Efficient algorithms and implementation in exact linear algebra

Pascal Giorgi

Algorithmes et implantations efficaces en algèbre linéaire exacte

Pascal Giorgi

28 October 2019
Version: V 2.1.0

Université de Montpellier

Mémoire d'habilitation à diriger des recherches

Algorithmes et implantations efficaces en algèbre linéaire exacte

Pascal Giorgi

28 October 2019

Après avis des rapporteurs:

M.	George Labahn	Professeur	University of Waterloo, ON, Canada
M.	Grégoire Lecerf	Directeur de recherches	CNRS Palaiseau - LIX
M.	Emmanuel Thomé	Directeur de recherches	INRIA Nancy - Loria

Devant la commission d'examen constituée de:

M.	George Labahn	Professeur	University of Waterloo, ON, Canada
M.	Philippe Langlois	Professeur	Université de Perpignan - LIRMM
M.	Grégoire Lecerf	Directeur de recherches	CNRS Palaiseau - LIX
M.	Emmanuel Thomé	Directeur de recherches	INRIA Nancy - Loria
M.	Gilles Trombettoni	Professeur	Université de Montpellier - LIRMM
M.	Gilles Villard	Directeur de recherches	CNRS Lyon - LIP

Contents

Introduction

1

Premature optimization is the root of all evil (or at least most of it) in programming.

— **Donald Knuth**
Turing award conference 1974

Polynomial and matrices are mathematical objects that are central in various domains which use mathematics to solve problems. Our main interest in these objects falls down in the domain of computer algebra. There, the original goal is to provide effectiveness of algebra, meaning that we do want to provide computer programs that manipulate and compute with these objects. In particular, our work mainly focuses on a central part of computer algebra, namely exact linear algebra.

Numerical linear algebra usually serves as a key tool for applications coming mostly from continuous problems. It therefore relies on approximated computation through floating point numbers. At the opposite, exact linear algebra aims to solve problems without any approximation and then relies on more complex algebraic structures, e.g. integers, rational numbers, finite fields, etc. One of the most notorious success of exact linear algebra has been the use of Number Field Sieve algorithm [LH93] or Function Field Sieve algorithm [Adl94] that tackle the integer factorization problem or the discrete logarithm problem. Those serve as the foundation for the most famous asymmetric cryptographic protocols: Diffie-Hellman, RSA or El Gamal [RSA78; DH76; El 85]. In this context, a major breakthrough arose in 1986 with the work of Wiedemann [Wie86] who provides a novel way to solve a sparse linear system over a finite field with less operations than the classical Gaussian elimination. The technique, closed to the iterative methods in numerical linear algebra using Krylov subspaces [GO89], is to replace any factorization of the original matrix by its minimal polynomial. The salient idea of Wiedemann is to recover the minimal polynomial of the original matrix A from the one of the linearly generated sequence uA^iv for random vectors u and v. The method is of course probabilistic since both polynomials might differ, but for a given set of parameters it is proven unlikely. Later, in 1994 Coppersmith adapted in [Cop94] the latter method to incorporate parallelism by taking many random vectors at the same time and using a minimal generator of the linear generated matrix sequence UA^iV for U, V random matrices. This method is nowadays known as the block Wiedemann method.

Sketch of block Wiedemann method: For a field K and a matrix $A \in \mathsf{K}^{N\times N}$, the block Wiedemann algorithm aims at computing the matrix power series $S(X) = \sum_{i\geq 1} UA^iVX^i \in \mathsf{K}[[X]]^{n\times n}$ for random matrices $U^T, V \in \mathsf{K}^{N\times n}$ and to find a polynomial matrix $\Pi \in \mathsf{K}[X]^{n\times n}$ such that $S(X)\Pi(X) = R(X) + O(X^{2N/n})$ where $\deg \Pi, \deg R < N/n = d$. Note that Π can be computed through Hermite-Padé approximation [Vil97b], block Toeplitz linear system [Kal94] or generalized matrix version of Berlekamp-Massey algorithm [Cop94; Tho02a]. It is called the minimal matrix generating polynomial of $S(X)$ in [Vil97a]. Let $g(X) = g_0 + g_1X + \cdots + g_d \in \mathsf{K}[X]^n$ be a column of Π, for $0 \le i < N/n$ the following relation holds with high probability

$$A^{i+1}Vg_d + A^{i+2}Vg_{d-1} + \cdots + A^{i+d+1}Vg_0 = 0_n \in \mathsf{K}^n.$$

In particular, we have

$$A(Vg_d + AVg_{d-1} + A^2Vg_{d-2} + \cdots + A^dVg_0) = 0_n$$

which thus gives the expression of a right kernel vector of A as a linear combination of the powers of A projected on vectors V. In his original paper [Cop94], Coppersmith proved that linear combination is non-zero with a good probability only in the case of sufficiently large field. A few years later Villard provided a complete proof for any field, especially for the interesting case $\mathbb{F}_2$ [Vil97b; Vil97a].

As for Wiedemann's algorithm, the magic of this method is that A is never modified (no triangularization/diagonalization) and thus no filling of the matrix arises contrary to Gaussian elimination. Therefore, the cost is mostly dominated by the matrix-vector products with A. When A is sparse with $O(N)$ non zero-elements and assuming $n = O(1)$ this yields a method with a complexity of $O(N^2)$ operations in K improving upon the $O(N^3)$ complexity of Gaussian elimination. Note this complexity can be decreased by a factor of n when using parallelism, making it more suitable than Wiedemann for challenging computation. Yet another advantage of block Wiedemann is its space complexity of $O(N)$ while the fill-in of Gaussian elimination can possibly get the matrix to $O(N^2)$ non-zero entries. This lower memory requirement has democratized the use of such a method with sieving algorithms in cryptography which are known to generate huge matrices (few billions of rows/columns) with only few entries per rows (i.e. few hundreds) [Kle+12]. Of course, such matrices are out of reach with direct methods.

Note that this presentation of block Wiedemann is rather simplified and hides a lot of technical details, the objective being to provide a general idea of the method. A more complete description can be found in the original paper of Coppersmith [Cop94] or in the following references [Vil97b; Kal95; Tho02a].

1.1 Motivation

Since my thesis work in 2004, the block Wiedemann method inspired most of my research activity as it embraces a diversity of fundamental problems that are quite motivating and interesting by themselves. Of course, besides the goal of providing efficient software to solve sparse linear systems of equations over any field using this method, as we do in the LinBox project (`https://github.com/linbox-team`), we also looked at some of the main ingredients under the hood. These works have been conducted with or without correlation with the initial context of block Wiedemann. This for instance concerns:

- polynomial arithmetic: mainly for the basic operations such as multiplication, division, gcd; and its usage in the computation of minimal generators of a linearly generated sequences (i.e. Hermite-Padé approximants),

- elementary matrix operations: such as multiplication or factorization which are basic building blocks for matrix adaptation of the Berlekamp-Massey algorithm [Mas69],
- integer arithmetic: it serves commonly as a foundation for computing with prime fields on computers; typically the matrix coefficients involved within the block Wiedemann algorithm over large prime fields;
- practical consideration for improving algorithms: incorporating more parallelism, reducing constants in the complexities, reducing memory footprint;
- providing tools to efficiently verify computations or check provided implementations;
- and last but not least: efficient software implementations.

This manuscript intends to give a quick overview on what has been done so far, focusing on the salient ideas that make our contributions original and interesting. In particular, we will try to provide as much as possible of the context for every detailed part in the chapters 2 to 4 that cover some of our main interesting contributions. Besides, this first chapter will review in a more general way all my contributions since I've been an assistant professor at the University of Montpellier in 2007.

