Generic reductions for in-place polynomial multiplication

Pascal Giorgi, Bruno Grenet, Daniel S. Roche

Generic reductions for in-place polynomial multiplication

Pascal Giorgi[*], Bruno Grenet[*]and Daniel S. Roche[†]

February 6, 2019

Abstract

The polynomial multiplication problem has attracted considerable attention since the early days of computer algebra, and several algorithms have been designed to achieve the best possible time complexity. More recently, efforts have been made to improve the space complexity, developing modified versions of a few specific algorithms to use no extra space while keeping the same asymptotic running time.

In this work, we broaden the scope in two regards. First, we ask whether an arbitrary multiplication algorithm can be performed in-place generically. Second, we consider two important variants which produce only part of the result (and hence have less space to work with), the so-called middle and short products, and ask whether these operations can also be performed in-place.

To answer both questions in (mostly) the affirmative, we provide a series of reductions starting with any linear-space multiplication algorithm. For full and short product algorithms these reductions yield in-place versions with the same asymptotic time complexity as the out-of-place version. For the middle product, the reduction incurs an extra logarithmic factor in the time complexity only when the algorithm is quasi-linear.

Keywords— arithmetic, polynomial multiplication, in-place algorithm, self reduction

1 Introduction

1.1 Polynomial multiplication

Polynomial multiplication is a fundamental problem in mathematical algorithms. It forms the basis (and key bottleneck) for other fundamental problems such

[*]LIRMM, Université de Montpellier, CNRS,Montpellier, France. (`firstname.lastname@lirmm.fr`)

[†]United States Naval Academy, Annapolis, Maryland, USA. `roche@usna.edu`

as division with remainder, GCD computation, evaluation/interpolation, resultants, factorization, and structured linear algebra (see, e.g., [9, §8–15] and [3, §2–7,10,12]).

As such, significant effort has gone to improving the time to multiply two size-n polynomials, most notably Karatsuba's algorithm [16], Toom-Cook multiplication [8], and Schönhage-Strassen [21]; more recent results have improved the complexity further but have not yet seen wide adoption in practice [6, 13].

1.2 Space complexity

After minimizing the runtime, an important question both in theory and in practice is how much extra *space* these algorithms require. While the classical algorithm can be made to use only a constant number of temporary values, all the faster algorithms mentioned above require $O(n)$ space to multiply two size-n polynomials. In fact, proven time-space trade-offs in the algebraic circuit and branching program models indicate that space at least polynomial in n is required for any sub-quadratic multiplication algorithm [20, 1].

But in a model where the output space admits both random writes *and* reads, these time-space lower bounds can be broken. [19] developed a variant of Karatsuba's algorithm using only $O(\log n)$ space. Later, an FFT-based multiplication algorithm using $O(n \log n)$ time and constant space was developed for the case that the coefficient ring contains a suitable root of unity [14]. Space-saving versions of Karatsuba's algorithm can also be found in [23, 5, 22, 7].

1.3 Short and middle products

Besides the usual *full product* computation, two other variants have also been extensively studied: the *short product* which truncates the output to the first n terms, and the *middle product* which truncates the result on both ends. These variants are important especially for power series, and specific variants of Karatsuba's algorithm and others have been developed, usually gaining a constant factor compared to a full product followed by a truncation [10, 18, 11, 12].

[4] shows that the middle product can be viewed essentially as the reverse of a full product and in the same space. However, in our model which uses the space of the output as temporary working space, this reversal implies that the inputs must also be destroyed for an in-place middle product. In some sense it would not be surprising if middle and short products were more difficult in our setting, as the truncated size of the output essentially limits the working space of the algorithm.

1.4 Our work

In this paper, we develop reductions which can transform any multiplication algorithm which uses $O(n)$ extra space into full, short, and middle product algorithms which use only $O(1)$ extra space. The time complexity for full and short product is the same as that of the original, while that for middle product incurs an additional $\log n$ factor.

This improves the $O(\log n)$ space of the most space-constrained Karatsuba algorithm [19], and implies for the first time: in-place versions of Toom-Cook multiplication; in-place FFT-based multiplication even when the ring does not contain a root of unity; in-place subquadratic short product algorithms; and in-place middle product algorithms which do not overwrite their inputs.

We begin by carefully stating our space complexity model and then defining the multiplications problems in Sections 2 and 3. A few easier but important reductions and equivalences are presented next in Section 4, followed by the critical reductions in Section 5 which prove our main results.

2 Complexity model

We use the model of an *algebraic-RAM* that is equipped with two kinds of registers: the standard registers store integers as in the classical Word-RAM model, whereas the *algebraic registers* store elements from the base field $\mathbb{K}$ of coefficients. As in Word-RAM, we assume that the standard registers can store integers of size $O(\log n)$ where n is the number of coefficients in the inputs.

Word-RAM machines are a classical model in computational complexity, in particular for *fine-grained complexity* that classifies the difficulty of polynomial-time problems [24]. We use it in order to distinguish between the space needed to store indices (that is thus hidden in the standard registers) from the space needed to store elements from the base field.

Time complexity As mentioned, we use the number of arithmetic operations as the time complexity measure since the cost of the operations on indices is negligible with respect to arithmetic operations. Formally, we assume that any ring operation on the algebraic registers has cost 1.

Space complexity We divide the registers into three categories: the input space is made of the (algebraic) registers that store the inputs, the output space is made of the (algebraic) registers where the output must be written, and the work space is made of (algebraic and non-algebraic) registers that are used as extra space during the computation. The space complexity is then the maximum number of work registers used simultaneously during the computation. An algorithm is said to be "in-place" if its space complexity is $O(1)$, and "out-of-place" otherwise.

One can then distinguish different models depending on the read/write permissions on the input and output registers:

1. Input space is read-only, output space write-only;
2. Input space is read-only, output space is read/write;
3. Input and output spaces are both read/write.

The first model is the classical one from complexity theory [2]. Despite its theoretical interest, it does not reflect low-level computation where output is typically in some DRAM or Flash memory on which reading is no more costly than writing. Furthermore, polynomial multiplication here has a quadratic lower bound for time times space [1], limiting the possibility for meaningful improvements.

The second model has been used in the context of in-place polynomial multiplication [19, 14]. This is a very reasonable model since it matches the paradigm of parallel computing with shared memory. This is the model in which we develop our algorithms.

The third model has been used to provide a generic approach for preserving memory designing algorithms via the transposition principle [4]: Given an algorithm for a linear map with time complexity $t(n)$ and space complexity $s(n)$, the transposition principle yields an algorithm for the transposed linear map which has the same space complexity and time complexity $O(t(n))$ [4, Propositions 1 and 2]. However, the inputs are destroyed during the computation, which is problematic particularly for recursive algorithms that re-use their operands; we will not use this too-permissive model.

Notation The output space in our algorithms is denoted by $\mathtt{R}$ and registers are indexed from 0 to $n-1$. We write $\mathtt{R}_{[k..\ell[}$ to denote the registers of indices k to $\ell - 1$.

3 Polynomial multiplications

Define the *size* of a univariate polynomial as the number of coefficients in its (dense) representation; a polynomial of size n has degree at most $n-1$. Importantly, we allow zero padding: a size-n polynomial could have degree strictly less than $n-1$; the size indicates only how it is represented.

Let $f = \sum_{i=0}^{n-1} f_i X^i$ and $g = \sum_{i=0}^{n-1} g_i X^i$ be two size-n polynomials. Their product $h = fg$ is a polynomial of size $2n-1$, what we call a *balanced full product.* More generally, if f has size m and g has size n, their product has size $m+n-1$. We call this case the *unbalanced full product* of f and g.

