). To the best of our knowledge, the complexity estimates of Tables 1 and 2 are published for the first time.

Polynomial multiplication algorithms. For the rest of this paper, we will consider the *arithmetic cost* of our algorithms, that is the number of additions and multiplications in $\mathbb{A}$ they perform. The algorithms in this paper rely on two variants of polynomial multiplication, called middle and short products. In order to describe them, we introduce the following notation, used in all that follows: if $a = \sum_i a_i x^i$ is in $\mathbb{A}[x]$ or $\mathbb{A}[[x]]$, and n, m are integers with $m \geqslant n$, then we write

$$a_{n\ldots m} = a_n + a_{n+1} x + \cdots + a_{m-1} x^{m-n-1},$$

so that $a_{n\ldots m}$ has degree less than $m-n$. If $a, b \in \mathbb{A}[x]$, we denote respectively by $a \bmod b$ and $a \operatorname{div} b$ the remainder and quotient of the Euclidean division of a by b.

Let $a, b \in \mathbb{A}[x]$ with b of degree less than n. Then, the middle product $\mathrm{MP}(a, b, n)$ of a and b is defined as the part $c_{n-1\ldots 2n-1}$ of the product $c := a\,b$, so that $\deg(\mathrm{MP}(a, b, n)) < n$. Naively, the middle product is computed via the full multiplication $c := (a\ b \mod x^{2n-1}) \operatorname{div} x^{n-1}$, which is done in time $2\,\mathsf{M}(n) + \mathcal{O}(n)$, but this is not optimal. Indeed, the middle product is closely related to the *transposed multiplication* (Bostan et al., 2003; Hanrot et al., 2004); precisely, it is a transposed multiplication, up to the reversal of polynomial b; we deduce using for instance a general theorem in (Bürgisser et al., 1997), or the algorithms in (Bostan et al., 2003; Hanrot et al., 2004), that the arithmetic cost MP of the middle product $\mathrm{MP}(a, b, n)$ satisfies

$$\mathsf{MP}(n) = \mathsf{M}(n) + \mathcal{O}(n).$$

Let now $a, b \in \mathbb{A}[x]$ be both of degree less than n. The *low short product*, or just *short product*, of a and b is denoted by $\mathrm{SP}(a, b, n) := (a\ b) \bmod x^n$. Its variant, the *high short product* of a and b is denoted by $\mathrm{HP}(a, b, n) := (a\ b) \operatorname{div} x^{n-1}$. The two operations are closely related since $\mathrm{HP}(a, b, n) = \mathrm{rev}_n(\mathrm{SP}(\mathrm{rev}_n(a), \mathrm{rev}_n(b), n))$ where $\mathrm{rev}_n(a) := x^{n-1}\, a(1/x)$ denotes the reversal of length n of the polynomial a of degree less than n. Therefore, these two short products have the same arithmetic cost.

We denote by $\mathsf{SP}(n)$ the arithmetic cost of the short product at precision n, and by C_{SP} a constant such that $\mathsf{SP}(n) \leqslant C_{\mathsf{SP}}\,\mathsf{M}(n) + \mathcal{O}(n)$ holds for all $n \in \mathbb{N}^*$. Of course, we can always assume $C_{\mathsf{SP}} \leqslant 1$, but the actual cost of the short product is hard to pin down: although the size of the output is halved, we seldom gain a factor 2 in the cost.

As always, it is easy to adapt the naive multiplication algorithm to compute only the first terms; in this case, we gain a factor two in the cost, *i.e.* we can take $C_{\mathsf{SP}} = 1/2$. The paper (Mulders, 2000) published the first approach for having $C_{\mathsf{SP}} < 1$ for the cost function $\mathsf{M}(n) = n^{\log_2(3)}$, which is an approximation of the cost of Karatsuba's multiplication, giving $C_{\mathsf{SP}} = 0.81$; however, taking for $\mathsf{M}(n)$ the *exact* arithmetic cost of Karatsuba's multiplication, the best known upper bound remains $C_{\mathsf{SP}} = 1$ (Hanrot and Zimmermann, 2004). For an hybrid multiplication algorithm that uses the naive algorithm for small values and switches to Karatsuba's method for larger values, the situation is better: for a threshold $n_0 = 32$, the bound $\mathsf{SP}^*(n) \leqslant 0.57\ \mathsf{M}^*(n)$ is proved in (Hanrot and Zimmermann, 2004) for multiplicative complexity; it is beyond the scope of this paper to prove that this bound remains valid for arithmetic complexity (for the implementation of (Hanrot and Zimmermann, 2004), $\mathsf{SP}(n) \leqslant 0.6\,\mathsf{M}(n)$ is a realistic practical bound).

	half-line - $\mathsf{H}_{\mathtt{FS}}$	half-line with middle product - $\mathsf{H}_{\mathtt{vdH}}$
naive	$\mathsf{H}_{\mathtt{FS}}(n) \leqslant 2\,\mathsf{M}(n) + \mathcal{O}(n\log(n))$	$\mathsf{H}_{\mathtt{vdH}}(n) \leqslant 1.5\,\mathsf{M}(n) + \mathcal{O}(n\log(n))$ *
Karatsuba	$\mathsf{H}_{\mathtt{FS}}(n) \leqslant 3\,\mathsf{M}(n) + \mathcal{O}(n\log(n))$	$\mathsf{H}_{\mathtt{vdH}}(n) \leqslant 2\,\mathsf{M}(n) + \mathcal{O}(n\log(n))$ *
FFT	$\mathsf{H}_{\mathtt{FS}}(n) \sim \frac{1}{2}\,9\,n\log_2(n)^2$	$\mathsf{H}_{\mathtt{vdH}}(n) \sim \frac{1}{4}\,9\,n\log_2(n)^2$ *

Table 1. Complexity of half-line multiplication

	online - $\mathsf{O}_{\mathtt{FS}}$	online with short/middle products - $\mathsf{O}_{\mathtt{LS}}$
naive	$\mathsf{O}_{\mathtt{FS}}(n) \leqslant \mathsf{M}(n+1) + \mathcal{O}(n\log(n))$	$\mathsf{O}_{\mathtt{LS}}(n) \leqslant \mathsf{M}(n+1) + \mathcal{O}(n\log(n))$
Karatsuba	$\mathsf{O}_{\mathtt{FS}}(n) \leqslant 2.5\,\mathsf{M}(n+1) + \mathcal{O}(n\log(n))$	$\mathsf{O}_{\mathtt{LS}}(n) \leqslant \frac{3\,C_{\mathsf{SP}}+2}{2}\,\mathsf{M}(n+1) + \mathcal{O}(n\log(n))$
FFT	$\mathsf{O}_{\mathtt{FS}}(n) \sim 9\,n\log_2(n)^2$	$\mathsf{O}_{\mathtt{LS}}(n) \sim \frac{1}{2}\,9\,n\log_2(n)^2$

Table 2. Complexity of online multiplication

No improvement is known for the short product based on FFT multiplication. This does not matter for our purposes since the overall contribution of short products will turn out to be negligible when we use FFT multiplication.

Our complexity results. Table 1 gives bounds on the arithmetic complexity of *half-line* multiplication algorithms depending on the algorithm we use to multiply truncated power series (naive, Karatsuba or FFT). We will often use the notation $f(n) \leqslant g(n) + \mathcal{O}(h(n))$ in our complexity statements for functions $f, g, h\colon \mathbb{N} \to \mathbb{N}^*$ such that there exists $D \in \mathbb{R}_{>0}$ such that for all $n \in \mathbb{N}$, $f(n) \leqslant g(n) + D\,h(n)$.

Table 1 sums up the results of Corollary 14 and Proposition 10 (the asterisk in some cells point to Remark 2). The half-line multiplication algorithm which appears in (Fischer and Stockmeyer, 1974) gives the costs of the first column; we give an overview of this algorithm in Section 2.1. The second column corresponds to the half-line algorithm using middle product presented in (van der Hoeven, 2003), which can be found in Section 2.2.

Remark in particular that the cost of half-line algorithms using FFT polynomial multiplication involves the function $9\,n\log_2(n)$, which is a smoothed version of the "staircase" cost function of the FFT mentioned above.

Table 2 describes online algorithms. The first column of Table 2 corresponds to the online multiplication algorithm of (Fischer and Stockmeyer, 1974; van der Hoeven, 1997; Berthomieu et al., 2011), which is presented in Section 2.3. Our contribution, the online multiplication using middle and short products, gives the results of the second column and is presented in Section 2.4. These complexity results are proved in Propositions 15, 16 and 10.

The factor before $\mathsf{M}(n+1)$ appearing for $\mathsf{O}_{\mathtt{LS}}$ with Karatsuba's algorithm lies between 1.75 for $C_{\mathsf{SP}} = 0.5$ and 2.5 for $C_{\mathsf{SP}} = 1$. In practice, if we expect a behavior close to $C_{\mathsf{SP}} = 0.6$ as in (Hanrot and Zimmermann, 2004), we obtain a bound $\mathsf{O}_{\mathtt{LS}}(n) \leqslant 1.9\ \mathsf{M}(n+1) + \mathcal{O}(n\log(n))$. In all cases, note that the bounds for our new algorithm $\mathsf{O}_{\mathtt{LS}}$ match, or compare favorably to those for $\mathsf{O}_{\mathtt{FS}}$.

Remark 1. It was remarked in (van der Hoeven, 1997; van der Hoeven, 2002) that Karatsuba's multiplication could be rewritten directly as an online algorithm, thus leading to

an online algorithm using exactly the same number of operations. However, this algorithm is often not practical: the rewriting induces $\Omega(\log(n))$ function calls at each step, which makes it poorly suited to most practical implementations (see (van der Hoeven, 2002, Section 4.2.1)). For these reasons, we will not study this algorithm.