1.2 Useful notations and definitions

In this manuscript, we use the notation $\mathbb{F}_p$ to define the finite field with p elements for a given prime integer p. When dealing with generic fields, we will use the notation K, while we will use the notation R for commutative rings.

Time complexity As usual in computer algebra, we shall mainly focus on the number of arithmetic operations performed by our algorithms. To be more formal, the estimate of the complexity of algorithms will mainly fit the word Random Access Memory model (word-RAM) [Hag98] and we use an algebraic RAM complexity estimate, where each algebraic operations are assumed to have a cost of $O(1)$. We will extensively use in this manuscript the classical complexity function $\mathsf{M}(n)$. Let R be a ring, we will denote $\mathsf{M}(n)$ the time function for the multiplication of univariate polynomials of degree less than n, meaning that one can multiply two polynomials of $\mathsf{R}[X]$ of degree less than n using at most $\mathsf{M}(n)$ operations in R. Note that throughout the manuscript we will use n-size polynomial to refer to polynomials of degree less than n.

When dealing with integers, we will use a computation model similar to the one described in [GG13, Chapter 2] or [BZ10, Chapter 1.1]: we fix a base β, that can typically be a power of two such as $2, 2^{16}$ or 2^{32}, and we assume that all non-zero integers are written in this basis: such an integer a is represented by its coefficients $(a_0, \dots, a_{s-1})$, where each a_i in $\{0, \dots, \beta-1\}$ and a_{s-1} is non-zero, together with a sign bit, such that $a = \pm(a_0 + a_1\beta + \cdots + a_{s-1}\beta^{s-1})$. The coefficients a_i are referred to as *words*; the integer s is called the *length* of a, and denoted by $\lambda(a)$. Our complexity statements are then given in terms of the number of operations on words, counting each operation at unit cost (for the details of which operations are used, see [GG13, Chapter 2]). For simplicity, we assume that all words can also be read or written in time $O(1)$ such that our complexity estimates fit the standard word RAM model. We let $\mathsf{I} : \mathbb{N} \to \mathbb{N}$ be such that two integers a, b of length at most n can be multiplied in $\mathsf{I}(n)$ word operations, assuming the base-β expansions of a and b are given.

For both $\mathsf{M}(n)$ and $\mathsf{I}(n)$, it is assumed that they satisfy the super-linearity assumptions of [GG13, Chapter 8], that is $\mathsf{M}(n)/n$ and $\mathsf{I}(n)/n$ are non-decreasing. When necessary, we might use the time function $\mathsf{M}(m,n)$ and $\mathsf{I}(m,n)$ to express the time complexity of unbalanced multiplication of polynomials of size m and n, or integers of m and n words. In particular, assuming $m > n$ we have that $\mathsf{M}(m,n) \le \lceil m/n \rceil \mathsf{M}(n)$ and similarly $\mathsf{I}(m,n) \le \lceil m/n \rceil \mathsf{I}(n)$.

Similarly, we will use $\mathsf{MM}(n)$ to denote the time function for the multiplication of matrices in $\mathsf{R}^{n\times n}$, meaning that one can multiply two matrices in $\mathsf{R}^{n\times n}$ with at most $\mathsf{MM}(n)$ operations in R. A common assumption is to say that $\omega \in \mathbb{R}$ is an acceptable exponent for matrix multiplication problem if $\mathsf{MM}(n) = O(n^{\omega})$. It is clear that $2 \le \omega < 3$. We also use $\mathsf{MM}(m,n,k)$ to refer to the cost of matrix multiplication of $m \times k$ by $k \times n$ matrices, which is of course bounded by $O(mnk \min(m,n,k)^{\omega-3})$.

Another classical notation used in this manuscript is the Soft-Oh notation used to hide logarithmic factors in the complexity estimates, that is $\tilde{O}(n) = O(n(\log n)^{O(1)})$. As an abuse of notation we will use $\log(n)$ as the logarithm of n in base two.

Space complexity Another important notion we will use is the space complexity of an algorithm. For this purpose, we use a model that divides the memory 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.

While the first model is classical in computational complexity [AB09], it does not reflect low-level computation where output is typically in some Digital RAM or Flash memory on which reading is no more costly than writing. Therefore, we will mainly study space complexity of algorithms using either the second or third model, since they are more appropriate to our desire: efficiency in practice.

1.3 Main contributions

The following sections intend to give only a brief overview of our research approaches and the achieved contributions. We hope this short overview will provide a guideline through the different areas we've explored. We will provide further details on some of the contribution in the next chapters.

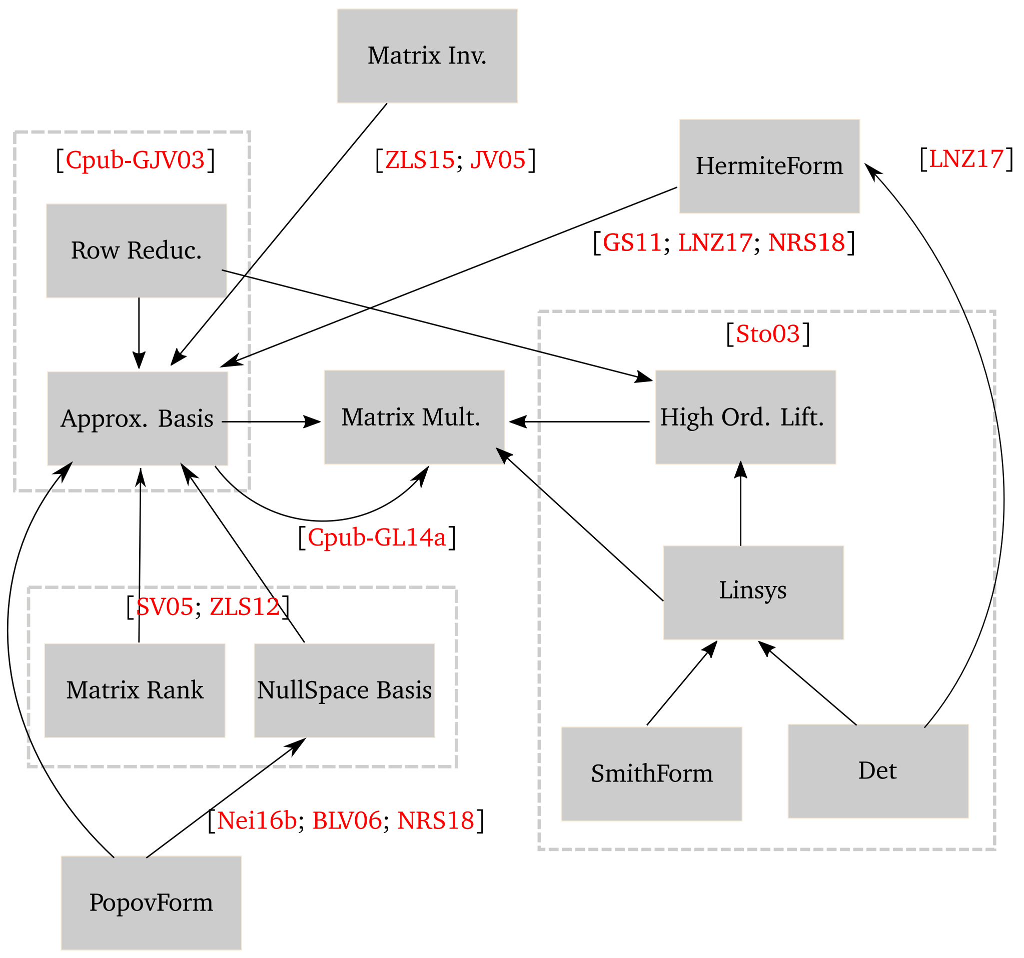
1.3.1 Reduction to efficient core algorithms

In computer algebra, the two main ingredients that are used by almost every fast algorithms are either fast arithmetic on polynomials or integers and fast matrix multiplication. Since the seminal work of Karatsuba in 1962 [KO63] who has shown that integer multiplication, as well as polynomial multiplication, can be done in subquadratic time, many other results lowered this complexity down to $\mathsf{M}(n) = O\tilde{}(n)$ and $\mathsf{I}(n) = O\tilde{}(n)$ [SS71; Sch77; CK91; HH19a; HH19b]. The same history can be found on the problem of matrix multiplication that has been proven for the first time to be subcubic time complexity in 1969 by Strassen [Str69]. Nowadays, the best result on matrix multiplication is given by Le Gall [Le 14] who proved $\mathsf{MM}(n) = O(n^{\omega})$ with $\omega < 2.3729$.