We now define precisely the short product, middle product, and half-additive full product.

Definition 3.1. *Let f and g be two size-n polynomials. Their* low short product *is the size-n polynomial defined as*

$$\mathsf{SP}_{\mathsf{lo}}(f,g) = (f \cdot g) \bmod X^n$$

and their high short product *is the size-$(n-1)$ polynomial defined as*

$$\mathsf{SP}_{\mathsf{hi}}(f,g) = (f \cdot g) \operatorname{quo} X^n.$$

The low short product is actually the meaningful notion of product for truncated power series. Note also that the definition of the high short product that we use implies that the result does not depend on all the coefficients of f and g. The rationale for this choice is to have the identity $fg = \mathsf{SP}_{\mathsf{lo}}(f,g) + X^n\mathsf{SP}_{\mathsf{hi}}(f,g)$.

Definition 3.2. *Let f and g be two polynomials sizes $n+m-1$ and n, respectively. Their* middle product *is the size-m made of the central coefficients of the product fg, that is*

$$\mathsf{MP}(f,g) = \big((f\cdot g) \operatorname{quo} X^{n-1}\big) \bmod X^m.$$

If $f = \sum_{i<n+m-1} f_iX^i$ and $g = \sum_{j<n} g_jX^j$, then

$$\mathsf{MP}(f,g) = \sum_{n-1\le i+j\le n+m-1} f_ig_jX^{i+j-n+1}.$$

The middle product, most commonly in the special case $n = m$, arises naturally in several algorithms manipulating polynomials or power series which are based on Newton's iteration, such as division or square root [11].

Further, the middle product is obtained by *Tellegen's transposition principle* from the full product algorithm [4, 11]. This implies that any full product algorithm yields an algorithm for the middle product of same time complexity. On the other hand, whether the transposition can be performed while also preserving the space complexity remains an open problem [15, 4] if one considers the inputs to be read-only.

Definition 3.3. *Let f and g be two polynomials of degree less than n, and h be a polynomial of degree less than $(n-1)$. The* (low-order) half-additive full product *of f and g given h is $\mathsf{FP}^+_{\mathsf{lo}}(f,g,h) = h + fg$. Similarly, their* high-order half-additive full product *is $\mathsf{FP}^+_{\mathsf{hi}}(f,g,h) = X^nh + fg$. An* in-place *half-additive full product algorithm is an algorithm computing a half-additive full product where h is initially stored in the output space.*

This variant of the full product which has a partially-initialized output space will be useful to derive other in-place algorithms.

3.1 Multiplications as linear maps

For ease of explanation, we will use the linear property of polynomial multiplications when an operand is fixed.

Let $f = \sum_{i=0}^{n-1} f_iX^i$ and $g = \sum_{i=0}^{n-1} g_iX^i$ be two size-n polynomials. If f is fixed, the product $h = fg$ can be described as a linear map from $\mathbb{K}^n$ to $\mathbb{K}^{2n-1}$. The matrix, denoted $\mathfrak{M}_{\mathsf{FP}(f)}$, for this map is to a Toeplitz matrix built from the coefficients of f, and the product fg corresponds to the following matrix-vector product:

$$\underbrace{\begin{pmatrix} f_0 & & \\ \vdots & \ddots & \\ f_{n-1} & & f_0 \\ & \ddots & \vdots \\ & & f_{n-1} \end{pmatrix}}_{\mathfrak{M}_{\mathsf{FP}(f)}} \times \underbrace{\begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{n-1} \end{pmatrix}}_{\vec{g}} = \underbrace{\begin{pmatrix} h_0 \\ h_1 \\ \\ \vdots \\ \\ h_{2n-1} \end{pmatrix}}_{\vec{h}} \tag{1}$$

where $\mathfrak{M}_{\mathsf{FP}(f)} \in \mathbb{K}^{(2n-1)\times n}$, $\vec{g} \in \mathbb{K}^n$ and $\vec{h} \in \mathbb{K}^{2n-1}$.

The low and high short products being defined as part of the result of the full product, their corresponding linear maps are endomorphisms of $\mathbb{K}^n$ and $\mathbb{K}^{n-1}$ respectively, given by submatrices of $\mathfrak{M}_{\mathsf{FP}(f)}$ as follows:

$$\underbrace{\begin{pmatrix} f_0 & & & \\ f_1 & \ddots & & \\ \vdots & \ddots & \ddots & \\ f_{n-1} & \dots & f_1 & f_0 \end{pmatrix}}_{\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}} \quad \underbrace{\begin{pmatrix} f_{n-1} & \dots & f_2 & f_1 \\ & \ddots & & f_2 \\ & & \ddots & \vdots \\ & & & f_{n-1} \end{pmatrix}}_{\mathfrak{M}_{\mathsf{SP}_{\mathsf{hi}}(f)}} \tag{2}$$

Finally, the middle product corresponds also to a linear map from $\mathbb{K}^n$ to $\mathbb{K}^m$ when the larger operand is fixed, given by the $m \times n$ Toeplitz matrix

$$\underbrace{\begin{pmatrix} f_{n-1} & f_{n-2} & \cdots & f_1 & f_0 \\ f_n & f_{n-1} & & f_2 & f_1 \\ \vdots & \vdots & & \vdots & \vdots \\ f_{n+m-2} & f_{n+m-3} & \cdots & f_{m-2} & f_{m-1} \end{pmatrix}}_{\mathfrak{M}_{\mathsf{MP}(f)}}.$$

4 Time and space preserving reductions

In this section, we compare the relative difficulties of the full product, the half-additive full product, the low and high short products, and the middle product, in the framework of time and space efficient algorithms. To this end, we define a notion of *time and space preserving reduction* between problems.

We say that a problem A is TISP-reducible to a problem B if, given an algorithm for B that has time complexity $t(n)$ and space complexity $s(n)$, one can deduce an algorithm for A that has time complexity $O(t(n))$ and space complexity $s(n) + O(1)$. We write $A \leq_{\mathsf{TISP}} B$ is A is TISP-reducible to B and $A \equiv_{\mathsf{TISP}} B$ if both $A \leq_{\mathsf{TISP}} B$ and $B \leq_{\mathsf{TISP}} A$. Note that the TISP-reduction is transitive.

The reduction we use can be defined using oracles and is an adaptation of the notion of *fine-grained reduction* [24, Definition 2.1] adapted to time-space fine-grained complexity classes [17].

Theorem 4.1. *Half-additive full products and short products are equivalent under* TISP*-reductions, that is*

$$\mathsf{FP}^+_{\mathsf{hi}} \equiv_{\mathsf{TISP}} \mathsf{FP}^+_{\mathsf{lo}} \equiv_{\mathsf{TISP}} \mathsf{SP}_{\mathsf{hi}} \equiv_{\mathsf{TISP}} \mathsf{SP}_{\mathsf{lo}}.$$

Furthermore, if SP *denotes either* $\mathsf{SP}_{\mathsf{lo}}$ *or* $\mathsf{SP}_{\mathsf{hi}}$*,*

$$\mathsf{FP} \leq_{\mathsf{TISP}} \mathsf{SP} \leq_{\mathsf{TISP}} \mathsf{MP}.$$

Proof. The equivalences $\mathsf{SP}_{\mathsf{hi}} \equiv_{\mathsf{TISP}} \mathsf{SP}_{\mathsf{lo}}$ and $\mathsf{FP}^+_{\mathsf{hi}} \equiv_{\mathsf{TISP}} \mathsf{FP}^+_{\mathsf{lo}}$ are proved below in Lemmas 4.3 and 4.4. The equivalence $\mathsf{SP} \equiv \mathsf{FP}^+$ (where SP denotes any of $\mathsf{SP}_{\mathsf{lo}}$ and $\mathsf{SP}_{\mathsf{hi}}$, and FP^+ any of $\mathsf{FP}^+_{\mathsf{lo}}$ and $\mathsf{FP}^+_{\mathsf{hi}}$) is proved in Section 4.2.