Remark 2. When the required precision n is known in advance, it is possible to adapt the online multiplication algorithms to this specific precision and thus lower the bounds given in Tables 1 and 2. The only known bounds are the ones stated without proof in (van der Hoeven, 2003); if n is known in advance, they are claimed to be $\mathsf{H}_{\mathtt{vdH}}(n) \sim \frac{1}{2}\,\mathsf{M}(n)$ for naive multiplication, $\mathsf{H}_{\mathtt{vdH}}(n) \sim \mathsf{M}(n)$ for Karatsuba and $\mathsf{H}_{\mathtt{vdH}}(n) \sim \frac{1}{2}\,\mathsf{M}(n)\log(n)$ for FFT.

Remark 3. We expect that our complexity results extend to online multiplication of p-adic integers. In this case, one has to handle carries, but we believe that the resulting extra cost should be only $\mathcal{O}(n\log(n))$.

2 Description of the algorithms

In this section, we present our main algorithms for half-line and online multiplication; we postpone the detailed complexity analysis to the next sections.

In all cases, we will use the following notational device. To compute a product of the form $a\,b$, either half-line or online, we will start from a "core" routine which takes as input a and b, as well as an extra input $c \in \mathbb{A}[x]$ and a parameter $i \in \mathbb{N}$: the polynomial c stores the current state of the multiplication and the integer i indicates at which step we are. Suppose that `Algo` is such an algorithm, with input in $\mathbb{A}[x]^3 \times \mathbb{N}$ and output in $\mathbb{A}[x]$; then, the main multiplication pattern is given in algorithm $\mathtt{Loop}_{\mathtt{Algo}}$.

Algorithm $\mathtt{Loop}_{\mathtt{Algo}}$

Input: $a, b \in \mathbb{A}[x]$ and $n \in \mathbb{N}$
Output: $c \in \mathbb{A}[x]$

1. $c = 0$
2. **for** i from 1 to n
   a. $c = \mathtt{Algo}(a, b, c, i)$
3. **return** c

To state correctness, we will use the following properties $(\mathcal{HL})$ and $(\mathcal{OL})$, which express that $\mathtt{Loop}_{\mathtt{Algo}}$ is a half-line, respectively online, multiplication algorithm. The half-line property reads as follows:

Property $(\mathcal{HL})$. *For any $n \in \mathbb{N}$ and any $a, b \in \mathbb{A}[x]$, the result $c \in \mathbb{A}[x]$ of the computation $\mathtt{Loop}_{\mathtt{Algo}}(a, b, n)$ satisfies $c = a\,b$ modulo x^n. Moreover, during the computation, the algorithm reads at most the coefficients $a_0, \ldots, a_{n-1}$ of the input a.*

The property for online algorithms is in a similar vein:

Property ($\mathcal{OL}$). *Algorithm* `Algo` *must satisfy Property* ($\mathcal{HL}$) *and, additionally, reads at most the coefficients* $b_0, ..., b_{n-1}$ *of the input* b.

For all algorithms below, we first give a recursive version of the algorithm, which is easy to describe and applies when the target precision n has a special form, such as $n = 2^k$ or $n = 2^k - 1$. Though these recursive versions won't be used afterwards, we feel that the reader may benefit from them as another point of view on the algorithms. For instance, we know from experience that some online algorithms are closely related to recursive divide-and-conquer algorithms; we hope that this presentation can help shed some light on this relation.

After the recursive presentation, we give the iterative form of the algorithms, obtained by "serializing" the recursion tree of the recursive algorithm (using iterative algorithms is necessary to fit in our framework of $\texttt{Loop}_{\texttt{Algo}}$ so that we can check properties ($\mathcal{HL}$) or ($\mathcal{OL}$)).

Finally, we introduce three auxiliary complexity functions $\mathbb{N} \to \mathbb{N}$, defined as

$$\begin{aligned}
\mathsf{M}^{(1)}(n) &:= \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \mathsf{M}(2^k) \\
\mathsf{M}^{(2)}(n) &:= \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n}{2^k} \right\rfloor \mathsf{M}(2^k) \\
\mathsf{M}^{(3)}(n) &:= \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n}{2^{k+1}} + \frac{1}{2} \right\rfloor \mathsf{M}(2^k).
\end{aligned}$$

2.1 Fischer and Stockmeyer's half-line algorithm

The first half-line multiplication algorithm was introduced by Fischer and Stockmeyer (Fischer and Stockmeyer, 1974), and rediscovered by van der Hoeven (van der Hoeven, 1997; van der Hoeven, 2002), up to a slight change in the recursion pattern.

We first give the recursive version of van der Hoeven's variant. In its recursive form, the algorithm computes $a\,b$, with $\deg(a) < n$ and $\deg(b) < n-1$, half-line in a, with n being a power of two. Define $a_0 = a \bmod x^{n/2}$ and $a_1 = a \operatorname{div} x^{n/2}$, as well as $b_0 = b \bmod x^{n/2-1}$ and $b_1 = b \operatorname{div} x^{n/2-1}$. Note that a and b are not split the same way. Then, compute the following:

1. $d_0 := a_0\,b_0$ (recursive half-line multiplication)
2. $d_0 := d_0 + a_0\,b_1\,x^{n/2-1}$ (off-line multiplication)
3. $d_0 := d_0 + a_1\,b_0\,x^{n/2}$ (recursive half-line multiplication)
4. $d_0 := d_0 + a_1\,b_1\,x^{n-1}$ (off-line multiplication)

One can verify that the half-line constraints are maintained throughout this process. This recursive algorithm computes the full multiplication $a\,b$ at step $n = 2^k$. However, we will see that the property ($\mathcal{HL}$) only guarantees that our product is correct modulo x^n at other steps. Algorithm `Halfline_FS` below gives the iterative version of this algorithm; a is the online argument, and $\nu_2(n)$ denotes the 2-adic valuation of integer n.

Algorithm `Halfline_FS`

Input: $a, b, c \in \mathbb{A}[x]$ and $i \in \mathbb{N}$
Output: $c \in \mathbb{A}[x]$

1. **for** k from 0 to $\nu_2(i)$
   a. $c = c + a_{i-2^k \ldots i}\, b_{2^k-1 \ldots 2^{k+1}-1}\, x^{i-1}$
2. **return** c

The diagram in Figure 1 shows the multiplications done when calling the iterative algorithm $\texttt{Loop}_{\texttt{Halfline\_FS}}$. The coefficients a_0, a_1, ... of a are placed in abscissa and the coefficients $b_0, b_1, ...$ of b in ordinate. Each unit square corresponds to a product between corresponding coefficients of a and b, *i.e.* the unit square whose left-bottom corner is at coordinates $(i,\ j)$ stands for $a_i\ b_j$. Each larger square corresponds to a product of polynomials; an $s \times s$ square whose left-bottom corner is at coordinates (i, j) stands for $a_{i \ldots i+s}\, b_{j \ldots j+s}$. The number inside the square indicates at which step i of $\texttt{Loop}_{\texttt{Halfline\_FS}}$ this computation is done in the iterative algorithm.

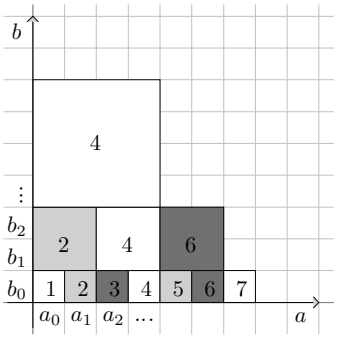

Fig. 1. Fischer-Stockmeyer's half-line multiplication

We can check on Figure 1 that for all $n \in \mathbb{N}$, all the coefficients of the product $\sum_{i=0}^{n-1} \sum_{j=0}^{i} a_j\, b_{i-j}\, x^i = (a \cdot b) \bmod x^n$ are computed before or at step n. We can also check that the algorithm is half-line in a since at step i, we use at most the coefficients $a_0, ...,$ a_{i-1} of a, so Algorithm `Halfline_FS` satisfies Property $(\mathcal{HL})$. However the operand b is off-line because, for example, the algorithm reads the coefficients $b_0, ..., b_6$ of b at step 4.

We will denote by $\mathsf{H}_{\texttt{FS}}(n)$ the arithmetic complexity of algorithm $\texttt{Loop}_{\texttt{Halfline\_FS}}$ with input precision n, *i.e.* to compute the product modulo x^n.

Proposition 4. *The following holds:*

$$\mathsf{H}_{\texttt{FS}}(n) = \mathsf{M}^{(2)}(n) + \mathcal{O}(n \log(n)).$$

PROOF. We do at each step a product of polynomials of degree 0 which each costs $\mathsf{M}(1)$, hence n such products to reach precision n. Additionally, we do every other step, starting from step 2, a product of polynomials of degree 1, which each costs $\mathsf{M}(2)$ for a total of $\lfloor n/2 \rfloor \mathsf{M}(2)$; generally, we do $\lfloor n/2^k \rfloor$ products in degree $2^k - 1$. Altogether, this accounts for the first term $\mathsf{M}^{(2)}(n) = \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n}{2^k} \right\rfloor \mathsf{M}(2^k)$ in the formula (note that the upper bound in the sum is the last value of k for which $\lfloor n/2^k \rfloor$ is nonzero).

Keeping an exact count of all additions necessary to compute c is not necessary: at worst, each product with input size 2^k incurs 2^{k+1} scalar additions to add its output to c. The total is thus at most $\sum_{k=0}^{\lfloor \log_2(n) \rfloor} \lfloor \frac{n}{2^k} \rfloor 2^{k+1} \leqslant \sum_{k=0}^{\lfloor \log_2(n) \rfloor} 2\,n$, which is $\mathcal{O}(n \log(n))$. □

2.2 van der Hoeven's half-line algorithm

Another half-line algorithm was introduced by van der Hoeven in (van der Hoeven, 2003). Whereas algorithm `Halfline_FS` used plain multiplication as a basic tool, this new algorithm uses middle products. As before, this algorithm is half-line with respect to the input a.