Following these important results many other problems have been reduced to polynomial (or integer) multiplication in order to improve their complexity. This is for instance the case with polynomial gcd or its integral analogue that are proven to have a cost of respectively $O(\mathsf{M}(n)\log(n))$ and $O(\mathsf{I}(n)\log(n))$ [Knu70; Sch71]. This kind of algorithmic reduction is even more relevant for linear algebra where most problems over a field are reduced to matrix multiplication, some of them being even equivalent to it: e.g. matrix inversion [Str69] or determinant [BS83]. One major difficulty for linear algebra in the last two decades has been to properly control expression swell over principal rings, e.g. ring of integers or univariate polynomials over a field. Indeed, the classical algebraic complexity model counts only operations on ring elements. However, such elements could be either a polynomial of degree n or a constant one. Hence, it is usual to require a finer complexity model where every operation on a word or on a coefficient counts. For a long time, it has been thought that finer complexity estimates for linear algebra would always incur a penalty that is linear in matrix dimension, compared to the pure algebraic version. In 2002, Storjohann broke this belief by providing algorithms for linear systems, integrality certification and determinant over $\mathsf{K}[X]$ with the same complexity as for matrix multiplication [Sto03]. Three years later, he extended these results to the integer case [Sto05]. Following this breakthrough, the last decades have seen a lot of improvements for linear algebra over integers or univariate polynomials. To do so, the problems have been reduced to matrix multiplication mostly with probabilistic algorithms. Some recent efforts have been made to make these reductions deterministic [LNZ17; ZLS15; Gup+12]. We refer to the following books and PhD or HDR manuscripts to learn more about this effective reductions in exact arithmetic and exact linear algebra [GG13; BZ10; Bos+17; Per14; Vil03; Zho12].

During my PhD thesis, we proved with Jeannerod and Villard [Cpub-GJV03] that the computation of minimal approximant bases (also called sigma bases), that are minimal basis of the $\mathsf{K}[X]$-module $\mathscr{A} = \left\{\mathbf{p} \in \mathsf{K}[X]^{1\times m} \mid \mathbf{pF} = 0 \bmod X^d\right\}$ for $\mathbf{F} \in \mathsf{K}[X]^{m\times m}$ and $d \in \mathbb{N}_{>0}$, reduces to matrix multiplication over $\mathsf{K}[X]$, meaning a complexity of $O\tilde{}(m^{\omega}d)$ operations in K. This result improved the important work of Beckerman and Labahn on fast computation for matrix-type Padé approximation [BL94] for some specific cases. Using [Vil97b; Kal95], our work makes it possible to compute minimal matrix generating polynomial in block Wiedemann within polynomial matrix multiplication cost, improving upon any other known methods [Cop94; Tho02a; Kal94]. We also provide in [Cpub-GJV03] a probabilistic algorithm for matrix row reduction over $\mathsf{K}[X]$ that reduces to matrix multiplication, extending the list of problems started by Storjohann in [Sto03]. This result has been fundamental as it still serves today as foundation for many fast algorithms in linear algebra over $\mathsf{K}[X]$, see Figure 1.1 below.

Fig. 1.1: Overview of algorithmic reductions in dense linear algebra over $\mathsf{K}[X]$



Since then, our research has been to make fast minimal approximant basis practicable and available in the software library LinBox (`https://github.com/linbox-team`). The LinBox library has been created in 1997 from an international effort between Canada, France and USA, under the impulse of T. Gautier, J.-L Roch, E. Kaltofen, B.D. Saunders and G. Villard. At the beginning, the aim of the project was to provide a generic library to solve exact linear algebra problems using blackbox method, as Wiedemann's one. From these early developments, the project evolved a lot and it has investigated many different areas of exact linear algebra, leading both to more efficient algorithms and to several libraries: FFLAS-FFPACK for dense linear algebra, Givaro for basic arithmetic and representation of the algebraic objects (prime fields, integers, rationals, polynomial rings, ...) and LinBox providing the main "generic/dedicated" solutions to dense, sparse and blackbox linear algebra problems: determinant, rank, Smith form, linear system solving, ...

In order to develop efficient code for minimal approximant basis, we first investigated with Dumas and Pernet, the effectiveness of the reduction to matrix multiplication for linear algebra problems over finite fields. For this purpose, we based our work on the LQUP matrix factorization [IMH82], a generalization of the classical LU factorization. In [Jpub-DGP08], we provide a detailed analysis of the constants in the reductions to matrix multiplication for basic dense linear algebra problems over finite field: LQUP, determinant, rank, inverse; and demonstrate that these reductions can be in practice as efficient as expected. Following the work in [DGP02] that uses numerical BLAS[1] libraries for speeding-up matrix multiplications over half word-size prime field, we provide in [Jpub-DGP08] the foundation of the FFLAS-FFPACK library to provide efficient dense linear algebra over that specific fields. In particular, our implementation relies on fast Strassen-Winograd [Str69] and it establishes its practical benefit even for matrices of moderate sizes. A side effect of this development has been to learn that floating point numbers operations have usually a better peak performance than its integer counterpart on current processors, see our slides at SIAM conference PP'14 [Ppub-Gio14]. The FFLAS-FFPACK library has been built following this remark and mainly focuses on primes up to 26-bits to ensure exact computing within the mantissa of floating point numbers. This library, available at (`https://github.com/linbox-team/fflas-ffpack`), serves today as an international reference for these specific computations. Software such as MAPLE, NTL, SAGE, FLINT, MATHEMAGIX have followed up on our work and either re-implement our solution or build a similar approach using integer (see NTL benchmarks explanations [2]). Section 4.2 presents the main ideas behind our contribution and the development that has been achieved so far.

In order to reach a larger class of prime fields, in particular the ones that are larger than a word-size, we study later the use of multi-modular approach and the Chinese Remainder theorem [GG13, Section 5.4] for integer matrix multiplication. Our approach was motivated by the performances on moderate size integers, i.e. few tens or hundreds of words, that could appear in discrete logarithm computation over finite fields or elliptic curves. In [MB72] it is shown that multi-modular reduction and reconstruction of an n-words integer with n moduli fitting a single word cost $O(\mathsf{I}(n)\log(n))$ word operations. Hence, one can reach a quasi-linear time complexity of $O\tilde{}(n)$ using fast integer multiplication. However, such fast algorithms are not relevant for moderate sizes as the hidden logarithmic factors incur too much overhead in practice, and the naive quadratic approach remains preferred. For matrix multiplication, the fact that simultaneous reductions or reconstructions are to be done leaves some room for doing precomputations and then improving the time complexity of the naive approach. In [Jpub-Dol+18], we propose an algorithm using precomputation that reduces the complexity from $O(n^3)$ word operations to $O(n^\omega)$, when n integers of n-words are considered to be converted to n single word moduli. Besides its theoretical interest, our method enables to benefit from efficiency of matrix multiplication libraries that optimally re-use data and minimize cache effect, allowing to reach almost the peak performance of the processors [GG08; Int07]. Section 3.3 presents the detail of our algorithm together with some experiments showing its superiority against few major libraries in computer algebra for integers of bit length up to 2^{17}.