The reduction $\mathsf{FP} \leq_{\mathsf{TISP}} \mathsf{SP}$ simply amounts to the identity $\mathsf{FP}(f,g) = \mathsf{SP}_{\mathsf{lo}}(f,g) + X^n\mathsf{SP}_{\mathsf{hi}}(f,g)$. The reductions $\mathsf{SP} \leq_{\mathsf{TISP}} \mathsf{MP}$ and $\mathsf{FP} \leq_{\mathsf{TISP}} \mathsf{MP}$ follow from the following equalities where 0 denotes the zero polynomial stored in size n:

$$\begin{aligned} \mathsf{SP}_{\mathsf{lo}}(f,g) &= \mathsf{MP}(0 + X^n f, g), \\ \mathsf{SP}_{\mathsf{hi}}(f,g) &= \mathsf{MP}(f + X^n 0, g), \text{ and} \\ \mathsf{FP}(f,g) &= \mathsf{MP}(0 + X^n f + X^{2n} 0, g). \end{aligned}$$

Hence, one can compute the full product, the low and high short products of f and g simply by calling a middle product algorithm on f padded with zeroes and g. In our model of read-only inputs, an actual padding is not required. It is sufficient to use some kind of *fake padding* where the data structure storing f is responsible for returning 0 when needed. ☐

The relative order of difficulty $\mathsf{FP} \leq_{\mathsf{TISP}} \mathsf{SP} \leq_{\mathsf{TISP}} \mathsf{MP}$ makes intuitive sense based on the size of the output compared to the size of the inputs since the output can be used as work space: The full product maps $2n$ coefficients to $2n-1$ coefficients, the short products map $2n$ coefficients to n coefficients and the middle product maps $3n$ coefficients to n coefficients. In Section 5, we shall give a partial converse to $\mathsf{SP} \leq_{\mathsf{TISP}} \mathsf{MP}$: There exists a reduction from SP to MP which preserves space and either maintains the asymptotic complexity or increases it by a logarithmic factor.

4.1 Equivalences based on reverse polynomials

Definition 4.2. *The* size-n reversal *of a polynomial* f *is* $\mathrm{rev}_n(f) = X^{n-1}f(1/X)$.

We note that any algorithm whose input is a size-n polymial f can be turned into a new algorithm that computes the same function with input $\mathrm{rev}_n(f)$, simply by replacing a query to any coefficient with index i with one of index $n-i$, not affecting the number of ring operations.

Let us now prove that $\mathsf{SP}_{\mathsf{hi}} \equiv_{\mathsf{TISP}} \mathsf{SP}_{\mathsf{lo}}$.

Lemma 4.3. *Let* f *and* g *be two size-*n *polynomials. Then*

$$\mathsf{SP}_{\mathsf{hi}}(f,g) = \mathrm{rev}_{n-1}\left(\mathsf{SP}_{\mathsf{lo}}(\mathrm{rev}_{n-1}(f \operatorname{quo} X), \mathrm{rev}_{n-1}(g \operatorname{quo} X))\right).$$

Proof. Let $\tilde{f} = \mathrm{rev}_{n-1}(f \,\mathrm{quo}\, X)$ and $\tilde{g} = \mathrm{rev}_{n-1}(g \,\mathrm{quo}\, X)$. Then

$$\mathsf{SP}_{\mathsf{lo}}(\tilde{f}, \tilde{g}) = \sum_{\substack{0 \le i,j < n-1 \\ i+j < n-1}} f_{n-1-i} g_{n-1-j} X^{i+j},$$

whence

$$\mathrm{rev}_{n-1}\left(\mathsf{SP}_{\mathsf{lo}}(\tilde{f}, \tilde{g})\right) = \sum_{\substack{0 \le i,j < n-1 \\ i+j < n-1}} f_{n-1-i} g_{n-1-j} X^{n-2-(i+j)}.$$

One can change the indices of summation using $k = n-1-i$ and $\ell = n-1-j$. Then $n-2-(i+j) = k+\ell-n$ and the indices i and j such that $0 \le i+j < n-1$ are mapped to indices k and ℓ such that $2n-1 > k+\ell \ge n$. In other words,

$$\mathrm{rev}_{n-1}\left(\mathsf{SP}_{\mathsf{lo}}(\tilde{f}, \tilde{g})\right) = \sum_{\substack{0 < k,\ell \le n-1 \\ n \le k+\ell < 2n-1}} f_k g_\ell X^{k+\ell-n} = \mathsf{SP}_{\mathsf{hi}}(f, g). \qquad \square$$

Similarly, we can prove that $\mathsf{FP}^+_{\mathsf{hi}} \equiv_{\mathsf{TISP}} \mathsf{FP}^+_{\mathsf{lo}}$.

Lemma 4.4. *Let f and g be two size-n polynomials and h be a size-$(n-1)$ polynomial. Then*

$$\mathsf{FP}^+_{\mathsf{hi}}(f, g, h) = \mathrm{rev}_{2n-1}\left(\mathsf{FP}^+_{\mathsf{lo}}(\mathrm{rev}_n(f), \mathrm{rev}_n(g), \mathrm{rev}_{n-1}(h))\right).$$

Proof. Let $f^* = \mathrm{rev}_n(f)$, $g^* = \mathrm{rev}_n(g)$ and $h^* = \mathrm{rev}_{n-1}(h)$. First note that $\mathrm{rev}_{2n-1}(h^*) = X^n h$ by definition. Since $\mathrm{rev}_{2n-1}(f^* g^*) = \mathrm{rev}_n(f^*)\,\mathrm{rev}_n(g^*)$ we get that $\mathrm{rev}_{2n-1}(f^* g^* + h^*) = \mathrm{rev}_n(f^*)\,\mathrm{rev}_n(g^*) + \mathrm{rev}_{2n-1}(h^*) = fg + X^n h = \mathsf{FP}^+_{\mathsf{hi}}(f, g, h)$. $\square$

4.2 Equivalence between short products and half-additive full products

Reduction from SP to FP^+ Let f and g be two size-n polynomials and h be a size-$(n-1)$ polynomial. The half-additive full product $\mathsf{FP}^+_{\mathsf{lo}}(f, g, h)$ equals $fg + h$. Note that $fg = \mathsf{SP}_{\mathsf{lo}}(f, g) + X^n \mathsf{SP}_{\mathsf{hi}}(f, g)$. This already proves that the non-additive full product can be computed using algorithms for low and high short products. For the half-additive full products, it is sufficient to store an intermediate result in the free registers of the output space.