First, we give the recursive version; in this case, we have both $\deg(a) < n$ and $\deg(b) < n$, for n of the form $n = 2^k - 1$. This time, we define $a_0 = a \bmod x^{(n-1)/2}$, $a_1 = a \operatorname{div} x^{(n+1)/2}$ and $b_0 = b \bmod x^{(n-1)/2}$, so that all these polynomials have degree less than $2^{k-1} - 1$; define as well $a_0^\star = a \bmod x^{(n+1)/2}$. The algorithm does not compute the product $a\,b$, but rather the short product $a\,b \bmod x^n$ at steps n of the form $n = 2^k - 1$. It proceeds as follows:

1. $d_0 := a_0\,b_0 \bmod x^{(n-1)/2}$ (recursive half-line short product)
2. $d_0 := d_0 + \mathrm{MP}\Big(a_0^\star, b, \frac{(n+1)}{2}\Big)\, x^{(n-1)/2}$ (off-line middle product)
3. $d_0 := d_0 + \big(a_1\,b_0 \bmod x^{(n-1)/2}\big)\, x^{(n+1)/2}$ (recursive half-line short product)

Again, one can check that the half-line constraints are maintained for the recursive calls.

Since we compute only one middle product, whose size and cost are roughly those of one of the two multiplications done in the previous Subsection 2.1, we expect this algorithm to be faster than the previous one. To make this precise, we will analyze the iterative version $\mathtt{Loop}_{\mathtt{Halfline\_vdH}}$ of this algorithm, where subroutine `Halfline_vdH` looks as follows; the mechanism of this algorithm is sketched in Figure 2.

Algorithm `Halfline_vdH`

Input: $a, b, c \in \mathbb{A}[x]$ and $i \in \mathbb{N}$
Output: $c \in \mathbb{A}[x]$

1. Let $m := \nu_2(i)$
2. $c = c + \mathrm{MP}(a_{i-2^m \ldots i}, b_{0 \ldots 2^{m+1}-1}, 2^m)\, x^{i-1}$
3. **return** c

One easily sees that Algorithm `Halfline_vdH` satisfies Property $(\mathcal{HL})$, but the input argument b is off-line because (for example) at step 2, the algorithm reads b_0, b_1, b_2.

We will denote by $\mathsf{H}_{\mathtt{vdH}}(n)$ the arithmetic complexity of the half-line multiplication algorithm $\mathtt{Loop}_{\mathtt{HalfLine\_vdH}}$, with target precision n.

Proposition 5. *The following holds:*

$$\mathsf{H}_{\mathtt{vdH}}(n) = \mathsf{M}^{(3)}(n) + \mathcal{O}(n \log(n)).$$

PROOF. We claim that the cost of polynomial multiplications is given by

$$\sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n + 2^k}{2^{k+1}} \right\rfloor \mathsf{MP}(2^k). \tag{1}$$

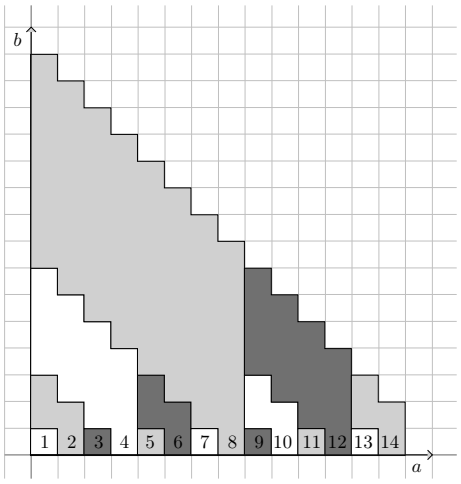

Fig. 2. van der Hoeven's half-line multiplication with middle product

Indeed, as we can see on Figure 2, for any integer k, we do a middle product of degree 2^k every 2^{k+1}th step, starting from step 2^k. We saw before that $\mathsf{MP}(2^k) = \mathsf{M}(2^k) + \mathcal{O}(2^k)$; applying this to formula (1) shows that the cost of polynomial multiplications is

$$\sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n+2^k}{2^{k+1}} \right\rfloor \mathsf{M}(2^k) + \mathcal{O}(n \log(n)) = \mathsf{M}^{(3)}(n) + \mathcal{O}(n \log(n)).$$

We must also take into account the additions of polynomials. Reasoning as in the proof of Proposition 4, we see that the extra cost is $\mathcal{O}(n \log(n))$. □

2.3 Fischer and Stockmeyer's online algorithm

We continue with the online multiplication algorithm due to Fischer and Stockmeyer, which is built upon their half-line algorithm. We first give the recursive version of this algorithm, for a and b of degree less than n, with n of the form $2^k - 1$. To compute $a\,b$, online in a and b, define $a_0 = a \bmod x^{(n-1)/2}$ and $a_1 = a \operatorname{div} x^{(n-1)/2}$, and define similarly b_0 and b_1. Then, compute the following:

1. $d_0 := a_0\,b_0$ (recursive online multiplication)
2. $d_0 := d_0 + a_0\,b_1\,x^{(n-1)/2}$ (half-line multiplication)
-. $d_0 := d_0 + a_1\,b_0\,x^{(n-1)/2}$ (half-line multiplication)
3. $d_0 := d_0 + a_1\,b_1\,x^{n-1}$ (off-line multiplication)

One can verify that the online constraints are maintained throughout this process, provided the two half-line products are done "in parallel".

Algorithm `Online_FS` below gives the iterative version of this algorithm, that applies to any n; as before, $\nu_2(n)$ denotes the 2-adic valuation of the integer n; Figure 3 sums up the computation made at each step by the iterative algorithm $\mathtt{Loop}_{\mathtt{Online\_FS}}$ and shows that it satisfies Property $(\mathcal{OL})$.

Algorithm `Online_FS`

Input: $a, b, c \in \mathbb{A}[x]$ and $i \in \mathbb{N}$
Output: $c \in \mathbb{A}[x]$

1. **for** k from 0 to $\nu_2(i+1)$
   a. $c = c + a_{i-2^k \dots i}\, b_{2^k-1 \dots 2^{k+1}-1}\, x^{i-1}$
   b. **if** $(i+1 = 2^{k+1})$
      return c
   c. $c = c + a_{2^k-1 \dots 2^{k+1}-1}\, b_{i-2^k \dots i}\, x^{i-1}$
2. **return** c

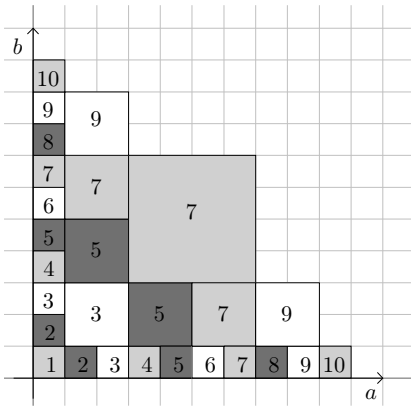

Fig. 3. Fischer and Stockmeyer's online multiplication

We denote by $\mathsf{O}_{\mathtt{FS}}(n)$ the arithmetic cost induced by all operations done up to precision n.

Proposition 6. *The following holds:*

$$\mathsf{O}_{\mathtt{FS}}(n) = 2\,\mathsf{M}^{(2)}(n+1) - 3\,\mathsf{M}^{(1)}(n+1) + \mathsf{M}(2^\ell) + \mathcal{O}(n\log(n)).$$

PROOF. For any $k \geqslant 0$, we do one product in degree $2^k - 1$ at step $2^{k+1} - 1$, then two such products every 2^kth step. The total number of such products with target precision n is

$$\left\lfloor \frac{n - (2^k - 1)}{2^k} \right\rfloor + \left\lfloor \frac{n - (2^{k+1} - 1)}{2^k} \right\rfloor = 2\left\lfloor \frac{n+1}{2^k} \right\rfloor - 3,$$

provided $(n+1)/2^k \geqslant 2$. This accounts for

$$\sum_{k=0}^{\lfloor \log_2(n+1) \rfloor - 1} \left(2\left\lfloor \frac{n+1}{2^k} \right\rfloor - 3 \right) \mathsf{M}(2^k).$$

Summing from 0 to $\ell' = \lfloor \log_2(n+1) \rfloor$ instead of from 0 to $\ell' - 1$, we can rewrite this contribution as $2\,\mathsf{M}^{(2)}(n+1) - 3\,\mathsf{M}^{(1)}(n+1) + \mathsf{M}(2^\ell)$.

As in the previous propositions, accounting for all polynomial additions induces the extra $\mathcal{O}(n\log(n))$ term. □

2.4 A new online algorithm

The algorithm in the previous subsection relied on Fischer-Stockmeyer's half-line algorithm to derive an online algorithm. In this subsection, we show how using van der Hoeven's half-line product algorithm of Section 2.2 leads to a new online multiplication algorithm.

As before, we start by giving the recursive version of the algorithm, which takes as input a and b of degrees less than n, with this time n of the form $2^k - 2$; the output is the short product $(a\,b) \bmod x^n$. We define now $a_0 = a \bmod x^{(n-2)/2}$, $a_0^\star = a \bmod x^{n/2}$ and $a_1 = a \operatorname{div} x^{n/2}$, and similarly for b_0, $b_0^\star$ and b_1, and compute the following:

1. $d_0 := a_0\,b_0 \bmod x^{(n-2)/2}$ (recursive online multiplication)
2. $d_0 := d_0 + \mathrm{HP}(a_0^\star, b_0^\star, n/2)\,x^{(n-2)/2}$ (off-line high product)
3. $d_0 := d_0 + \big(a_0\,b_1 \bmod x^{n/2}\big)\,x^{n/2}$ (half-line short product)

-. $d_0 := d_0 + \big(a_1\,b_0 \bmod x^{n/2}\big)\,x^{n/2}$ (half-line short product)

This gives us the following iterative algorithm, that is online with respect to inputs a and b.