Going back to minimal approximant basis, we've been asked by George Labahn in 2005 if it would be possible to provide an iterative variant of our `PM-Basis` algorithm given in [Cpub-GJV03]. We answered this question only in 2014 when supervising the postdoctoral studies of Romain Lebreton within the ANR project HPAC. There, our motivation was to design a fast online algorithm for minimal approximant basis. Online algorithms fit a computational model where inputs are discovered partially during the course of the algorithm. For instance when multiplying two polynomials f and g, only

[1]Basic Linear Algebra Subroutines defined in [Law+79]
[2]`https://www.shoup.net/ntl/benchmarks.pdf`

the first k coefficients of f and g are needed to compute the first k coefficients of fg. Basically, when writing the k-th coefficient of the output, online algorithms only have access to the first k coefficients of the inputs. When computing with polynomial series, this model implicitly allows the precision of the inputs to be increased as soon as a larger precision is required for the output, see [Leb12] for a more complete description. With block Wiedemann algorithm it appears that the matrix power series $S(X) = \sum_{i\leq 1} UA^i VX^i$ is usually precomputed to the maximal precision to be able to retrieve the minimal polynomial matrix Π. For generic matrices with low rank deficiency, as in cryptography, the general *a priori* bound of $2N/n + O(1)$ given in [Cop94] is tight and almost no superfluous coefficients of $S(X)$ are computed. However a significant rank deficiency makes this bound loose, which introduces a computational overhead. As observed in [KL99], when several zero discrepancies appeared consecutively in the matrix Berlekamp-Massey algorithm, it is with good probability that the computed Π is already correct. Of course this remark remains heuristic but it has been proved useful for early termination within the BenOr-Tiwari sparse interpolation algorithm [KL03]. Remark that similar early termination technique has been proven for the Lanczos algorithm over large field [EK97]. With Lebreton, we propose in [Cpub-GL14a] to exploit this early termination process and then to design an online algorithm for minimal approximant basis. Our goal was to design an algorithm that computes the minimal number of coefficients of the matrix power series when using early termination. Every prior work on fast variants for matrix Berlekamp-Massey algorithm uses a recursive divide-and-conquer scheme that needs to double the length of the sequence before to further pursue the last recursive branch. In that context, early termination can overestimate by a factor of two the precision of the series. Our online algorithm in [Cpub-GL14a] completely removes this overshooting. For doing this, we first provide an iterative variant of `PM-Basis` and then we describe an half-line[3] algorithm for polynomial middle product. There, we get an online algorithm `oPM-Basis` that achieves minimal precision for the series and having complexity similar to `PM-Basis` up to the extra logarithmic factor classical with the online model [Leb12]. Note this extra logarithmic factor is not problematic in block Wiedemann as the cost remains dominated by the computation of the series $S(X)$ itself. Our experimentation with `oPM-Basis` in the LinBox library demonstrated the complete removal of the staircase effect when early termination is used in block Wiedemann. Section 4.3 describes in more details this contribution.

1.3.2 Certification of results

In [Cpub-Dum+07] we have computed the rank of very large sparse matrices arising in algebraic K-theory in order to provide more credibility to a conjecture in number theory [Sou99]. Our approach was based on a block Wiedemann's rank algorithm [KS91; Tur06] and the computation extensively used `PM-Basis` code from LinBox. The larger matrix we could reach at that time was a $1\,911\,130 \times 1\,955\,309$ matrix whose rank is $1\ 033\,568$ over $\mathbb{F}_{65537}$ and the computation took 1050 CPU days. Note that we used only shell task parallelism for computing the sequence, not for the minimal approximant basis that have been run sequentially. The fact that block Wiedemann's rank algorithm was Monte-Carlo over finite fields [SSV04] raised the natural question to trust or not the computed result. Although the algorithm may have succeeded, our implementation could have been unsafe. Beside providing a Las Vegas algorithm for the

[3] A variant of the online model that assumes one operand to be known completely in advance.

rank over finite field, we were more generally interested to find a verification procedure for block Wiedemann and then for approximant bases computation.

Linear algebra operations are good candidates for designing fast verification algorithms since they often have a cost related to matrix multiplication while their input only uses quadratic space. This is also true with sparse matrices as time complexity is often an order of magnitude larger that the input size. The first example one may think of is linear system solving. Indeed, given a solution vector $x \in \mathsf{K}^n$ to a system $Ax = b$ defined by $A \in \mathsf{K}^{n\times n}$ and $b \in \mathsf{K}^n$, one can directly verify the correctness by checking the equations at a cost of $O(n^2)$ operations in K, or $O(n)$ is matrix A is sparse. Comparatively, solving the system with the fastest known algorithm costs $O(n^\omega)$ operations in K or $O(n^2)$ if A is sparse. Another famous result, due to Freivalds [Fre79], gives a method to verify matrix multiplication. Given matrices $A, B, C \in \mathsf{K}^{n\times n}$, the idea is to check $uC = (uA)B$ for a random row vector $u \in \{0, 1\}^{1\times n}$, rather than $C = AB$. This verification algorithm costs $O(n^2)$ and is *false-biased one-sided* Monte-Carlo (it is always correct when it answers "false"); the probability of error can be made arbitrarily small by picking several random vectors. In some cases, one may require an additional piece of data to be produced together with the output in order to prove the correctness of the result. For example, Farkas' lemma [Far02] certifies the infeasibility of a linear program thanks to an extra vector. Although the verification is deterministic in this example, the design of certificates that are verified by probabilistic algorithms opened a line of work for faster verification methods in linear algebra [KNS11; Kal+12; DK14; Dum+16b]. In this context, one of the main challenges is to design *optimal* certificates, that are the ones leading to a linear time verification. Furthermore, the time and space needed to produce the certificate must remain negligible. For instance, [DK14] provides an optimal certificate for the rank over finite fields, making all probabilistic iterative methods of Las Vegas type.

In fact, mots of these results extend the seminal work of Goldwasser, Kalai, Rothblum in 2008 on delegating computation [GKR08]. Indeed, Goldwasser *et. al* describe a generic method that automatically produces certificates that are verifiable in almost linear time for any problem (using a boolean circuit model) that belongs to the complexity class NC. One main drawback of their result is that generating the certificate is quite costly: cubic in the original time. In [Tha13], Thaler removes this time blow up when the problem is described with a sufficiently regular boolean circuit. He then achieves optimal certificates in our sense. While this last result applies for almost every problems in linear algebra, it carries some extra constant in the time complexity that are not satisfactory from a practical perspective. Another downside of these automatic verifications is that they only certify that a program runs properly. Therefore, if the description of the program is erroneous, they would fail to detect it. Our main interest here is twofold: first, to provide a verification procedure that minimizes the constant overhead from the original time complexity; second, to provide a way to check if an implementation of an algorithm returns the desired answer. In particular, we are interested in avoiding if possible the use of interactive proof as it might not satisfy our last need. Note that most dedicated certificates in linear algebra extensively use that concept, see [DK14; Dum+16b].

In this context, we have been able to prove in [Jpub-Gio18] that one can verify, without any certificate, the truncated and the middle product of two univariate polynomials. Our approach is based on expressing these operations as linear maps and to use a verification *à la* Freivalds as in [KS93]. Based on this result, we have been able with Neiger in [Cpub-GN18] to extend this result to polynomial matrices and to use it to provide an optimal certificate for verifying minimal approximant basis computation. Section 2.2 presents our result on polynomial products while Section 4.3.3 presents in more details our result for minimal approximant. One should notice that several advances for the certification of Wiedemann's algorithm has been done in [DKT15; Dum+16b] and its block variant should be reachable following our result.