Assuming $\mathtt{R}_{[0..n-1[}$ holds the value of h, the following instructions reduces the computation of $\mathsf{FP}^+_{\mathsf{lo}}(f, g, h)$ to two short products plus $(n-1)$ additions.

```
1: $\mathtt{R}_{[n-1..2n-1[} \leftarrow \mathtt{SP}_{\mathtt{lo}}(f, g)$
2: $\mathtt{R}_{[0..n-1[} \leftarrow \mathtt{R}_{[0..n-1[} + \mathtt{R}_{[n-1..2n-2[}$
3: $\mathtt{R}_{n-1} \leftarrow \mathtt{R}_{2n-1}$
4: $\mathtt{R}_{[n..2n-1[} \leftarrow \mathtt{SP}_{\mathtt{hi}}(f, g)$
```

Reduction from FP^+ to SP Let f and g be polynomials of degree less than n. Splitting f and g by half such that $f = f_0 + X^{\lceil n/2 \rceil} f_1$ and $g = g_0 + X^{\lceil n/2 \rceil} g_1$, we have

$$\mathsf{SP}_{\mathsf{lo}}(f, g) = f_0 g_0 + X^{\lceil n/2 \rceil}(f_0 g_1 + f_1 g_0) \bmod X^n.$$

What is needed is the full product of f_0 and g_0, and the low short products of f_0 and g_1, and f_1 and g_0. Actually, since f_0 is larger than g_1 when n is odd (and g_0 larger than f_1), one only needs the short products $\mathsf{SP}_{\mathsf{lo}}(f_0^-, g_1)$ and $\mathsf{SP}(f_1, g_0^-)$ where $f_0^- = f \bmod X^{\lfloor n/2 \rfloor}$ and $g_0^- = g \bmod X^{\lfloor n/2 \rfloor}$.

To avoid any recursive call that would imply storing a call stack, we can actually use full products instead of short products: We first compute $f_0^- g_1 + f_1 g_0^-$ using a full product and a half-additive full product. Then we can forget about the higher order terms, and add $f_0 g_0$ to this sum using a second half-additive full product. The following instructions summarize this approach:

```
1: $\mathtt{R}_{[0..2\lfloor n/2 \rfloor - 1[} \leftarrow \mathtt{FP}(f_0^-, g_1)$            ▷ half-additivity not needed
2: $\mathtt{R}_{[0..2\lfloor n/2 \rfloor - 1[} \leftarrow \mathtt{FP}^+_{\mathtt{lo}}(f_1, g_0^-)$         ▷ erase higher part of $f_0^- g_1$
3: $\mathtt{R}_{[\lceil n/2 \rceil..n[} \leftarrow \mathtt{R}_{[0..\lfloor n/2 \rfloor[}$                  ▷ keep lower part of $f_0^- g_1 + f_1 g_0^-$
4: $\mathtt{R}_{[0..2\lceil n/2 \rceil - 1[} \leftarrow \mathtt{FP}^+_{\mathtt{hi}}(f_0, g_0)$
```

The correctness is clear. The complexity of the algorithm is the cost of three full products in degree approximately $n/2$: One non-additive full product in size $\lfloor n/2 \rfloor$ and two half-additive full products in size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, respectively.

As direct consequence of Lemmas 4.3 and 4.4, one obtains the same reductions to $\mathsf{SP}_{\mathsf{hi}}$ and from $\mathsf{FP}^+_{\mathsf{lo}}$ or $\mathsf{FP}^+_{\mathsf{hi}}$.

4.3 From half-additive full product to unbalanced full product

The unbalanced full product can be computed using any algorithm for the (balanced) full product. Nevertheless, the space complexity increases since intermediate results must be stored. Given an algorithm for the balanced full product of space complexity $s(n)$, one obtains an algorithm with space complexity $s(n) + (n-1)$ for the unbalanced full product. In this section, we prove that if the original full product algorithm is *half-additive*, the resulting unbalanced full product algorithm has the same space complexity.

Let f be a size-m polynomial and g be a size-n polynomial with $m > n$. Write $f = \sum_{k=0}^{\lceil m/n \rceil - 1} X^{kn} f_k$, where each sub-polynomial $f_0, \ldots, f_{\lceil m/n \rceil - 1}$ has size at most n. The computation of $f \cdot g$ reduces to the computations of each $f_k \cdot g$. The following instructions prove that using half-additivity, the intermediate results $f_k \cdot g$ can be computed directly in the output space.

```
1: $\mathtt{R}_{[\lceil m/n \rceil n..m+n[} \leftarrow \mathtt{FP}(f_{\lceil m/n \rceil}, g)$            ▷ using fake padding
2: for $k$ from $\lceil m/n \rceil - 1$ down to 0 do
3:     $\mathtt{R}_{[kn..(k+2)n-1[} \leftarrow \mathtt{FP}^+_{\mathtt{hi}}(f_k, g)$
```

Note that at step 1, the polynomial computed may have a larger size that what is needed, due to padding. Yet one can use without difficulty the lower part of the output space to store these additional useless coefficients, that are then erased at step 3.

The time complexity remains $\lceil m/n \rceil \mathsf{M}(n)$ where $\mathsf{M}(n)$ is the complexity of the half-additive full product.

5 In-place algorithms from out-of-place algorithms

In this section, we show how to obtain in-place algorithms from out-of-place algorithms. The theorem below summarizes the main results described in this section.

Theorem 5.1.
1. *Given a full product algorithm with time complexity $\mathsf{M}(n)$ and space complexity $\leq cn$, one can build an in-place algorithm for the half-additive full product with time complexity $\leq (2c+7)\mathsf{M}(n) + o(\mathsf{M}(n))$.*
2. *Given a (low or high) short product algorithm with time complexity $\mathsf{M}(n)$ and space complexity $\leq cn$, one can build an in-place algorithm for the same problem with time complexity $\leq (2c+5)\mathsf{M}(n) + o(\mathsf{M}(n))$.*
3. *Given a middle product algorithm with time complexity $\mathsf{M}(n)$ and space complexity $\leq cn$, one can build an in-place algorithm for the same problem with time complexity $\leq \mathsf{M}(n)\log_{\frac{c+1}{c+2}}(n) + O(\mathsf{M}(n))$ if $\mathsf{M}(n)$ is quasi-linear, and $O(\mathsf{M}(n))$ otherwise.*

Actually, our reductions work for any space bound $s(n) \leq O(n)$. Smaller space bounds yield better time bounds though we do not have a general expression in terms of $s(n)$. Yet sublinear space bounds still imply an increase of the time complexity by a multiplicative constant for full and short products.

Formally, we give self-reductions for the three problems. That is, we use an out-of-place algorithm for the problem as building block of our in-place version. The general idea is similar in the three cases. In a first step, we use the out-of-place algorithm to compute some part of the output, using the unused output space as temporary work space. Then a recursive call finishes the work. The (constant) amount of space needed in our in-place algorithms correspond the space needed to process the base cases.

Using the language of linear algebra, we aim to apply some specific matrix to a vector. The general construction we use consists in first applying the top or bottom rows of the matrix to the vector using the out-of-place algorithm, and applying the remaining rows using a recursive call (*cf.* Fig. 1). In the cases of full and short products, the diamond and triangular shapes of the corresponding matrices imply that the recursive call is made on two smaller inputs: For instance, to apply the first rows of a triangular matrix to a vector, one only needs to apply it to the first entries of the vector. For the middle product, the square shape imply that one input remains of the same size in the recursive call. This difference explains the difference in the time complexities in Theorem 5.1.