Algorithm `Online_LS`

Input: $a, b, c \in \mathbb{A}[x]$ and $i \in \mathbb{N}$
Output: $c \in \mathbb{A}[x]$

1. $m = \nu_2(i+1)$
2. **if** $(i+1 = 2^m)$
   a. $c = c + \mathrm{HP}(a_{0\ldots i}, b_{0\ldots i}, i)\,x^{i-1}$
   b. **return** c
3. $c = c + \mathrm{MP}(a_{i-2^m\ldots i}, b_{0\ldots 2^{m+1}-1})\,x^{i-1}$
4. $c = c + \mathrm{MP}(b_{i-2^m\ldots i}, a_{0\ldots 2^{m+1}-1})\,x^{i-1}$
5. **return** c

Figure 4 sums up the computations of the iterative algorithm $\mathtt{Loop}_{\mathtt{Online\_LS}}$ and shows that it satisfies Property $(\mathcal{OL})$. Similarly to what we did in the previous sections, we denote by $\mathsf{O}_{\mathsf{LS}}(n)$ the cost of this algorithm with target precision n.

Proposition 7. *The following holds:*

$$\mathsf{O}_{\mathsf{LS}}(n) \leqslant (C_{\mathsf{SP}} - 2)\,\mathsf{M}^{(1)}(n+1) + 2\,\mathsf{M}^{(3)}(n+1) + \mathcal{O}(n\log(n)).$$

PROOF. Let $\ell' := \lfloor \log_2(n+1) \rfloor$. Depending on the step, our algorithm performs either one short product or two middle products. At steps $2^k - 1$ where $1 \leqslant k \leqslant \ell'$, one high short product is performed, for a total cost of $\sum_{k=1}^{\ell'} \mathsf{SP}(2^k - 1)$. Since $\mathsf{SP}(n) \leqslant C_{\mathsf{SP}}\,\mathsf{M}(n) + \mathcal{O}(n)$, this cost is less than $C_{\mathsf{SP}}\,\mathsf{M}^{(1)}(n+1) + \mathcal{O}(n\log(n))$.

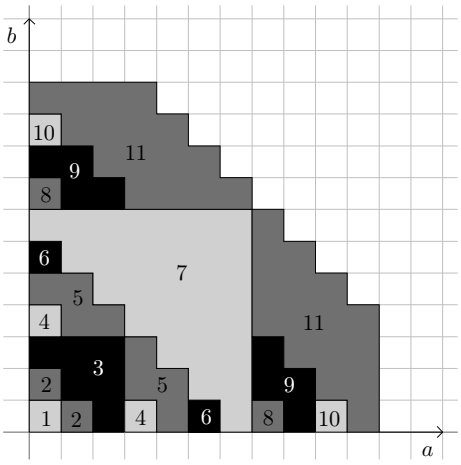

Fig. 4. Online multiplication with middle and short products

Then two middle products in size 2^k are done every 2^{k+1} steps, starting from step $3\cdot 2^k-1$, leading to a sum of terms of the form

$$\sum_{k=0}^{\ell'-1}\left\lfloor\frac{n+2^{k+1}-(3\cdot 2^k-1)}{2^{k+1}}\right\rfloor \mathsf{MP}(2^k)=\sum_{k=0}^{\ell'-1}\left\lfloor\frac{n+1}{2^{k+1}}-\frac{1}{2}\right\rfloor \mathsf{MP}(2^k).$$

Summing from 0 to ℓ' instead of from 0 to $\ell'-1$, and using the relation $\mathsf{MP}(2^k)=\mathsf{M}(2^k)+\mathcal{O}(2^k)$, these middle products cost $2\left(\mathsf{M}^{(3)}(n+1)-\mathsf{M}^{(1)}(n+1)\right)+\mathcal{O}(n\log(n))$.

As usual, the extra additions add up to a $\mathcal{O}(n\log(n))$ term. □

Remark 8. Even though there is no efficient short FFT multiplication algorithm, we can compute the high short product of Step 2 efficiently. Indeed, if a, b are polynomials of length n and $c=a\,b$, the FFT algorithm can compute $c_{0\ldots n}+c_{n\ldots 2n-1}=(a\,b) \bmod (x^n-1)$ using nth roots of unity. Since the part $c_{0\ldots n}$ was already computed by previous steps, we can access to $c_{n\ldots 2n-1}$. This computation takes about half the time of the FFT multiplication $a\,b$ which uses 2 nth roots of unity. However, we will see that for FFT multiplication, the contribution of these short products is in any case negligible.

3 Quasi-linear multiplication algorithms

We have expressed the complexity of all four online algorithms in terms of the auxiliary functions $\mathsf{M}^{(1)}, \mathsf{M}^{(2)}$ and $\mathsf{M}^{(3)}$. In this section and Section 4, we give asymptotically tight bounds on these auxiliary functions. Since their behaviors vary when M corresponds to a super-linear, resp. a quasi-linear function, we separate these two cases and start with the case of quasi-linear functions.

In this section, we work under the following hypothesis.

Hypothesis ($\mathcal{QL}$). *There exists $K\in\mathbb{R}_{>0}$ and $(i,j)\in\mathbb{N}^2$ such that*

$\mathsf{M}(2^k)\sim K\,2^k\,k^i\log_2(k)^j$.

This hypothesis is verified by the fast Fourier transform algorithm which satisfies $\mathsf{M}(2^k) = 9 \cdot 2^k\ k + \mathcal{O}(2^k)$ under the condition that there exists enough 2^kth roots of unity (see (von zur Gathen and Gerhard, 2003, Chapter 8.2)). Another suitable algorithm is the Truncated Fourier Transform because its cost coincides with the one of the FFT on powers of two (van der Hoeven, 2004). However, the Schönhage-Strassen multiplication algorithm does not fit in, as the ratio $\mathsf{M}(2^k)/(2^k k \log_2(k))$ has no limit at infinity.

Lemma 9. *If* M *satisfies hypothesis* $(\mathcal{QL})$ *and* $\ell = \lfloor \log_2(n) \rfloor$, *then*

$$\begin{aligned}
\mathsf{M}^{(1)}(n) &= \mathcal{O}(\mathsf{M}(n)),\\
\mathsf{M}^{(2)}(n) &\sim \frac{1}{(i+1)} \frac{n}{2^\ell} \mathsf{M}(2^\ell) \log_2(n),\\
\mathsf{M}^{(3)}(n) &\sim \frac{1}{2\,(i+1)} \frac{n}{2^\ell} \mathsf{M}(2^\ell) \log_2(n).
\end{aligned}$$

PROOF. From hypothesis $(\mathcal{QL})$, we get $\mathsf{M}^{(1)}(n) \sim \sum_{k=0}^{\lfloor \log_2(n) \rfloor} K\, 2^k\, k^i\, (\log_2 k)^j$. Because $\sum_{k=0}^{\ell} K\, 2^k\, k^i\, (\log_2 k)^j \leqslant 2\,(K\, 2^\ell\, \ell^i\, (\log_2 \ell)^j) \sim 2\, \mathsf{M}(2^\ell)$, we deduce our first point. Also, one has

$$\mathsf{M}^{(2)}(n) = \sum_{k=0}^{\ell} \lfloor n/2^k \rfloor\, \mathsf{M}(2^k) = \sum_{k=0}^{\ell} (n/2^k)\, \mathsf{M}(2^k) + \mathcal{O}\big(\mathsf{M}^{(1)}(n)\big).$$

Conclude using the following equivalents when n tends to infinity

$$\begin{aligned}
\sum_{k=0}^{\ell} (n/2^k)\, \mathsf{M}(2^k) &\sim K \sum_{k=0}^{\ell} (n/2^k)\, 2^k\, k^i \log_2^j(k)\\
&\sim K\, n \left(\sum_{k=0}^{\ell} k^i \log_2^j(k) \right)\\
&\sim K\, n \left(\frac{\ell^{i+1}}{i+1} \log_2^j(\ell) \right).
\end{aligned}$$

Finally, we deal with $\mathsf{M}^{(3)}$:

$$\mathsf{M}^{(3)}(n) = \sum_{k=0}^{\ell} \frac{n}{2^{k+1}} \mathsf{M}(2^k) + \mathcal{O}\big(\mathsf{M}^{(1)}(n)\big) \sim \frac{1}{2\,(i+1)} \frac{n}{2^\ell} \mathsf{M}(2^\ell) \log_2(2^\ell). \qquad \square$$

Proposition 10. *If* M *satisfies hypothesis* $(\mathcal{QL})$ *and* $\ell = \lfloor \log_2(n) \rfloor$, *then*

$$\begin{aligned}
\mathsf{H}_{\mathtt{FS}}(n) &\sim \frac{1}{2} \frac{n}{2^\ell} \mathsf{M}(2^\ell) \log_2(n),\\
\mathsf{H}_{\mathtt{vdH}}(n) &\sim \frac{1}{4} \frac{n}{2^\ell} \mathsf{M}(2^\ell) \log_2(n),\\
\mathsf{O}_{\mathtt{FS}}(n) &\sim \frac{n}{2^\ell} \mathsf{M}(2^\ell) \log_2(n),\\
\mathsf{O}_{\mathtt{LS}}(n) &\sim \frac{1}{2} \frac{n}{2^\ell} \mathsf{M}(2^\ell) \log_2(n).
\end{aligned}$$

PROOF. We use Lemma 9 and Proposition 12 for $\mathsf{H}_{\mathtt{FS}}$, $\mathsf{H}_{\mathtt{vdH}}$ and $\mathsf{O}_{\mathtt{FS}}$. For $\mathsf{O}_{\mathtt{LS}}$, we need to go back to Proposition 7 to deduce $\mathsf{O}_{\mathtt{LS}}(n) = 2\, \mathsf{M}^{(3)}(n+1) + \mathcal{O}\big(\mathsf{M}^{(1)}(n)\big)$ and our result. $\square$

Taking $\mathsf{M}(2^\ell) = 9 \cdot 2^\ell \, \ell + \mathcal{O}(2^\ell)$, the previous proposition gives the third row of Tables 1 and 2 after a quick simplification. We remark that the cost $\frac{n}{2^\ell} \, \mathsf{M}(2^\ell)$ is a smoothed version of $\mathsf{M}(n)$, especially for the "staircase" cost function of the FFT. Indeed, the expression $\frac{n}{2^\ell} \, \mathsf{M}(2^\ell) \, \log_2(2^\ell)$ is equivalent to $K \, n \, \log_2(n)^i \, \log_2(\log(n))^j$ under Hypothesis $(\mathcal{QL})$. The equivalent simplifies further to an actual $\mathsf{M}(n) \, \log_2(n)$ for the Truncated Fourier Transform algorithms or for quasi-linear evaluation interpolation schemes at n points.