1.3.3 Memory efficiency

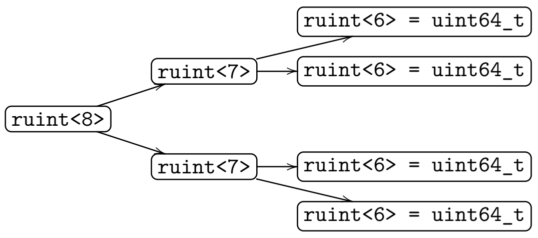
Another difficulty arising with the block Wiedemann algorithm is its memory requirement. While using a block of n vectors allows a n-fold parallelism, its memory requirement is increased by a factor n. This increase can really become a bottleneck when attacking computational challenges in cryptography that exhibit huge matrices and then need to use as much parallelism as possible. For instance, the world record for factoring a 768-bits RSA modulus [Kle+12] used a value of $n = 8$, incurring a memory peak in RAM of 1TB for computing the minimal matrix polynomial Π. Besides, the world record for discrete logarithm computation in a 768-bits prime field [Kle+17], used a value of $n = 16$ and it required an impressive amount of 8TB of memory. One of the main reason is due to the use of algorithms that rely on fast fast Fourier transform [CT65] or similar transforms to achieve quasi-linear time complexity. There, classical implementations only focus on performance and usually do not really care about minimizing memory. For instance, when a prime field does not contain a primitive root of unity, it is common to rather consider the problem over the integer and then use three-prime FFT [GG13, Section 8.3] to achieve the computation with a quasi-linear time complexity. Unfortunately, this approach could increase the memory requirement by a factor of 3. In order to decrease the memory pressure, new fast algorithms must be designed to explicitly manage their data in a more clever way. Few results have been proposed to improve the memory requirement for the products of size-n polynomials. In particular, [Tho02b] proposes a description of Karatsuba algorithm using only a temporary space of n. This has been improved later in [Roc09] where a new variant achieves only a space complexity of $O(\log(n))$. The latter article from Roche also provides a description of a FFT-based algorithm that is in-place, meaning only $O(1)$ extra-memory. Similar result have been proved in [HR10] for the more general variant of FFT that is call TFT "Truncated Fourier Transform" and that does not require polynomials to have their size to be a power of two. Note in both results it is assumed that the ring of coefficient embeds a proper primitive root of unity.

In order to progress on minimizing block Wiedemann memory, we have investigated the core operations that are used by our `PM-Basis` algorithm. From our previous work on dense linear algebra [Jpub-DGP08] and some refinement on memory for fast matrix multiplication [Boy+09], it is known that most dense linear operations can be done in-place, at least using Strassen-Winograd algorithm. Hence, we have rather investigated the polynomial operations of `PM-Basis` that are polynomial multiplication and polynomial middle product [HQZ04].

Following the work of Roche [Roc09], we have then sought to strengthen memory efficiency for these two operations. In order to provide more general results on memory efficiency we adopt the framework of algorithmic reductions. In particular, with Grenet and Roche, we propose in [Cpub-GGR19] to use self-reduction techniques to decrease the temporary space of any polynomial product algorithms. Our general idea is similar to the one in [Boy+09] for matrix multiplication, and it decomposes the problem in two sub-problems having their result not overlapping in the output space. There, we can solve the first sub-problem with classical algorithm using unused output space as temporary, and then use tail recursion to solve the second one. Thanks to this idea, we prove that any algorithm requiring a linear amount of extra memory can be turned into an algorithm that needs only $O(1)$ extra memory. Our results also show these reductions preserve the asymptotic time complexity for both classical multiplication and its truncated variant. In particular, we have that in-place full product costs $(2c+7)\mathsf{M}(n)+o(\mathsf{M}(n))$ operations in K assuming the space complexity of the original out-of-place algorithm in less than cn. For the truncated product the cost is $(2c+5)\mathsf{M}(n)+o(\mathsf{M}(n))$ operations in K using the same assumption. For the middle product our in-place variant incurs a logarithmic increase on the time complexity when quasi-linear algorithms are used, still using the same assumption on their space complexity. Finally, we further exhibit that the latter operation is more complex by investigating some kind of time/space efficiency equivalence between the main univariate polynomial product operations. In section 2.3, we first begin with a study of the classical algorithms for polynomial multiplication to exhibit their time and space complexity, our aim being to give explicitly all the constants. We think this presentation is highly relevant as it is not usually presented clearly in the literature. The end of this section is then devoted to providing a quick overview of our space-reducing and space-preserving algorithms for the short, middle and full product of univariate polynomials.

From a more practical point of view, we also investigated the possibility to reach a lower space storage for multi-precision integers that are usually based on the GMP library [Gt16]. The latter library is nowadays the standard for general purpose multi-precision integer computations. However, its versatility makes it not appropriate to represent integers with only few words, e.g. less than 512 bits. During the ANR HPAC, we have strengthen our LinBox block Wiedemann implementation in order to develop some cryptography attack for breaking ECDLP over a 116-bit binary field. Even if this computation was a toy challenge, it exhibited some bottleneck of our implementation: memory. In particular, relying on GMP for dealing with 232-bit integers was not a good idea: at least 128-bits of overhead are needed for the multi-precision structure `_mpz_struct`. We then propose in [Cpub-Bre+16] a new fixed precision arithmetic package whose aim is to minimize memory while offering good performance for classical operations. This package has been integrated in the LinBox project through the Givaro library (`https://github.com/linbox-team/givaro`). Our approach here is orthogonal to GMP as it targets fixed precision and it can avoid any structural information, providing an optimal memory usage. Our main idea is to use a recursive structure allocated on the stack and let the compiler do the rest. For this purpose, we rely on template meta programming [Ale01] to express the data and the algorithms. In particular, a 2^k-bit integer is represented recursively with two 2^{k-1}-bit integers stopping the recursion on native 64-bit integers. Figure 1.2 below emphasizes this design. One asset of this structure is to be really designed for recursive algorithms and especially for Karatsuba multiplication: i.e. the splitting of the operands being already given. Efficiency of the code is ensured by the compiler that enables to flatten the recursivity at compile time and then apply some classical optimizations: i.e. vectorization. In [Cpub-Bre+16], the experiments showed that our approach reveals more efficient than GMP below 1024 bits for integer addition and below 256 bits for integer multiplication. We also showed that for a 256-bits integer p, our package allows to speed-up the dense matrix-vector product modulo p by a factor of 4 compared to a GMP-based code (using `mpz_t`). Using this

Fig. 1.2: Template recursive structure of fixed precision integers in Givaro library.



```
template <size_t k>
struct ruint {
  ruint<k-1> High, Low;
};

template <> struct ruint<6>
{uint64_t Value;};
```

new package, we were able to reduce the memory footprint of block Wiedemann within LinBox and then we managed to compute the discrete logarithm on a machine with 1TB of memory, while our previous code did not succeed by lack of memory resources. An alternative of our approach would be to use automatically generated code for a given precision as done within the MPFQ library [GT07]. We think that our method here is rather simple and easy to maintain in a mainstream library as Givaro. However, the performances reached with MPFQ's approach might be clearly superior but the time efficiency was not our main purpose.

1.3.4 Using parallelism in practice

In the last past decade parallel computing regained attention due to the emergence of multi-core processors and programmable graphic processor units (GPGPU). Those cheap and easy access devices allow to use parallelism without requiring the access to large infrastructure and then avoiding complicated communication management. As our main motivation remains practical performance, we have investigated some work on this axis. In particular, we co-supervised two PhD students working mainly in this context: Thomas Izard (2008–2011) and Bastien Vialla (2012–2015). We also participated from 2012 to 2017 in the ANR project "HPAC: High Performance Algebraic Computing" that aims to design a DSL (Domain Specific Language) for parallel exact computing and use it to speeds up the LinBox library.