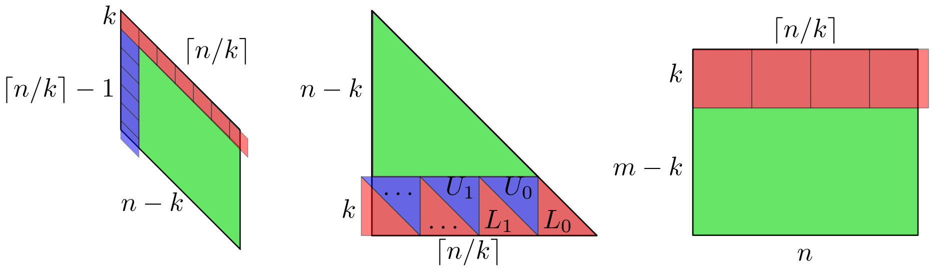

Figure 1: Tilings of the matrices $\mathfrak{M}_{\mathsf{FP}(f)}$ (left), $\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}$ (center) and $\mathfrak{M}_{\mathsf{MP}(f)}$ (right).

5.1 In-place full product algorithm

Our aim is to build an in-place (low-order) half-additive full product algorithm $\mathtt{iFP}^{+}_{\mathtt{hi}}$ based on an out-of-place full product algorithm $\mathtt{oFP}$ that has space complexity cn. That is, we are given two polynomials f and g of degree $< n$ in the input space and a polynomial h of degree $< n-1$ in the $(n-1)$ low-order registers of the output space $\mathtt{R}$ and we aim to compute $fg + h$ in $\mathtt{R}$. The algorithm is based on the tiling of the matrix $\mathfrak{M}_{\mathsf{FP}(f)}$ given in Fig. 1 (left).

For some $k < n$ to be fixed later, let $f = \hat{f}X^k + f_0$ and $g = \hat{g}X^k + g_0$ where $\deg f_0, \deg g_0 < k$. Then we have

$$h + fg = h + f_0 g + \hat{f} g_0 X^k + \hat{f}\hat{g}X^{2k}. \tag{3}$$

Recall that the output $\mathtt{R}$ has size $2n-1$ with its $n-1$ lowest registers containing h. Then equation (3) can be evaluated with the following three steps:

```
1: $\mathtt{R}_{[0..n+k-1[} \leftarrow h + f_0 g$
2: $\mathtt{R}_{[k..n+k-1[} \leftarrow \mathtt{R}_{[k..n+k-1[} + \hat{f} g_0$
3: $\mathtt{R}_{[2k..2n[} \leftarrow \mathtt{R}_{[2k..2n[} + \hat{f}\hat{g}$
```

The first two steps corresponds exactly to two *additive* unbalanced full products, that is unbalanced full products that must be added to some already filled output space. One can describe an algorithm $\mathsf{oFP}^{+}_{\mathtt{u}}$ for this task, based on a (standard) full product algorithm $\mathtt{oFP}$: If f has degree $< k$ and g has degree $< n$, $n > k$, we write $g = \sum_{i=0}^{\lceil n/k \rceil - 1} g_i X^{ki}$ with $\deg(g_i) < k$. Then $fg = \sum_i fg_i$: The algorithm computes the $\lceil n/k \rceil$ products fg_i in $2k-1$ extra registers and adds them to the output. If $\mathtt{oFP}$ has time complexity $\mathsf{M}(n)$ and space complexity cn, the time complexity of $\mathsf{oFP}^{+}_{\mathtt{u}}$ is $\lceil n/k \rceil\,(\mathsf{M}(k)+2k-1)$ and its space complexity $(c+2)k-1$.

The last step computes $h+fg$ and corresponds to a half-additive full product on inputs of degree $< n-k$, since only the $n-k-1$ first registers of $\mathtt{R}_{[2k..2n[}$ are filled: Indeed, $\deg(h + f_0 g + \hat{f} g_0 X^k) < n + k - 1$. This last step is thus a recursive call.

In order to make this algorithm run in place, k must be chosen so that the extra memory needed in the two calls to $\mathsf{oFP}^{+}_{\mathtt{u}}$ fits exactly in the unused part of R. This is the case when

$$(c+2)k - 1 \leq 2n - 1 - (n + k - 1)$$

which gives $k \leq \frac{n+1}{c+3}$. The resulting algorithm is formally depicted below.

Algorithm 1 $\texttt{iFP}^{+}_{\texttt{hi}}\texttt{\_from\_oFP}$

Input: f and g of degree $< n$ in the input space, h of degree $< n-1$ in the output space R

Output: R contains $fg + h$

Required alg.: A full product algorithm oFP with space complexity $\leq cn$

```
1: if $n < c + 2$ then
2:     $\mathtt{R} \leftarrow \mathtt{R} + fg$                                   ▷ using a naive algorithm
3: else
4:     $k \leftarrow \lfloor (n+1)/(c+3) \rfloor$
5:     $\mathtt{R}_{[0..n+k-1[} \leftarrow \mathsf{oFP}^{+}_{\mathtt{u}}(h, f_0, g)$          ▷ work space: $\mathtt{R}_{[n+k-1..2n[}$
6:     $\mathtt{R}_{[k..n+k-1[} \leftarrow \mathsf{oFP}^{+}_{\mathtt{u}}(h + f_0 g, f, g_0)$          ▷ same work space
7:     $\mathtt{R}_{[2k..2n[} \leftarrow \mathtt{iFP}^{+}_{\mathtt{hi}}\mathtt{\_from\_oFP}(f \operatorname{quo} X^k, g \operatorname{quo} X^k)$
```

Complexity analysis The algorithm uses two calls to $\mathsf{oFP}^{+}_{\mathtt{u}}$ with inputs of sizes (k, n) and $(n-k, k)$ respectively. The total complexity amounts to $\lceil n/k \rceil\, \mathsf{M}(k) + (\lceil n/k \rceil] - 1)\mathsf{M}(k) + 2(\lceil n/k \rceil - 1)(2k-1)$ plus a recursive call in size $n - k$. Let $T(n)$ be the complexity of $\mathtt{iFP}^{+}_{\mathtt{hi}}$, we thus have

$$T(n) = T(n-k) + (2\lceil n/k \rceil - 1)\left[\mathsf{M}(k) + (2k-1)\right].$$

Note that k depends upon n, this implies that the analysis must be done without k. Since $k = \lfloor (n+1)/(c+3) \rfloor$, $\lceil n/k \rceil \leq c+4$ for $n \geq (c+2)(c+4)$. Therefore,

$$T(n) \leq T\left(\frac{c+2}{c+3}(n+1)\right) + (2c+7)\left[\mathsf{M}\left(\frac{n+1}{c+3}\right) + 2\frac{n}{c+3} - \frac{c+1}{c+3}\right].$$

Using Corollary 5.6, we conclude that $T(n) \leq (2c+7)\mathsf{M}(n) + o(\mathsf{M}(n))$.

5.2 In-place short product algorithm

Our goal is to describe an in-place (low) short product algorithm based on an out-of-place one, based on the tiling of $\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}$ depicted on Fig. 1 (center). Let $f = \sum_{i=0}^{n-1} f_i X^i$ and $g = \sum_{i=0}^{n-1} g_i X^i$, and let $h = \sum_{i=0}^{n-1} h_i X^i = \mathsf{SP}_{\mathsf{lo}}(f, g)$. The idea is to fix some $k < n$ and to have two phases. The first phase corresponds to the bottom k rows of $\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}$ and computes h_{n-k} to h_{n-1} using the out-of-place algorithm on smaller polynomials. The second phase corresponds to the top $(n-k)$ rows and is a recursive call to compute h_0 to h_{n-k-1}: Indeed, $h \bmod X^{n-k} = \mathsf{SP}_{\mathsf{lo}}(f \bmod X^{n-k}, g \bmod X^{n-k})$.