4 Super-linear multiplication algorithms

Similarly to Section 3, we study the behavior of the auxiliary functions $\mathsf{M}^{(1)}, \mathsf{M}^{(2)}$ and $\mathsf{M}^{(3)}$, but in the case of a super-linear multiplication cost function M.

Our objective is to give bounds that relate as closely as possible to practice. We choose not to assume the classical superlinearity hypothesis ($\mathsf{M}(n)/n$ increasing), since this would not be satisfied for the exact operation count of Karatsuba's algorithm (this assumption would be satisfied if we used the upper bound $\mathsf{M}(n) = c \, n^{\log_2(3)}$, for some suitable c, but since we want precise estimates, we need to be more subtle).

4.1 Assumptions

In this section, we will work under the following assumption.

Hypothesis ($\mathcal{SL}$). *The arithmetic cost function* M *satisfies*

1. $\mathsf{M}(2\,n) = c\,\mathsf{M}(n) + a\,n + b$ *with* $a, b \in \mathbb{Z}$, $c \in]2; +\infty[$ *and*
2. $\mathsf{M}(2\,n+1) - \mathsf{M}(2\,n) \geqslant \mathsf{M}(3) - \mathsf{M}(2)$ *for* $n \geqslant 1$.

As we will see, this framework includes both naive and Karatsuba's algorithms. However, it does not include Toom-Cook algorithms, or the variant of Karatsuba's algorithm that reverts to the naive one for small values of n.

Naive multiplication. The naive algorithm has $\mathsf{M}(n) = n^2 + (n-1)^2 = 2\,n^2 - 2\,n + 1$. Using this expression, it is straightforward to verify that it satisfies hypothesis $(\mathcal{SL})$, with $\mathsf{M}(2\,n) = 4\,\mathsf{M}(n) + 4\,n - 3$.

Karatsuba's algorithm. Counting all operations, Karatsuba's algorithm can be implemented using $\mathsf{K}(n)$ operations, where $\mathsf{K}(1) = 1$ and K satisfies the following recurrence relation:

$$\mathsf{K}(n) = 2\,\mathsf{K}(\lceil n/2 \rceil) + \mathsf{K}(\lfloor n/2 \rfloor) + 4\,n - 4.$$

The first two terms in the right-hand side require no justification, but we may say a few words about the linear term $4\,n - 4$. For instance, for $n = 2\,m$, writing $a = a_0 + x^m \, a_1$ and $b = b_0 + x^m \, b_1$, we do $2\,m$ additions prior to the recursive calls to compute $a_0 + a_1$ and $b_0 + b_1$, $6\ m - 3$ additions and subtractions after the recursive call to compute $(a_0 + a_1)\,(b_0 + b_1) - a_0\,b_0 - a_1\,b_1$ and $m - 1$ additions to add that term to the result, for a total of $8\,m - 4 = 4\,n - 4$. The case $n = 2\,m + 1$ is similar.

In particular, we have $\mathsf{K}(1) = 1, \mathsf{K}(2) = 7$ and $\mathsf{K}(3) = 23$. For even inputs, this becomes $\mathsf{K}(2\,n) = 3\,\mathsf{K}(n) + 8\,n - 4$, which shows that the first part of our assumption is satisfied with $c = 3, a = 8$ and $b = -4$.

To prove the second item of $(\mathcal{SL})$, we show by induction that for $n \geqslant 1$, $\mathsf{K}(n+1) - \mathsf{K}(n) \geqslant \mathsf{K}(2) - \mathsf{K}(1)$. Indeed, the case $n = 1$ is clear, and the inductive step follows from the equalities

$$\mathsf{K}(n+1) - \mathsf{K}(n) = \begin{cases} 2\,(\mathsf{K}(n/2+1) - \mathsf{K}(n/2)) + 4 & \text{if } n \text{ even} \\ \mathsf{K}((n-1)/2+1) - \mathsf{K}((n-1)/2) + 4 & \text{if } n \text{ odd.} \end{cases}$$

As claimed, we deduce that

$$\mathsf{K}(2\,n+1) - \mathsf{K}(2\,n) = 2\,(\mathsf{K}(n+1) - \mathsf{K}(n)) + 4 \geqslant 2\,(\mathsf{K}(2) - \mathsf{K}(1)) + 4 = \mathsf{K}(3) - \mathsf{K}(2).$$

Remark that it is possible to save $\lfloor n\,/\,2 \rfloor - 1$ redundant additions, see for instance Exercise 1.9 in (Brent and Zimmermann, 2011). This improved algorithm still satisfies our assumptions, but our implementation does not use it.

4.2 Complexity analysis

In the following lemmas, we use assumption $(\mathcal{SL})$ to prove upper bounds on functions $\mathsf{M}^{(1)}$, $\mathsf{M}^{(2)}$ and $\mathsf{M}^{(3)}$. To this effect, let us define the constants

$$a' := \frac{a}{c-2}, \quad b' := \frac{b}{c-1} \quad \text{and} \quad e := |a'| + |b'|,$$

as well as the function $d(\lambda) := \mathsf{M}(\lambda) + a'\,\lambda + b'$, for λ in $\mathbb{N}$.

Lemma 11. *Assumption $(\mathcal{SL})$ implies that $|\mathsf{M}(2^k\,\lambda) - d(\lambda)\,c^k| \leqslant e\,2^k\,\lambda$ holds for $\lambda \in \mathbb{N}^*$.*

PROOF. It suffices to unroll the recurrence k times, and sum the geometric progressions:

$$\begin{aligned} \mathsf{M}(2^k\,\lambda) &= c\,\mathsf{M}(2^{k-1}\,\lambda) + a\,2^{k-1}\,\lambda + b \\ &= c^k\,\mathsf{M}(\lambda) + a\,\lambda\,(2^{k-1} + 2^{k-2}\,c + \cdots + c^{k-1}) + b\,(1 + \cdots + c^{k-1}) \\ &= c^k\,\mathsf{M}(\lambda) + \frac{a\,\lambda\,(c^k - 2^k)}{c-2} + \frac{b\,(c^k-1)}{c-1} \\ &= c^k \left(\mathsf{M}(\lambda) + \frac{a\,\lambda}{c-2} + \frac{b}{c-1} \right) - \frac{a\,2^k\,\lambda}{c-2} - \frac{b}{c-1}. \end{aligned}$$

The conclusion follows immediately. □

Remark in particular that the former lemma implies that $|\mathsf{M}(2^k) - d(1)\,c^k| = \mathcal{O}(2^k)$. In particular, because M is non-negative, we deduce that $d(1) > 0$.

Lemma 12. *Let n be in $\mathbb{N}$, with base-2 expansion given by $n = \sum_{i=0}^{\ell} n_i\,2^i$, where $\ell := \lfloor \log_2(n) \rfloor$. Then, under assumption $(\mathcal{SL})$, we have*

$$\begin{aligned} \mathsf{M}^{(1)}(n) &= \frac{c}{c-1}\,\mathsf{M}(2^\ell) + \mathcal{O}(n) \\ \mathsf{M}^{(2)}(n) &= \frac{c}{c-2} \sum_{i=0}^{\ell} n_i\,\mathsf{M}(2^i) + \mathcal{O}(n \log(n)) \\ \mathsf{M}^{(3)}(n) &= \frac{c-1}{c-2} \sum_{i=0}^{\ell} n_i\,\mathsf{M}(2^i) + \mathcal{O}(n \log(n)). \end{aligned}$$

PROOF. In all that follows, we write for simplicity $d := d(1)$. We start with $\mathsf{M}^{(1)}$, applying the previous lemma to each summand:

$$\begin{aligned}
\mathsf{M}^{(1)}(n) &= \sum_{k=0}^{\ell} \mathsf{M}(2^k) = \sum_{k=0}^{\ell} d\,c^k + \mathcal{O}\left(\sum_{k=0}^{\ell} 2^k\right) = \left(\frac{c}{c-1}\,d\,c^{\ell}\right) + \mathcal{O}(n) \\
&= \left(\frac{c}{c-1}\,\mathsf{M}(2^{\ell}) + \mathcal{O}(n)\right) + \mathcal{O}(n)
\end{aligned}$$

Next, one has

$$\begin{aligned}
\mathsf{M}^{(2)}(n) &= \sum_{k=0}^{\ell} \left\lfloor \frac{n}{2^k} \right\rfloor \mathsf{M}(2^k) = \sum_{k=0}^{\ell} \sum_{i=k}^{\ell} n_i\, 2^{i-k}\, \mathsf{M}(2^k) \\
&= \sum_{k=0}^{\ell} \sum_{i=k}^{\ell} n_i\, 2^{i-k}\, d\, c^k + \mathcal{O}\left(\sum_{k=0}^{\ell} \sum_{i=k}^{\ell} n_i\, 2^{i-k}\, 2^k\right) \\
&= \sum_{i=0}^{\ell} n_i\, d\, 2^i \sum_{k=0}^{i} \left(\frac{c}{2}\right)^k + \mathcal{O}\left(\sum_{i=0}^{\ell} n_i\, 2^i\, (i+1)\right) \\
&= \left(\sum_{i=0}^{\ell} n_i\, d\, 2^i\, \frac{(c/2)^{i+1}-1}{(c/2)-1}\right) + \mathcal{O}(n\log(n)) \\
&= \left(\frac{(c/2)}{(c/2)-1}\left[\sum_{i=0}^{\ell} n_i\, d\, c^i\right] + \mathcal{O}(n)\right) + \mathcal{O}(n\log(n)) \\
&= \frac{c}{c-2}\left[\sum_{i=0}^{\ell} n_i\, \mathsf{M}(2^i) + \mathcal{O}(n)\right] + \mathcal{O}(n\log(n))
\end{aligned}$$