During Izard's PhD, we studied the possibility to use GPGPU to improve the performance of modular integer arithmetic that arises in elliptic curve cryptography, meaning integers of only few words. At that time, not much work had been done in this direction. Our work in [Cpub-GIT09] provided one of the first tweaks to optimize modular arithmetic on GPU using Montgomery reduction [Mon85]. In particular, one major difficulty was to map multi-precision integers into GPU's memory, and to minimize load and store between the different memory regions of the GPU (registers, share and global memory). Indeed, these memory movements are quite penalizing in practice, and their optimization is mandatory to exploit all the computing power of GPUs. In fact, we used a similar structure as for our previous recursive integer given in Figure 1.2. However, this prior version was designed with an unbalanced splitting that isolates each word: meaning a k-word integer is a leading word plus a $(k-1)$-word integer. Thanks again to template meta programming, we demonstrated that we could design templatized loop to express classic quadratic integer arithmetic on GPU with such a structure. The only difficulty with that approach was the ability of the GPU compiler, here the Nvidia one (nvcc), to do a good job on optimizing the mapping of variables into the registers. Nonetheless, our early benchmarks in [Cpub-GIT09] demonstrated that our GPU implementation was already competitive with the MPFQ library [GT07] that is the best library on CPU for that kind of computation. The conducted experiments and their specifications can be found in [Cpub-GIT09]. It makes no sense to provide more details here as the performance and the features of both CPU and GPU evolved a lot since then. Nevertheless, this preliminary work opened a breach for using such devices in exact computing. Many other persons have continued to explore this direction but embracing more complex operations over finite field to fully exploit the power of GPU. In particular, GPU has been used for block Wiedemann implementation in the context of FFS algorithm for breaking DLP over $\mathbb{F}_{2^{809}}$ [Bar+14]. The GPU have been essentially used to optimize the sparse matrix vector product.

Our experience with GPU being mitigated, the ratio development/performances being too high, we decided to study integer modular multiplication in the framework of multi-core processors. There, one obstacle was that modular multiplication is intrinsically sequential, indeed $ab \bmod p$ for n-words integers a, b and p is computed in at least two rounds: compute the $2n$-words integers ab then compute its residue modulo p. As proposed by Montgomery [Mon85] and Barrett [Bar86] the latter reduction can be done without any division and can be reduced roughly to two other sequential integer multiplications of the same wordsize, here n-words. The complexity of modular multiplication is then $3\mathsf{M}(n)$. The main difference between these two methods is that Montgomery treats integer from the trailing word to the leading one while Barrett is doing the opposite way, see [BZ10, Section 2] for a discussion on this similarity. [KT08] proved that both reductions can be used at the same time to exhibit a two fold parallelism for modular multiplication, yielding a parallel complexity of $O(\mathsf{M}(n/2) + 2\mathsf{M}(n, n/2))$ word operations. We provide in [Cpub-GII13] a generalization of this approach yielding more parallelism. While our method does not improve the parallel complexity of [KT08], we demonstrate that it reduces at least by two the number of thread synchronizations. In practice, for integers of moderate bit length, e.g. less than 2^{14}, thread synchronization remains costly compared to operations on integer machine words. We have provided experiments in [Cpub-GII13] that emphasize clearly this phenomenon on modern multi-core processor which makes our method of interest. Thanks to the reduction of synchronizations our approach yields a faster method for parallel modular multiplication with moderate size integers. The main idea of this contribution is detailed in section 3.2 together with the results of our experiments.

Our first experience with parallel computation with block Wiedemann was conducted in [Cpub-Dum+07] for matrix rank but using only a very limited amount of parallelism. In [Cpub-BDG10], we further extended our experiments. In particular, we provide hybrid formats for storing sparse matrix: mixing idea from sparse linear algebra community and introducing some specificity of exact computing. In particular, many matrices from the applications, e.g. in cryptography, have a bunch of ± 1 as coefficients that indeed require neither multiplication nor a real storage. Splitting these matrices into three different storage location, with apart the ones, the minus ones and the other values allows to significantly improve the performances of sparse matrix-vector product (SpMV for short). Similarly to the dense case, delaying modular arithmetic allows to minimize the number of calls of modular reduction. Thanks to these optimizations and to a first naive parallelization of our `PM-Basis` algorithm on multi-core machine, we were able to improve by 50% the performance of our first implementation [Cpub-Dum+07]. Few years later, during Vialla's PhD and ANR project "CaTREL: Cribles : Améliorations Théoriques et Résolution Effective du Logarithme discret", we pursued the optimization of SpMV operation [Cpub-GV14]. In particular, we enforced hand-made SIMD vectorization together with a compile time generation tool for choosing the most suited format and optimization techniques to apply for a given matrix. This extends similar idea mentioned in [Cpub-BDG10] and already exploited in the numerical context but optimizations being calculated at run time [VDY05]. Here, our approach is to run a pre-processing phase to automatically generate a file that encodes best tuned parameters for SpMV with the given matrix. With this new tool, we were able to again reach another speed-up of 50% on average, on a set of matrices representing some of the applications using exact linear algebra, see the sparse matrix collection from Dumas at `https://ljk.imag.fr/membres/Jean-Guillaume.Dumas/simc.html`. From these experiments we've learned that SpMV optimization is a rather more complicated task than its dense counterpart. It is mainly affected by the specificity of the given sparse matrices and then generic optimization technique would fail to provide good performances on most cases. Since our main purpose remains to provide generic code in the LinBox library, we did not further pursue this activity. Separately, many progress have been already done on SpMV for cryptography applications within the CADO-NFS software (`http://cado-nfs.gforge.inria.fr`), especially within Jeljeli's PhD [Jel15].

1.3.5 On some particular arithmetic operations.

During the supervision of Izard's PhD, we were interested by optimizing arithmetic operations for elliptic curve. In particular, our goal was to provide better formulas to describe scalar multiplication for elliptic curves with specific multipliers, e.g. 3, 5. The major advantage of having such specific formula is to provide more efficient algorithms for scalar multiplication than the one relying on classical double-and-add algorithm: the latter algorithm is only a variant of fast exponentiation for additive groups [Knu97]. Indeed, using tripling or quintupling can help to reduce the overall arithmetic count [DIM05]. In order to find better formulas for small scalars we suggested to express this operation trough a "minimal weight" direct acyclic graph. By definition, such a structure would automatically provide some kind of minimal formula. Of course finding such a DAG seems to be a hard problem [BU11]. We then use a simple heuristic in order to minimize as much as possible its weight. In [Cpub-GII09] we develop this approach and we propose to use basic arithmetic transforms based on the formula $(a \pm b)^2$ that potentially allow to trade multiplications with cheaper squares. Using this simple heuristic we have been able to improve the best known formula for quintupling using Jacobian coordinate [MD07].

As a different topic, we have been interested in a nice problem that is useful for some numerical applications. This concerns polynomial multiplication but when the coefficients are not expressed in the classical monomial basis. Such a problem is motivated from some application in approximation theory. There, the objective is to approximate functions with series that converge rapidly in order to reduce numerical errors. A classical way of obtaining rapid convergence is then to calculate the truncated series as a polynomial in Chebyshev basis. We shall mention that Chebyshev polynomials belongs to the large family of orthogonal polynomials. A natural question that has received less attention than the monomial basis is whether using such bases could change the difficulty of polynomial multiplication. More precisely, would it be possible to have multiplication algorithm with complexity $O(\mathsf{M}(n))$ for size-n polynomials expressed in Chebyshev basis.