For the second phase, we remark that the bottom k rows can be tiled by $\lceil n/k \rceil$ lower triangular matrices (denoted $L_0, \ldots, L_{\lceil n/k \rceil - 1}$ from the right to the left), and $\lceil n/k \rceil - 1$ upper triangular matrices (denoted $U_0, \ldots, U_{\lceil n/k \rceil - 2}$). One can identify the matrices L_i and U_i as matrices of some low and high short products. More precisely, the coefficients that appear in the lower triangular matrix L_i are the coefficients of degree ki to $k(i+1)-1$ of f. Thus, $L_i = \mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f_{ki,k(i+1)})}$ where $f_{ki,k(i+1)} = \sum_{j=ki}^{k(i+1)-1} f_j X^{j-ki}$. Similarly, $U_i = \mathfrak{M}_{\mathsf{SP}_{\mathsf{hi}}(f_{ki,k(i+1)})}$. The matrices $L_{\lceil n/k \rceil - 1}$ and $U_{\lceil n/k \rceil - 2}$ must be padded if k does not divide n. Altogether, this proves that this part of the computation reduces to $\lceil n/k \rceil$ low short products and $\lceil n/k \rceil - 1$ high short products, in size k.

In order for this algorithm to actually be in place, k must be small enough. If the out-of-place short product algorithm uses ck extra space, since we also need k free registers to store the intermediate results, k must satisfy $n-k \geq (c+1)k$, that is $k \leq n/(c+2)$.

Algorithm 2 `iSP_lo_from_oSP`

Input: f and g of degree $< n$

Output: R contains $\mathsf{SP}_{\mathsf{lo}}(f,g)$

Required alg.: Two short product algorithms $\mathtt{oSP_{lo}}$ and $\mathtt{oSP_{hi}}$ with space complexity $\leq cn$

```
1: if $n < c+2$ then
2:     $\mathtt{R} \leftarrow \mathsf{SP}_{\mathsf{lo}}(f,g)$                          ▷ using a naive algorithm
3: else
4:     $k \leftarrow \lfloor n/(c+2) \rfloor$
5:     for $i = 0$ to $\lceil n/k \rceil - 1$ do                  ▷ work space: $\mathtt{R}_{[0..n-k[}$
6:         $\mathtt{R}_{[n-k..n[} \mathrel{+}= \mathtt{oSP_{lo}}(f_{ki,k(i+1)}, g_{n-k(i+1),n-ki})$
7:     for $i = 0$ to $\lceil n/k \rceil - 2$ do                  ▷ same work space
8:         $\mathtt{R}_{[n-k..n[} \mathrel{+}= \mathtt{oSP_{hi}}(f_{ki,k(i+1)}, g_{n-k(i+2),n-k(i+1)})$
9:     $\mathtt{R}_{[0..n-k[} \leftarrow \mathtt{iSP_{lo}\_from\_oSP}(f \bmod X^{n-k}, g \bmod X^{n-k})$
```

Complexity analysis The algorithm performs $\lceil n/k \rceil$ low short products and $\lceil n/k \rceil - 1$ high short products plus one recursive call in size $n-k$. Let $\mathsf{M}(k)$ be the complexity of a low short product algorithm. Then the high short product can be computed in time $\mathsf{M}(k-1)$. Let $T(n)$ be the complexity of the recursive algorithm. Then $T(n) = \lceil n/k \rceil \mathsf{M}(k) + (\lceil n/k \rceil - 1)\mathsf{M}(k-1) + 2(\lceil n/k \rceil - 1)k + T(n-k)$ (the linear time is for the additions). Since $k = \lfloor n/(c+2) \rfloor$, $\lceil n/k \rceil \leq c+3$ for $n \geq (c+3)(c+2)$ and $n - k \leq \frac{c+1}{c+2}n + 1$. Thus,

$$T(n) \leq (c+3)\mathsf{M}\left(\frac{n}{c+2}\right) + (c+2)\mathsf{M}\left(\frac{n}{c+2} - 1\right) + 2n + T\left(\frac{c+1}{c+2}n + 1\right).$$

Using Corollary 5.6, this equation yields $T(n) \leq (2c+5)\mathsf{M}(n) + o(\mathsf{M}(n))$.

5.3 In-place middle product algorithm

To build an in-place middle product algorithm, we assume that we have an algorithm for the middle product that uses cn extra space to compute the middle product in size (n, m) (that is with inputs of degree $< n+m-1$ and $< n$, respectively).

The in-place algorithm is again based on the tiling given in Fig. 1 (right): The top k rows correspond to the matrix $\mathfrak{M}_{\mathsf{MP}(f \bmod X^k)}$ and the bottom $m-k$ rows to the matrix $\mathfrak{M}_{\mathsf{MP}(f \operatorname{quo} X^k)}$. The algorithm consists in computing $\mathfrak{M}_{\mathsf{MP}(f \bmod X^k)}\vec{g}$ using the out-of-place algorithm and then $\mathfrak{M}_{\mathsf{MP}(f \operatorname{quo} X^k)}\vec{g}$ using a recursive call.

To make this algorithm work in place, the value of k has to be adjusted so that the work space is large enough. The result of a middle product in size k has degree $< k$ and needs ck extra work space by hypothesis. Therefore, if $m-k \geq (c+1)k$, that is $k \leq m/(c+2)$, the computation can be performed in place.

Algorithm 3 `iMP_from_oMP`

Input: f and g of degree $< n+m-1$ and $< n$ respectively
Output: `R` contains $\mathsf{MP}(f, g)$
Required alg.: An out-of-place middle product algorithm `oMP` with space complexity $\leq cn$

```
1: if $m < c+2$ then
2:     $\mathtt{R} \leftarrow \mathtt{oMP}(f, g)$                          ▷ using a naive algorithm
3: else
4:     $k \leftarrow \lfloor m/(c+2) \rfloor$
5:     $\mathtt{R}_{[0..k[} \leftarrow \mathtt{oMP}(f \bmod X^{n+k}, g)$            ▷ work space: $\mathtt{R}_{[k..m[}$
6:     $\mathtt{R}_{[k..m[} \leftarrow \mathtt{iMP\_from\_oMP}(f \operatorname{quo} X^k, g)$        ▷ recursive call
```

Complexity analysis Let $\mathsf{M}(k)$ be the cost of an out-of-place balanced middle product algorithm. The cost of an unbalanced middle product is thus $\lceil n/k \rceil \mathsf{M}(k)$ for $k < n$. The in-place algorithm computes first a middle product using an out-of-place algorithm and then makes a recursive call on the remaining part. Note that n does not change during the algorithm and can be viewed as a large constant, while m is the parameter that varies. Then the cost of the algorithm verifies $T(m) \leq \lceil n/k \rceil \mathsf{M}(k) + T(m-k)$. Since $k = \lfloor m/(c+2) \rfloor$, $\lceil n/k \rceil < n(c+2)/(m-c-2)+1$ and $m-k \leq (c+1)m/(c+2)+1$. Furthermore, $\mathsf{M}(k) \leq m/n(c+2)\mathsf{M}(n)$, thus $\lceil n/k \rceil \mathsf{M}(k) \leq (m/(m-c-2)+m/n(c+2))\mathsf{M}(n)$. That is,

$$T(m) \leq \left(\frac{m}{n(c+2)} + \frac{c+2}{m-c-2} + 1\right)\mathsf{M}(n) + T\left(\frac{c+1}{c+2}m + 1\right).$$

Corollary 5.7 implies $T(n) \leq \mathsf{M}(n) \log_{\frac{c+2}{c+1}}(n) + O(\mathsf{M}(n))$ for $m = n$.