Finally, we have the inequalities

$$\begin{aligned}
\mathsf{M}^{(3)}(n) &= \sum_{k=0}^{\ell} \left(\left\lfloor \frac{n}{2^{(k+1)}} \right\rfloor + n_k\right) \mathsf{M}(2^k) = \sum_{k=0}^{\ell} \sum_{i=k+1}^{\ell} n_i\, 2^{i-(k+1)}\, \mathsf{M}(2^k) \\
&= \sum_{k=0}^{\ell} \sum_{i=k+1}^{\ell} n_i\, 2^{i-(k+1)}\, d\, c^k + \mathcal{O}(n\log(n)) \\
&= \left(\sum_{i=1}^{\ell} n_i\, 2^{i-1}\, d\, \frac{(c/2)^i - 1}{(c/2)-1}\right) + \mathcal{O}(n\log(n)) \\
&= \left(\frac{(1/2)}{(c/2)-1}\left[\sum_{i=0}^{\ell} n_i\, d\, c^i\right] + \mathcal{O}(n)\right) + \mathcal{O}(n\log(n)) \\
&= \frac{1}{c-2}\left[\sum_{i=0}^{\ell} n_i\, \mathsf{M}(2^i) + \mathcal{O}(n)\right] + \mathcal{O}(n\log(n))
\end{aligned}$$

□

The following inequality will allow us to control terms that appear in the estimates for $\mathsf{M}^{(2)}(n)$ and $\mathsf{M}^{(3)}(n)$ given above. For this, we introduce the notation $C := \frac{d(3)-d(2)}{d(1)}$.

Lemma 13. *Let n be in $\mathbb{N}$, with base-2 expansion given by $n = \sum_{i=0}^{\ell} n_i\, 2^i$, where*

$\ell := \lfloor \log_2(n) \rfloor$. *Then, under assumption* $(\mathcal{SL})$, *we have*

$$\mathsf{M}(2^\ell) + C \sum_{i=0}^{\ell-1} n_i \, \mathsf{M}(2^i) \leqslant \mathsf{M}(n) + \mathcal{O}(n \log(n)).$$

In particular, if $C \geqslant 1$, *we have*

$$\sum_{i=0}^{\ell} n_i \, \mathsf{M}(2^i) \leqslant \mathsf{M}(n) + \mathcal{O}(n \log(n)).$$

PROOF. The proof proceeds in three steps.

1. First, we prove that the inequality $d(2\ n\ +\ 1)\ -\ d(2\ n)\ \geqslant\ d(3)\ -\ d(2)$ holds for any $n \geqslant 1$. Indeed, we have that $d(2\,n+1) - d(2\,n) = \mathsf{M}(2\,n+1) - \mathsf{M}(2\,n) + a'$, so the assumption $\mathsf{M}(2\,n+1) - \mathsf{M}(2\,n) \geqslant \mathsf{M}(3) - \mathsf{M}(2)$ establishes our claim.

2. Next, we establish that for all $k \in \mathbb{N}$ and $m \geqslant 1$, we have

$$\mathsf{M}(2^{k+1}\,m) + C\,\mathsf{M}(2^k) \leqslant \mathsf{M}(2^{k+1}\,m + 2^k) + e'\,2^{k+1}\,m, \tag{2}$$

for some e' that does not depend on k or m. Indeed, Lemma 11 implies the inequalities

$$\begin{aligned} \mathsf{M}(2^{k+1}\,m) + C\,\mathsf{M}(2^k) &\leqslant c^k\,(d(2\,m) + C\,d(1)) + e\,2^k\,(2\,m + C) \\ c^k\,d(2\,m+1) &\leqslant \mathsf{M}(2^k\,(2\,m+1)) + e\,2^k\,(2\,m+1). \end{aligned}$$

On the other hand, inequality (2) implies that $d(2\,m) + C\,d(1) \leqslant d(2\,m+1)$, and our claim follows by taking (for instance) $e' = e\,(C+5)/2$.

3. We can now prove the lemma. Take n in $\mathbb{N}$, with base-2 digits $n_0, ..., n_\ell$. Applying inequality (2) with $k = \ell - 1$ and $m = 1$ yields

$$\mathsf{M}(2^\ell) + C\,n_{\ell-1}\,\mathsf{M}(2^{\ell-1}) \leqslant \mathsf{M}(2^\ell + n_{\ell-1}2^{\ell-1}) + e'\,2^\ell \leqslant \mathsf{M}(2^\ell + n_{\ell-1}2^{\ell-1}) + e'\,n',$$

with $n' = 2\,n$. Adding the term $C\,n_{\ell-2}\,\mathsf{M}(2^{\ell-2})$ and applying the same inequality (2) with $k = \ell - 2$ and $m = 2 + n_{\ell-1}$, so that we still have $2^{k+1}\,m \leqslant n'$, we get

$$\begin{aligned} \mathsf{M}(2^\ell) + \sum_{i=\ell-2}^{\ell-1} C\,n_i\,\mathsf{M}(2^i) &\leqslant \mathsf{M}(2^\ell + n_{\ell-1}2^{\ell-1}) + C\,n_{\ell-2}\,\mathsf{M}(2^{\ell-2}) + e'\,n' \\ &\leqslant \mathsf{M}(2^\ell + n_{\ell-1}2^{\ell-1} + n_{\ell-2}\,2^{\ell-2}) + e'\,(2\,n'). \end{aligned}$$

We can continue in this manner until we get $\mathsf{M}(2^\ell) + C \sum_{i=0}^{\ell-1} n_i\,\mathsf{M}(2^i) \leqslant \mathsf{M}(n) + e'\,\ell\,n'$, which proves the lemma. □

Corollary 14. *Under assumption* $(\mathcal{SL})$ *and if* $C \geqslant 1$, *one has*

$$\mathsf{H}_{\mathtt{FS}}(n) \leqslant \frac{c}{c-2}\,\mathsf{M}(n) + \mathcal{O}(n \log(n)) \quad \textit{and} \quad \mathsf{H}_{\mathtt{vdH}}(n) \leqslant \frac{c-1}{c-2}\,\mathsf{M}(n) + \mathcal{O}(n \log(n))$$

and these bound are asymptotically optimal since

$$\mathsf{H}_{\mathtt{FS}}(2^m) \sim \frac{c}{c-2}\,\mathsf{M}(2^m) \quad \textit{and} \quad \mathsf{H}_{\mathtt{vdH}}(2^m) \sim \frac{c-1}{c-2}\,\mathsf{M}(2^m).$$

PROOF. We deal only with $\mathsf{H}_{\mathtt{FS}}$ since $\mathsf{H}_{\mathtt{vdH}}$ is handled similarly. Using Proposition 12, then Lemma 12 for equality (3) and Lemma 13 for the inequality (4), we have for all $n \in \mathbb{N}$,

$$\begin{aligned} \mathsf{H}_{\mathtt{FS}}(n) &= \frac{c}{c-2} \sum_{i=0}^{\ell} n_i \, \mathsf{M}(2^i) + \mathcal{O}(n \log(n)) && (3) \\ &\leqslant \frac{c}{c-2} \, \mathsf{M}(n) + \mathcal{O}(n \log(n)). && (4) \end{aligned}$$

When $n = 2^m$, one has

$$\mathsf{H}_{\mathtt{FS}}(2^m) = \frac{c}{c-2} \, \mathsf{M}(2^m) + \mathcal{O}(n \log(n)) \sim \frac{c}{c-2} \, \mathsf{M}(2^m). \qquad \square$$

Proposition 15. *Under assumption* $(\mathcal{SL})$ *and if* $C \geqslant \frac{2\,c\,(c-1)}{c+2}$*, one has*

$$\mathsf{O}_{\mathtt{FS}}(n) \leqslant \frac{c+2}{(c-2)\,(c-1)} \, \mathsf{M}(n+1) + \mathcal{O}(n \log(n))$$

and this bound is asymptotically optimal, since

$$\mathsf{O}_{\mathtt{FS}}(2^m - 1) \sim \frac{c+2}{(c-2)\,(c-1)} \, \mathsf{M}(2^m).$$

PROOF. Let $\ell := \lfloor \log_2(n+1) \rfloor$ and $n+1 = \sum_{i=0}^{\ell} n_i \, 2^i$ be the base-2 expansion of $n+1$. Then, using Proposition 12 and Lemma 12, one deduces

$$\begin{aligned} \mathsf{O}_{\mathtt{FS}}(n) &= 2 \left(\frac{c}{c-2} \sum_{i=0}^{\ell} n_i \, \mathsf{M}(2^i) + \mathcal{O}(n \log(n)) \right) - 3 \left(\frac{c}{c-1} \, \mathsf{M}(2^\ell) + \mathcal{O}(n) \right) + \mathsf{M}(2^\ell) \\ &= \left(\frac{2\,c}{c-2} - \frac{3\,c}{c-1} + 1 \right) \mathsf{M}(2^\ell) + \frac{2 \cdot c}{c-2} \sum_{i=0}^{\ell-1} n_i \, \mathsf{M}(2^i) + \mathcal{O}(n \log(n)) \\ &= C_1 \, \mathsf{M}(2^\ell) + C_2 \sum_{i=0}^{\ell-1} n_i \, \mathsf{M}(2^i) + \mathcal{O}(n \log(n)) \end{aligned}$$

with $C_1 = \frac{c+2}{(c-2)\,(c-1)}$ and $C_2 = \frac{2 \cdot c}{c-2}$. Provided that $C_2 \,/\, C_1 \leqslant C$, we can then use Lemma 13 to deduce that $\mathsf{O}_{\mathtt{FS}}(n) \leqslant C_1 \, \mathsf{M}(n+1) + \mathcal{O}(n \log(n))$. For $n+1 = 2^m$, all n_i are zero for $i < \ell$, so one has

$$\mathsf{O}_{\mathtt{FS}}(2^m - 1) = C_1 \, \mathsf{M}(2^m) + \mathcal{O}(n \log(n)) \sim C_1 \, \mathsf{M}(2^m).$$