In fact, [BSS10; BSS08] provides fast algorithms for conversions between the monomial and the Chebyshev basis, and more generally between the monomial and any orthogonal polynomial basis. For Chebyshev the cost is $O(\mathsf{M}(n))$, while for other orthogonal polynomial bases the cost is either $O(\mathsf{M}(n))$ or $O(\mathsf{M}(n)\log(n))$, depending on the family of the polynomial. From this result it is immediate to have an algorithm of complexity $O(\mathsf{M}(n))$ for multiplication in Chebyshev basis: simply use fast conversions. In [Jpub-Gio12], we study the effectiveness of this reduction for the classical algorithms. In particular, we demonstrate that we can improve the reduction by a constant factor. Our new approach does not rely on any basis conversion, and it rather uses the structure of Chebyshev polynomials. Using reversal operator on polynomials we are able to directly use algorithms from the monomial basis without any conversion. In particular, we show that our reduction leads to a complexity of $2\mathsf{M}(n) + O(n)$ in general for any multiplication algorithm, improving the result using [BSS10] for that specific case. The latter complexity can even be further reduced to only $\mathsf{M}(n) + O(n)$ when FFT approach is used. Both reductions provide evidence that make them practicable. With several experiments, we further emphasize the benefit of our reduction compared to the state of the art (fast) algorithm for Chebyshev polynomial multiplication using DCT transforms [BT97]. Finally, we give some evidence on the numerical stability of the proposed reduction through some experimental verification. These results and our main idea are further developed in section 2.4.

Contributions to polynomial arithmetic 2

2.1 Introduction

Polynomial arithmetic has been intensively studied in the past decades, in particular following the work in 1962 of Karatsuba and Ofmann [KO63] who have shown that one can multiply integers, and then polynomials, in a subquadratic number of operations. Let two size-n univariate polynomials over a ring R be given in monomial basis, one can compute their product using Karatsuba's algorithm in $O(n^{\log_2 3})$ operations in R. Since this seminal work, many other algorithms have been invented in order to asymptotically reduce the cost of the multiplication. In particular, one can go down to $O(n^{\log_{r+1}(2r+1)})$ operations in R with the generalized Toom-Cook method [Too63; Coo66] for any integer $r > 0$. One can even achieve a quasi-linear time complexity of $O(n \log n)$ using techniques based on the so-called FFT [CT65] when the base ring R contains a $2n$-th primitive root of unity [GS66]. When no such properly exists, the methods of Cantor-Kalfoten [CK91] leads to a complexity of $O(n \log n \log\log n)$ operations in R for any ring R. Note that this method works for any algebra and can be seen as an algebraic counterpart of the well known fast integer multiplication of Schönage-Strassen [SS71]. One can read the book [BZ10, Section 8] for further details on these discoveries.

We shall mention that in the setting of bit complexity the situation of polynomial multiplication is somewhat different. Indeed, since operations in the base ring R come to the game the pure algebraic algorithms may not lead to the best complexity estimates. There, the use of Kronecker substitution together with fast integer multiplication turns out to be the best alternative [GG13, Section 8.4]. Following the improvement of integer multiplication by Fürer in [Für07] and its refinement [HHL16], it has been showed by Harvey, Hoeven and Lecerf [HHL17] that one can reach a bit complexity of $O(n \log p \log(n \log p) 8^{\log^*(n \log p)})$ for polynomial multiplication over $\mathbb{F}_p[X]$ for any prime field $\mathbb{F}_p$. In particular, this result reduces the gap between the bit complexity of integers and polynomials multiplications over finite fields. Very recently, multiplication problem on both integers [HH19a] and polynomials [HH19b] have been further improved with the complete removal of the $\log^*$ exponent in the complexity, yielding a bit complexity of $O(n \log p \log(n \log p))$ for polynomial multiplication over $\mathbb{F}_p[X]$. We shall remark that the superiority of all the later approaches are purely theoretical and it will not be visible for any practicable application in the foreseeable future.

Besides improving the time complexity of classical polynomial multiplication, several variants of this problem are of interest in computer algebra. For instance, when computing with series of $\mathsf{R}[[X]]$, it is classical to work at a given precision d and there it amounts to dealing with polynomials of $\mathsf{R}[X]_{<d}$. In this setting, the multiplication is exactly the short product operation [Mul00]. Recalling that $\mathsf{M}(n)$ is the complexity of the full product of polynomials, it is not yet known if the short product operations (high

and low) can be done in time strictly less $\mathsf{M}(n)$. Some results by Mulderd [Mul00] and Hanrot, Zimmerman [HZ04] illustrate that one can reach a complexity below of $\mathsf{M}(n)$ for some prescribed $\mathsf{M}(n)$, in particular in the Karatsuba regime, but this is not yet true in general.

The situation of the middle product operation is more favorable. Indeed, in the classical case where we compute the n-middle terms of the product of two polynomials of sizes respectively n and $2n-1$, one can reduce the complexity of the naive approach that costs $2\mathsf{M}(n)$. In fact [BLS03] pointed out that the middle product operation is the transposed operation of the full product operation. There, applying the transposition principle they show that the complexity of middle product is exactly $\mathsf{M}(n)+n-1$ operations in R. First results on this transposition appeared in [HQZ04] where the improvement by almost a factor of two is used to speed up power series inversion, square root and division.

Besides the complexity relations between the full product and these two variants (short and middle product), few other problems around multiplication do not exhibit similar relations and our work has been to improve this situation.

Useful definitions: 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 over $\mathsf{R}[X]$. Their product $H = FG$ is a polynomial of size $2n-1$, that 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. The *low short product* of F and G 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

$$\mathsf{SP}_{\mathsf{hi}}(F,G) = (F \cdot G) \text{ quo } X^n.$$

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 $FG = \mathsf{SP}_{\mathsf{lo}}(F,G) + X^n \mathsf{SP}_{\mathsf{hi}}(F,G)$. In the following, we will abusively use *full product* or *product* to refer to the balanced full product operation, and *short product* to refer to the *low short product* operation.

Another classical operation on two polynomials is the middle product operation introduced in [HQZ04; BLS03]. Let F and G be two polynomials of sizes $n+m-1$ and n, respectively, their *middle product* is the size-m polynomial made of the central coefficients of the product FG, that is

$$\mathsf{MP}(F,G) = \left((F \cdot G) \text{ quo } X^{n-1}\right) \bmod X^m.$$

One may remark that middle product can be also computed by mean of short products:

$$\mathsf{MP}(F,G) = \mathsf{SP}_{\mathsf{lo}}(F \text{ quo } X^{n-1}, G) + \mathsf{SP}_{\mathsf{hi}}((X(F \bmod X^{n-1}), G). \tag{2.1}$$

A useful operation on polynomials is to take its reversal, sometimes called mirror. The *size-n reversal* of a polynomial F is defined by $\mathrm{rev}_n(F) = X^{n-1}F(1/X)$. Thanks to this operation, it is immediate that we have the following relation on short products:

$$\mathsf{SP}_{\mathsf{hi}}(F,G) = \mathrm{rev}_{n-1}(\mathsf{SP}_{\mathsf{lo}}(\mathrm{rev}_{n-1}(F \text{ quo } X), \mathrm{rev}_{n-1}(G \text{ quo } X))) \tag{2.2}$$

In particular, this means that any algorithm for the low short product can be turned into an algorithm for the high short product, and conversely. Note this is also a consequence of the principle of transposition [BLS03].