Improvement for non quasi-linear algorithms The extra logarithmic factor only occurs when $\mathsf{M}(n) = n^{1+o(1)}$. Suppose to the contrary that $\mathsf{M}(n) \leq \lambda n^\gamma$ for some $\gamma > 1$. The recurrence now reads $T(m) \leq \left(\frac{n(c+2)}{m-c-2} + 1\right) \lambda \left(\frac{m}{c+2}\right)^\gamma +$

$T(\frac{c+1}{c+2}m+1)$. We claim that there exist constants μ and ν such that $T(m) \le \mu m^{\gamma-1}n + \nu m^{\gamma} + o(m^{\gamma-1}n + m^{\gamma})$ and prove it by induction. Using the recurrence relation and the induction hypothesis,

$$T(m) \le \frac{\lambda n m^{\gamma-1}}{(c+2)^{\gamma-1}} + \frac{\lambda m^{\gamma}}{(c+2)^{\gamma}} + \mu \left(\frac{c+1}{c+2}\right)^{\gamma-1} m^{\gamma-1} n \\ + \nu \left(\frac{c+1}{c+2}\right)^{\gamma} m^{\gamma} + o(m^{\gamma-1}n + m^{\gamma}).$$

The result follows as soon as $(\lambda + \mu(c+1)^{\gamma-1})/(c+2)^{\gamma-1} \le \mu$ and $(\lambda + \nu(c+1)^{\gamma})/(c+2)^{\gamma} \le \nu$. We can thus fix

$$\mu = \frac{\lambda}{(c+2)^{\gamma-1} - (c+1)^{\gamma-1}} \text{ and } \nu = \frac{\lambda}{(c+2)^{\gamma} - (c+1)^{\gamma}}.$$

Finally, taking $m = n$, we conclude that $T(n) \le (\mu + \nu)\lambda n^{\gamma} + O(n^{\gamma-1})$.

Reduction from short products to middle product The middle product of f and g can be computed as the sum of the low short product of $f \operatorname{quo} X^n$ with g and the high short product of $f \bmod X^n$ with g. Yet this reduction does not preserve the space complexity since one needs to store the results of the two short products in two zones of size n before summing them. Actually, the reduction given above from `oMP` to `iMP` can easily be adapted to a reduction from SP to MP that is space-preserving. Yet, the complexity also worsens with a logarithmic factor. Thus, we cannot conclude that $\mathsf{MP} \le_{\mathsf{TISP}} \mathsf{SP}$.

5.4 Resolution of recurrences

Lemma 5.2. *Let $T(n)$ be a function satisfying $T(n) \le f(n) + T(\lfloor \alpha n + \beta \rfloor)$ for some $\alpha < 1$. Then*

$$T(n) \le T(\lfloor n_K \rfloor) + \sum_{i=0}^{K-1} f(n_i)$$

where $n_i = \alpha^i n + \beta \frac{1-\alpha^{i+1}}{1-\alpha}$ and $K \le \log_{1/\alpha}(n)$.

Proof. Let $T(x) = T(\lfloor x \rfloor)$ for non integral x. By definition of n_i, $n = n_0$ and $T(n_i) \le f(n_i) + T(n_{i+1})$. Then by recurrence, $T(n) \le T(n_{i+1}) + \sum_{j=0}^{i} f(n_i)$. □

Lemma 5.3. *Let $n_i = \alpha^i n + \beta \frac{1-\alpha^{i+1}}{1-\alpha}$. Then*

$$\sum_{i=0}^{K-1} n_i \le \frac{n + \beta K}{1-\alpha}.$$

Proof. Since $\sum_{i=0}^{K-1} \alpha^i = (1-\alpha^K)/(1-\alpha)$ and $1-\alpha > 0$, $\sum_i \alpha^i n \le n/(1-\alpha)$. Then, $\sum_i (1-\alpha^{i+1})/(1-\alpha) = K/(1-\alpha) + (\alpha^{K+1} - \alpha)/(1-\alpha)^2 \le K/(1-\alpha)$ since $\alpha^{K+1} < \alpha$. □

Lemma 5.4. *Let $n_i = \alpha^i n + \beta\frac{1-\alpha^{i+1}}{1-\alpha}$. Then*

$$\sum_{i=0}^{K-1} \frac{1}{n_i - \beta/(1-\alpha)} = \frac{\alpha(\alpha^{-K}-1)}{(1-\alpha)n - \alpha\beta}.$$

Proof. Since $n_i = \alpha^i(n - \beta\alpha/(1-\alpha)) + \beta/(1-\alpha)$, $n_i - \beta/(1-\alpha)$ is a multiple of α^i. Thus,

$$\sum_{i=0}^{K-1} \frac{1}{n_i - \beta/(1-\alpha)} = \frac{1}{n - \beta\alpha/(1-\alpha)} \sum_{i=0}^{K-1} \alpha^{-i}.$$

Then, $\sum_i \alpha^{-i} = (1-\alpha^{-K})/(1-1/\alpha) = \alpha(\alpha^{-K}-1)/(1-\alpha)$, and $\sum_i 1/(n_i - \beta/(1-\alpha)) = \alpha(\alpha^{-K}-1)/((1-\alpha)n - \alpha\beta)$. □

Lemma 5.5. *If $\mathsf{M}(n)/n$ is non-decreasing, and $n_i = \alpha^i n + \beta(1-\alpha^{i+1})/(1-\alpha)$ for some $\alpha < 1$, then*

$$\sum_{i=0}^{K-1} \mathsf{M}(\lambda n_i + \mu) = \frac{\lambda}{1-\alpha}\mathsf{M}(n) + o(\mathsf{M}(n))$$

for $K \le \log_{1/\alpha}(n)$ and any λ and μ such that $\lambda n_i + \mu \le n$ for all n_i.

Proof. Since $\mathsf{M}(n)/n$ is non-decreasing, $M(\lambda n_i + \mu) \le \frac{\lambda n_i + \mu}{n}\mathsf{M}(n)$. Therefore, $\sum_i \mathsf{M}(\lambda n_i + \mu) \le \mathsf{M}(n)/n \sum_i \lambda n_i + \mu$. By Lemma 5.3, $\sum_i \mathsf{M}(\lambda n_i + \mu) \le \lambda\mathsf{M}(n)/(1-\alpha) + \lambda\beta K\mathsf{M}(n)/n(1-\alpha) + \mu K\mathsf{M}(n)/n$. Since $K = O(\log n)$, $K\mathsf{M}(n)/n = o(\mathsf{M}(n))$. □

Corollary 5.6. *Let $T(n) \le \sum_k a_k\mathsf{M}(\lambda_k n + \mu_k) + bn + c + T(\alpha n + \beta)$ with $\alpha < 1$ and $\lambda_k n + \mu_k < n$ for all k. Then*

$$T(n) \le \sum_k \frac{a_k\lambda_k}{1-\alpha}\mathsf{M}(n) + \frac{bn}{1-\alpha} + o(\mathsf{M}(n)).$$

The linear term is negligible but if $\mathsf{M}(n) = O(n)$.