$\square$

Proposition 16. *Under assumption* $(\mathcal{SL})$ *and if* $C \geqslant \frac{2\,(c-1)^2}{c\,(c-2)\,C_{\mathsf{SP}} + 2}$*, one has*

$$\mathsf{O}_{\mathtt{LS}}(n) \leqslant \frac{c\,(c-2)\,C_{\mathsf{SP}} + 2}{(c-2)\,(c-1)} \, \mathsf{M}(n+1) + \mathcal{O}(n \log(n))$$

and these bounds are asymptotically optimal provided that $\mathsf{SP}(2^k - 1) \sim C_{\mathsf{SP}} \, \mathsf{M}(2^k)$*:*

$$\mathsf{O}_{\mathtt{LS}}(2^m - 1) \sim_{m \to \infty} \frac{c\,(c-2)\,C_{\mathsf{SP}} + 2}{(c-2)\,(c-1)} \, \mathsf{M}(2^m).$$

PROOF. Let $\ell := \lfloor \log_2(n+1) \rfloor$ and $n+1 = \sum_{i=0}^{\ell} n_i\, 2^i$ be the base-2 expansion of $n+1$. Using Proposition 12 and Lemma 12, we deduce

$$\begin{aligned}\mathsf{O}_{\mathsf{LS}}(n) &\leqslant (C_{\mathsf{SP}} - 2)\left(\frac{c}{c-1}\,\mathsf{M}(2^\ell)\right) + 2\left(\frac{c-1}{c-2}\sum_{i=0}^{\ell} n_i\,\mathsf{M}(2^i)\right) + \mathcal{O}(n\log(n)) \\ &= C_1'\,\mathsf{M}(2^\ell) + C_2'\sum_{i=0}^{\ell-1} n_i\,\mathsf{M}(2^i) + \mathcal{O}(n\log(n))\end{aligned}$$

with $C_1' = \frac{c\,(c-2)\,C_{\mathsf{SP}} + 2}{(c-2)\,(c-1)}$ and $C_2' = \frac{2\,(c-1)}{c-2}$. Provided that $C_2'/C_1' \leqslant C$, we can then use Lemma 13 to deduce that $\mathsf{O}_{\mathsf{LS}}(n) \leqslant C_1'\,\mathsf{M}(n+1) + \mathcal{O}(n\log(n))$. For $n+1 = 2^m$, all n_i are zero for $i < \ell$, so one has

$$\mathsf{O}_{\mathsf{LS}}(2^m - 1) = C_1'\,\mathsf{M}(2^m) + \mathcal{O}(n\log(n)) \sim C_1'\,\mathsf{M}(2^m)$$

under the condition that C_{SP} is optimal in the sense $\mathsf{SP}(2^k - 1) \sim C_{\mathsf{SP}}\,\mathsf{M}(2^k)$. □

Let us now verify that the naive and Karatsuba's multiplication algorithms satisfy the hypotheses of Corollary 14 and Propositions 15 and 16. Proposition 15 requires

$$C \geqslant \frac{2\,c\,(c-1)}{c+2} = \begin{cases} 4 & \text{if } c = 4 \\ 12/5 & \text{if } c = 3 \end{cases},$$

whereas Propositions 16 is verified whenever

$$C \geqslant \frac{2\,(c-1)^2}{c\,(c-2)/2 + 2} = \begin{cases} 3 & \text{if } c = 4 \\ 16/7 & \text{if } c = 3 \end{cases},$$

since $C_{\mathsf{SP}} \geqslant 1/2$.

Naive multiplication. Since $\mathsf{M}(n) \sim 2\,n^2$ and $\mathsf{M}(2^k\,\lambda) \sim d(\lambda)\,4^k$ using Lemma 11, we get $d(\lambda) = 2\,\lambda^2$ and $C = \frac{d(3) - d(2)}{d(1)} = 5$. Therefore the naive multiplication satisfies the hypotheses of Corollary 14 and Propositions 15 and 16. This gives us the first row of Tables 1 and 2.

Karatsuba's algorithm. Recall that $\mathsf{K}(2\,n) = 3\,\mathsf{K}(n) + 8\,n - 4$, which implies $a' = 8$ and $b' = -2$. Since $d(\lambda) = \mathsf{M}(\lambda) + a'\,\lambda + b'$, we get $C = \frac{d(3) - d(2)}{d(1)} = \frac{45 - 13}{7} = \frac{32}{7}$. Therefore Karatsuba's multiplication satisfies the hypotheses of Corollary 14 and Propositions 15 and 16, from which we deduce the second row of Tables 1 and 2.

5 Implementation and timings

We give timings of the https://github.com/romainlebreton/OnlineMultiplication multiplication algorithms for the case of power series $\mathbb{F}_p[[x]]$ with the 29-bit prime number $p = 268435459$. Computations were done on one core of an Intel Core i7 running at 3.6 GHz with 8Gb of RAM running a 64-bit Linux. Our implementation is available at `https://github.com/romainlebreton/OnlineMultiplication`. It uses the polynomial multiplication of NTL 6.0.0 (Shoup et al., 1990). The threshold between the naive and Karatsuba's multiplications is at degree 16 and the one between Karatsuba's and FFT multiplications at degree 600. Our middle product implementation is based on the implementation described in (Bostan et al., 2003).

In Figure 5, we plot the timings in milliseconds of the multiplication of polynomials and of several online multiplication algorithms on power series depending on the precision in abscissa. The name M stands for NTL's multiplication, the name $\mathsf{H}_{\mathtt{vdH}}$ stands for the half-line multiplication using middle product of Section 2.2, the name $\mathsf{O}_{\mathtt{LS}}$ stands for the online multiplication using middle (and short) product of Section 2.4, and so on.

Online algorithms are always slower than off-line algorithms since they have an additional constraint. However, we will see that online algorithms are faster in very small precisions: this is because we compare online algorithms that compute short products $a\,b \bmod x^n$ at each step (and occasionally more, such as the full product for $\mathsf{O}_{\mathtt{FS}}$ and $\mathsf{O}_{\mathtt{LS}}$ when $n = 2^\ell$) and an off-line algorithm that always computes the full product $a\,b$.

We now draw the ratio of the timings of all online algorithms compared to NTL's off-line multiplication. We give three figures depending on which off-line multiplication algorithm is used. We start with the naive algorithm used in precisions $1 \leqslant n < 16$.

For these small precisions, the ratio of timings does not follow our theoretical analysis. We reckon that cache effects or other low-level hardware specificities have a non-negligible effect on our timings. Still, we can notice from this figure that the variants using middle product always improve the online algorithms.

Let us turn to intermediate precisions corresponding to Karatsuba's algorithm. NTL implements the variant of Karatsuba's algorithm using the naive variant in small degrees for plain multiplication and we coded an odd/even decomposition for short product. Although Proposition 16 does not deal with this hybrid multiplication algorithm, we believe that the results for "pure" Karatsuba's multiplication could apply in this case for n large enough and yield bounds $\mathsf{O}_{\mathtt{FS}} \leqslant 2.5\,\mathsf{M}(n)$, $\mathsf{H}_{\mathtt{FS}} \leqslant 3\,\mathsf{M}(n)$ and $\mathsf{H}_{\mathtt{vdH}} \leqslant 2\,\mathsf{M}(n)$, omitting terms in $\mathcal{O}(n)$. Concerning our algorithm, the short product has a ratio $C_{\mathsf{SP}} = 0.6$ in practice so we would expect $\mathsf{O}_{\mathtt{LS}} \leqslant 1.9\,\mathsf{M}(n)$.

This plot confirms the theoretical bounds for Karatsuba's multiplication, except on a few points for $\mathsf{H}_{\mathtt{vdH}}$. Once again, the variants using middle product always improve online algorithms by a constant factor.

Finally for precision corresponding to the FFT algorithm, the ratio grows with the precision. Figure 8 shows the logarithmic growth of the ratio for precisions $n = 2^\ell$. Note that NTL uses the 3-primes FFT algorithm on our field $\mathbb{F}_p$ since it was lacking 2^ℓth roots of unity (see (von zur Gathen and Gerhard, 2003, Chapter 8.3)). This algorithm still matches Hypothesis ($\mathcal{QL}$) in the range of degrees we consider and our analysis applies.

We can improve this analysis by plotting $\mathsf{T}(n) \,/\, (n \,/\, 2^\ell\, \mathsf{M}(2^\ell) \log_2(n))$, where T denotes one of the functions $\mathsf{H}_{\mathtt{FS}}$, ... that we are considering, expecting to observe constant ratios (in theory, this ratio should tend to 1 for $\mathsf{O}_{\mathtt{FS}}$, $1/2$ for $\mathsf{H}_{\mathtt{FS}}$ and $\mathsf{O}_{\mathtt{LS}}$, and $1/4$ for $\mathsf{H}_{\mathtt{vdH}}$). This is done in Figure 9, where we observe a good agreement with theory.

In conclusion, we can see that the use of middle product always improves the performance of both the online and half-line multiplication algorithms. We save up to a factor 2, which is attained for the FFT multiplication.

Acknowledgments

We are grateful to P. Zimmermann and the anonymous reviewers for their thorough proofreadings and helpful comments.

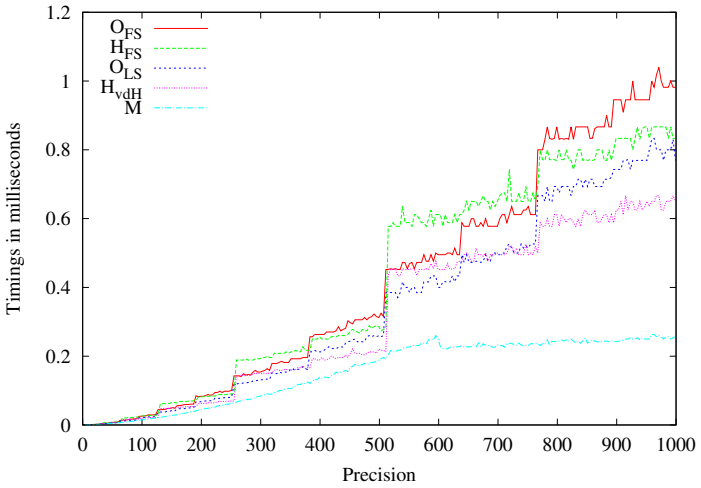

Fig. 5. Timings of different multiplication algorithms

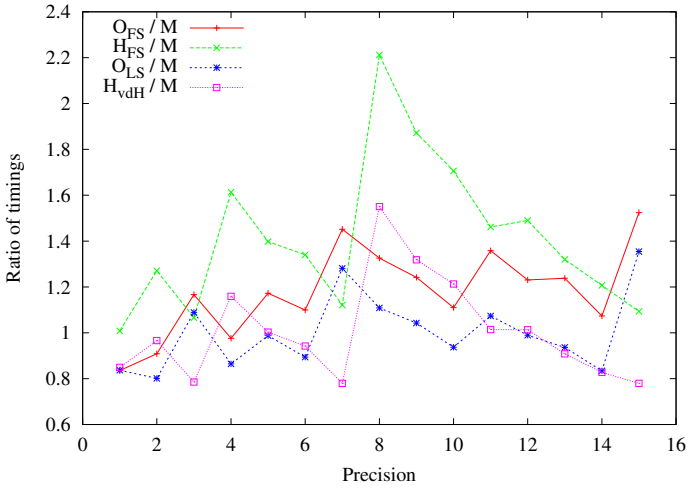

Fig. 6. Ratio of timings of different online products w.r.t. naive polynomial multiplication

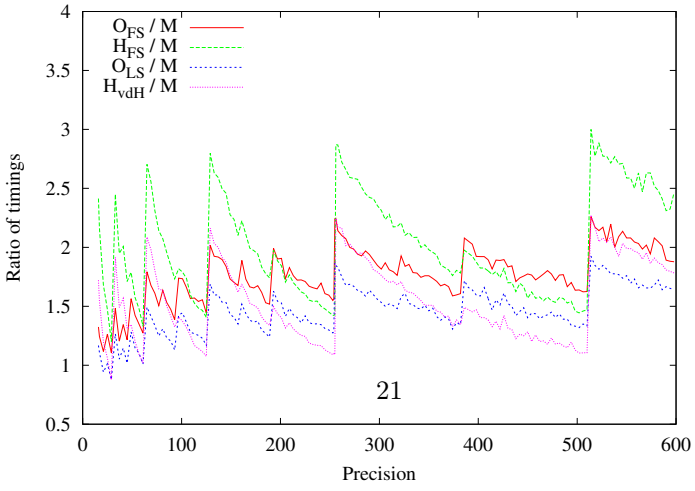

Fig. 7. Ratio of timings of different online products w.r.t. "hybrid" Karatsuba's multiplication

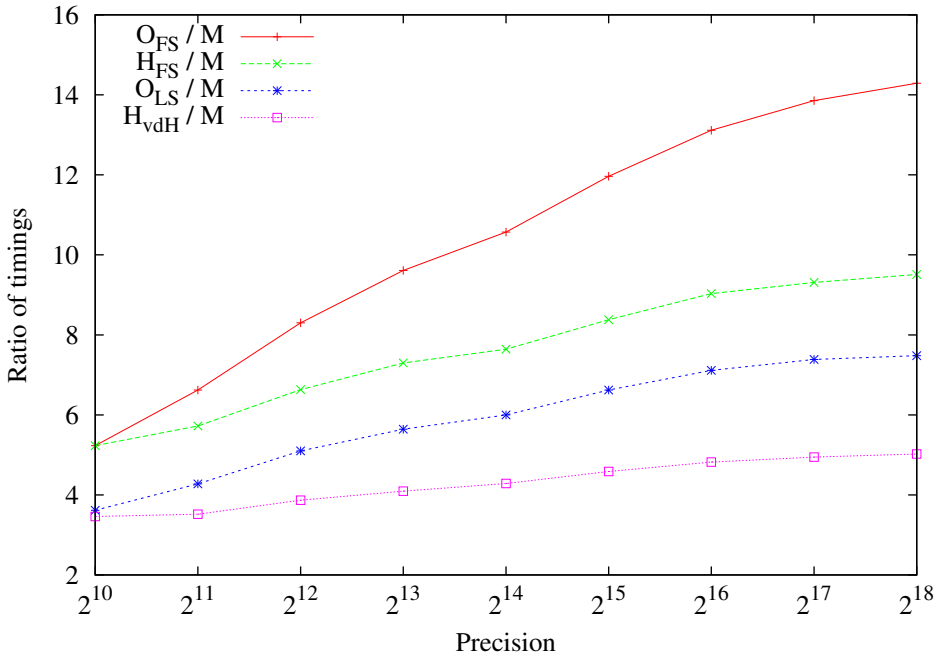

Fig. 8. Ratio of timings of online products w.r.t. FFT multiplication on precisions $N = 2^\ell$

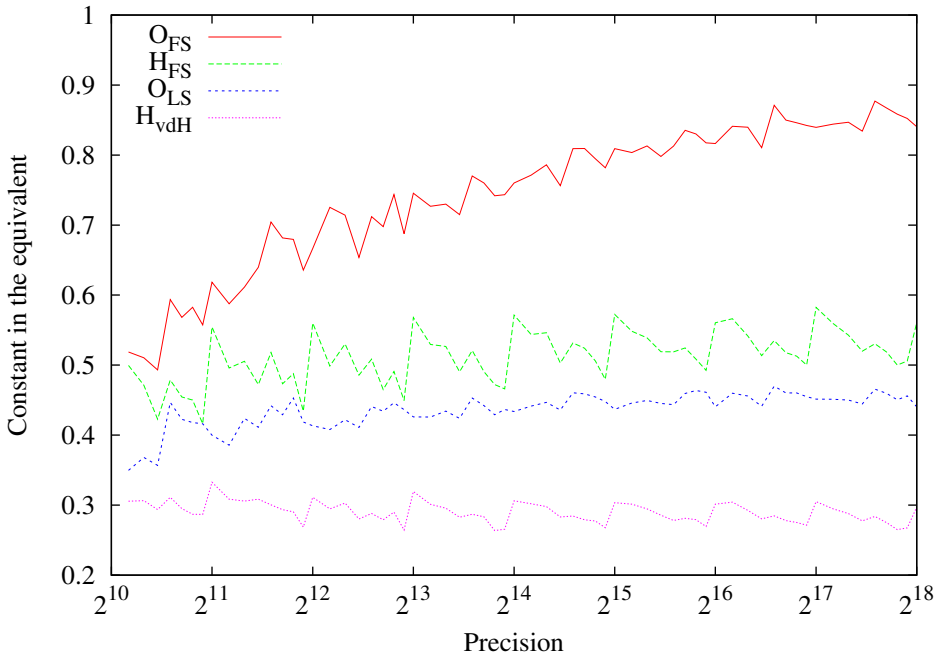

Fig. 9. Estimation of the constants in the equivalences of online product costs using FFT

References

Berthomieu, J., Lebreton, R., 2012. Relaxed p-adic Hensel lifting for algebraic systems. In: Proceedings of ISSAC'12. ACM Press, pp. 59–66.

Berthomieu, J., van der Hoeven, J., Lecerf, G., 2011. Relaxed algorithms for p-adic numbers. J. Théor. Nombres Bordeaux 23 (3), 541–577.

Bostan, A., Lecerf, G., Schost, É., 2003. Tellegen's principle into practice. In: Proceedings of ISSAC'03. ACM Press, pp. 37–44.

Brent, R., Zimmermann, P., 2011. Modern computer arithmetic. Vol. 18 of Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge.

Bürgisser, P., Clausen, M., Shokrollahi, M. A., 1997. Algebraic complexity theory. Vol. 315 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, Berlin, with the collaboration of Thomas Lickteig.

Fischer, M. J., Stockmeyer, L. J., 1974. Fast on-line integer multiplication. J. Comput. System Sci. 9, 317–331.

Hanrot, G., Quercia, M., Zimmermann, P., 2004. The middle product algorithm. I. Appl. Algebra Engrg. Comm. Comput. 14 (6), 415–438.

Hanrot, G., Zimmermann, P., 2004. A long note on Mulders' short product. J. Symbolic Comput. 37 (3), 391–401.

Hennie, F. C., 1966. On-line Turing machine computations. Electronic Computers, IEEE Transactions on EC-15 (1), 35 –44.

Lebreton, R., 2012. Contributions to relaxed algorithms and polynomial system solving. Ph.D. thesis, École Polytechnique.

Mulders, T., 2000. On short multiplications and divisions. Appl. Algebra Engrg. Comm. Comput. 11 (1), 69–88.

Schröder, M., 1997. Fast online multiplication of real numbers. In: STACS 97 (Lübeck). Vol. 1200 of Lecture Notes in Comput. Sci. Springer, Berlin, pp. 81–92.

Shoup, V., et al., 1990. NTL: a library for doing number theory. Version 6.0.0. Available from http://www.shoup.net/ntl/.

van der Hoeven, J., 1997. Lazy multiplication of formal power series. In: ISSAC '97. Maui, Hawaii, pp. 17–20.

van der Hoeven, J., 2002. Relax, but don't be too lazy. J. Symb. Comput. 34 (6), 479–542.

van der Hoeven, J., 2003. Relaxed multiplication using the middle product. In: Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 143–147 (electronic).

van der Hoeven, J., July 4–7 2004. The truncated Fourier transform and applications. In: Gutierrez, J. (Ed.), Proc. ISSAC 2004. Univ. of Cantabria, Santander, Spain, pp. 290–296.

van der Hoeven, J., 2007. New algorithms for relaxed multiplication. J. Symbolic Comput. 42 (8), 792–802.

van der Hoeven, J., 2014. Faster relaxed multiplication. In: Proc. ISSAC 2014. To appear.

von zur Gathen, J., Gerhard, J., 2003. Modern Computer Algebra, 2nd Edition. Cambridge University Press, Cambridge.