Proof. By Lemma 5.2, $T(n) \le T(n_K) + \sum_i f(n_i)$ with n_i defined as in the lemma and $f(n) = \sum_k a_k\mathsf{M}(\lambda_k n + \mu_k) + bn + c$. Then

$$\begin{aligned}
\sum_{i=0}^{K-1} f(n_i) &= \sum_k a_k \sum_{i=0}^{K-1} \mathsf{M}(\lambda_k n_i + \mu_k) + b\sum_{i=0}^{K-1} n_i + Kc \\
&\le \sum_k a_k\left(\frac{\lambda_k}{1-\alpha}\mathsf{M}(n) + o(\mathsf{M}(n))\right) + b\frac{n+\beta K}{1-\alpha} + Kc \\
&= \sum_k \frac{a_k\lambda_k}{1-\alpha}\mathsf{M}(n) + \frac{bn}{1-\alpha} + o(\mathsf{M}(n))
\end{aligned}$$

since $K = o(\mathsf{M}(n))$ and the sum over k is of fixed size. □

Corollary 5.7. *Let $T(m) \le (\lambda m/n + \mu/(m - \frac{1}{1-\alpha}) + 1)\mathsf{M}(n) + T(\alpha m + 1)$ with $\alpha < 1$ and $m \le n$. Then for $m = n$,*

$$T(n) \le \mathsf{M}(n)\log_{1/\alpha}(n) + \frac{\lambda + \mu\alpha}{1-\alpha}\mathsf{M}(n) + o(\mathsf{M}(n)).$$

Proof. By Lemma 5.2,

$$T(m) \le T(m_K) + \mathsf{M}(n)\sum_i \left(\frac{\lambda m_i}{n} + \frac{\mu}{m_i - 1/(1-\alpha)} + 1\right)$$

where $m_i = \alpha^i m + (1-\alpha^{i+1})/(1-\alpha)$. By Lemma 5.3, $\sum_i m_i \le (m+K)/(1-\alpha)$ and by Lemma 5.4, $\sum_i 1/(m_i - \frac{1}{1-\alpha}) \le \alpha^{-K+1}/((1-\alpha)m - \alpha)$. Altogether,

$$T(m) \le T(m_K) + K\mathsf{M}(n) + \frac{\lambda(m+K)}{n(1-\alpha)}\mathsf{M}(n) + \frac{\mu\alpha}{1-\alpha}\cdot\frac{(1/\alpha)^K}{m - \alpha/(1-\alpha)}\mathsf{M}(n).$$

If we plug $K = \log_{1/\alpha}(m)$ and fix $m = n$, we get

$$T(n) \le T(n_K) + \mathsf{M}(n)\log_{1/\alpha} n + \frac{\lambda + \mu\alpha}{1-\alpha}\mathsf{M}(n) + o(M(n)).$$

□

6 Perspectives

We have presented algorithms for polynomial multiplication problems which are efficient in terms of both time and space. Our results show that any algorithm for the full and short products of polynomials can be turned into another algorithm with the same asymptotic time complexity while using only $O(1)$ extra space. We obtain similar results for the middle product but only proved it for algorithms that do not have a quasi-linear time complexity. In the latter case, an increase of the time complexity by a logarithmic factor occurs. We provided analysis of our reductions that make their constants explicit. In particular, their values ensure that our reductions are practicable.

In a future work, we plan to address some remaining issues. By examining the constants in the already known algorithms, we can choose the algorithms to use as starting points of our reductions to optimize the complexity. For instance three variants of Karatsuba's algorithm with different time and space complexities are known [19, 23, 16]. Furthermore, it seems possible to improve on the complexity of low-space versions of Karatsuba's and Toom-Cook's algorithm, yielding faster in-place algorithms through our reductions. Another promising approach is to slightly relax the model of computation and work in model in which one can write on the input space as long as the original inputs are restored by the end of the computation. Preliminary results for Karatsuba's algorithm suggest that this could also yield a lower constant in the time complexity.

Finally, we have stated to explore the design of in-place algorithms for a broader range of problems of polynomials, such as division or evaluation/interpolation. The use of in-place middle and short products becomes crucial since one needs to avoid any increase in the size of the intermediate results.

Acknowledgements

This work was begun while the last author was graciously hosted by the LIRMM at the Université Montpellier.

This work was supported in part by the National Science Foundation under grants 1319994 (`https://www.nsf.gov/awardsearch/showAward?AWD_ID=1319994`) and 1618269 (`https://www.nsf.gov/awardsearch/showAward?AWD_ID=1618269`).

References

[1] K. Abrahamson. Time-space tradeoffs for branching programs contrasted with those for straight-line programs. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 402–409, 1986.

[2] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1st edition, 2009.

[3] A. Bostan, F. Chyzak, M. Giusti, R. Lebreton, G. Lecerf, B. Salvy, and E. Schost. *Algorithmes Efficaces en Calcul Formel*. 1.0 edition, Aug. 2017.

[4] A. Bostan, G. Lecerf, and E. Schost. Tellegen's principle into practice. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, ISSAC '03, pages 37–44, New York, NY, USA, 2003. ACM.

[5] R. Brent and P. Zimmermann. *Modern Computer Arithmetic*. Cambridge University Press, New York, NY, USA, 2010.

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

[7] Y. Cheng. Space-efficient karatsuba multiplication for multi-precision integers. *CoRR*, abs/1605.06760, 2016.

[8] S. A. Cook. *On the minimum computation time of functions*. PhD thesis, Harvard University, May 1966.

[9] J. v. z. Gathen and J. Gerhard. *Modern Computer Algebra (third edition)*. Cambridge University Press, 2013.

[10] G. Hanrot, M. Quercia, and P. Zimmermann. Speeding up the Division and Square Root of Power Series. Technical Report RR-3973, INRIA, 2000.

[11] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm i. *Applicable Algebra in Engineering, Communication and Computing*, 14(6):415–438, Mar 2004.

[12] G. Hanrot and P. Zimmermann. A long note on Mulders' short product. *Journal of Symbolic Computation*, 37(3):391–401, 2004.

[13] D. Harvey, J. van der Hoeven, and G. Lecerf. Faster polynomial multiplication over finite fields. *J. ACM*, 63(6):52:1–52:23, Jan. 2017.

[14] D. Harvey and D. S. Roche. An in-place truncated Fourier transform and applications to polynomial multiplication. In *ISSAC '10: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, pages 325–329, New York, NY, USA, 2010. ACM.

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

[16] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics-Doklady*, 7:595–596, 1963.

[17] A. Lincoln, V. Vassilevska Williams, J. R. Wang, and R. R. Williams. Deterministic Time-Space Trade-Offs for k-SUM. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[18] T. Mulders. On Short Multiplications and Divisions. *Applicable Algebra in Engineering, Communication and Computing*, 11(1):69–88, 2000.

[19] D. S. Roche. Space- and time-efficient polynomial multiplication. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, pages 295–302. ACM, 2009.

[20] J. Savage and S. Swamy. Space-time tradeoffs for oblivious integer multiplication. In H. Maurer, editor, *Automata, Languages and Programming*, volume 71 of *Lecture Notes in Computer Science*, pages 498–504. Springer Berlin / Heidelberg, 1979.

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

[22] C. Su and H. Fan. Impact of Intel's new instruction sets on software implementation of GF(2)[x] multiplication. *Information Processing Letters*, 112(12):497–502, 2012.

[23] E. Thomé. Karatsuba multiplication with temporary space of size $\leq$ n. online, 2002.

[24] V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings ICM*, 2018.