2.2 Verifying truncated polynomial products

While it is easy to probabilistically check that a product of two polynomials is correct over $\mathsf{K}[X]$ [1], by just checking the equality at a random point, it seems more complicated to have a similar test for short and middle product. The difficult here comes from the reduction $\bmod X^d$ that makes random evaluation not applicable. Indeed, verifying that $H = FG \bmod X^d$ by a single evaluation at $\alpha \in \mathsf{K}$ implies to know in advance the evaluation of FG quo X^d at α: $H(\alpha) = F(\alpha)G(\alpha) - (FG \text{ quo } X^d)(\alpha)$. It seems unlikely to try to check this equality without knowing $(FG \text{ quo } X^d)(\alpha)$. We have recently shown that it is possible to perform such verification in linear time without having to know in advance the remaining part of the product. Similarly, we have also proved that our approach remains valid for the verification of the middle product operation and any operations that compute a consecutive chunk of a product. The results provided in this section are based on [Jpub-Gio18] and [Cpub-GN18].

2.2.1 Scalar coefficients

Let $F, G \in \mathsf{K}[X]$ where $F = f_0 + f_1X + \cdots + f_{m-1}X^{m-1}$ and $G = g_0 + g_1X + \cdots + g_{n-1}X^{n-1}$. The product of polynomials FG is a bilinear application using the mapping $\mathsf{K}^m \times \mathsf{K}^n \to \mathsf{K}^{m+n}$, It can be turned into a linear application by fixing one of the operands and using either the mapping $\mathsf{K}^m \to \mathsf{K}^{m+n}$ or $\mathsf{K}^n \to \mathsf{K}^{m+n}$. There, fixing F, the polynomial product FG corresponds to the following matrix-vector product :

$$\underbrace{\begin{pmatrix} f_0 & & \\ f_1 & \ddots & \\ \vdots & \ddots & f_0 \\ f_{m-1} & & f_1 \\ & \ddots & \vdots \\ & & f_{m-1} \end{pmatrix}}_{\mathfrak{M}_{\mathsf{FP}(f)}} \times \underbrace{\begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{n-1} \end{pmatrix}}_{v_G} = \underbrace{\begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{m+n-2} \end{pmatrix}}_{v_H} \tag{2.3}$$

where $\mathfrak{M}_{\mathsf{FP}(f)} \in \mathsf{K}^{(m+n-1)\times n}$ is a Toeplitz matrix, $v_G \in \mathsf{K}^n$ and $v_H \in \mathsf{K}^{m+n-1}$ are the vectors of coefficients of G and $H = FG$ given in the canonical basis $(X^i)_{i\geq 0}$.

[1]Here, we place ourselves in a field without any further assumptions on its structure that could ease the verification of the product.

Choosing a random α from a finite subset $S \subset \mathsf{K}$ and checking that $H(\alpha) = F(\alpha)G(\alpha)$ can be done in $O(m+n)$ operations in K and the probability that H is equal to FG is greater than $1 - \frac{\deg H}{\#S}$ [Zip79; Sch80; DL78]. Our main observation here is that approach is equivalent to multiplying both parts of the equation (2.3) on the left by the row vector $\vec{\alpha} = [1, \alpha, \alpha^2, \dots, \alpha^{m+n-2}]$. By definition of v_H, we clearly have $\vec{\alpha} \cdot v_H = H(\alpha)$. Using the Toeplitz structure of the matrix $\mathfrak{M}_{\mathsf{FP}(f)}$ we have $\vec{\alpha}\mathfrak{M}_{\mathsf{FP}(f)} = F(\alpha)[1, \alpha, \dots, \alpha^{n-1}]$, which gives $(\vec{\alpha}\mathfrak{M}_{\mathsf{FP}(f)}) \cdot v_G = F(\alpha)G(\alpha)$. The probability estimates can be retrieved with the specific Freivalds certificate for matrix multiplication given in [KS93].

From there, we build up a similar procedure for the short product operations $\mathsf{SP}_{\mathsf{lo}}(F, G) = FG \bmod X^n$ and $\mathsf{SP}_{\mathsf{hi}}(F, G) = FG \operatorname{quo} X^n$ by simply keeping the meaningful rows in equation (2.3).

For the sake of clarity, we assume than $n = m$. Let $\vec{\alpha}_k = (1, \alpha, \dots, \alpha^{k-1}) \in \mathsf{K}^{1\times k}$ for $\alpha \in \mathsf{K}$ and $k \in \mathbb{N}^*$. The verification of the low short product of two size-n polynomials, $\mathsf{SP}_{\mathsf{lo}}(F, G) = H$, amounts to checking the following identify

$$\vec{\alpha}_n \cdot \underbrace{\begin{pmatrix} f_0 & & & \\ f_1 & \ddots & & \\ \vdots & \ddots & \ddots & \\ f_{n-1} & & f_1 & f_0 \end{pmatrix}}_{\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}} \times \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{n-1} \end{pmatrix} = \vec{\alpha}_n \cdot \begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{n-1} \end{pmatrix} \qquad (2.4)$$

while for the high short product, $\mathsf{SP}_{\mathsf{hi}}(F, G) = H$, the identity becomes

$$\vec{\alpha}_{n-1} \cdot \underbrace{\begin{pmatrix} f_{n-1} & f_{n-2} & \cdots & f_1 \\ & \ddots & \ddots & \vdots \\ & & \ddots & f_{n-2} \\ & & & f_{n-1} \end{pmatrix}}_{\mathfrak{M}_{\mathsf{SP}_{\mathsf{hi}}(f)}} \times \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n-1} \end{pmatrix} = \vec{\alpha}_{n-1} \cdot \begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{n-2} \end{pmatrix}. \qquad (2.5)$$

Lemma 2.2.1. *Let T_U and T_L be two Toeplitz matrices from $\mathsf{K}^{n\times n}$ being respectively upper and lower triangular. The matrix-vector products $\vec{\alpha}_n T_U$ and $\vec{\alpha}_n T_L$ can be computed in $O(n)$ operations in K.*

The previous lemma relies on the fact that the coefficients of these matrix-vector products are given during the course of Horner's algorithm on the polynomial given by the coefficients of the Toeplitz matrix. Indeed, each entry $v[i]$ of the vector $v = \vec{\alpha}_{n-1} \cdot \mathfrak{M}_{\mathsf{SP}_{\mathsf{hi}}(f)}$ is such that $v[i] = \alpha \cdot v[i-1] + f_{n-i}$, for $2 \le i \le n-1$ where $v[1] = f_{n-1}$. By transposition one has a similar approach for $\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}$: let $w = \vec{\alpha}_n \cdot \mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}$, then each entry $w[i]$ is such that $w[n-i] = \alpha^{-1} \cdot w[n-i+1] + \alpha^{n-1} f_i$, for $1 \le i \le n-1$ where $w[n] = \alpha^{n-1} f_0$. Therefore, the complexity of these operations is $O(n)$ and so for the whole verification procedure given by Equations (2.4) and (2.5). In fact the remaining operations are only dot products and polynomial evaluations of sizes $O(n)$ and one comparison of two elements of K.

The following corollary establishes a similar complexity for a full Toeplitz matrix as it is a sum of a lower and an upper triangular Toeplitz matrices.

Corollary 2.2.2. *Let $T \in \mathsf{K}^{n\times n}$ being a Toeplitz matrix and $\vec{\alpha}_n = (1, \alpha, \dots, \alpha^{n-1}) \in \mathsf{K}^n$. The matrix-vector product $\vec{\alpha}_n T$ and $T\vec{\alpha}_n$ can be computed in $O(n)$ operations in K.*

Using the Schwartz-Zippel lemma for univariate polynomials [Zip79; Sch80; DL78], the following lemma shows that our verification protocol has the same probability of success as the classical polynomial product verification.

Lemma 2.2.3. *Let $\alpha \in \mathsf{K}$ be chosen uniformly at random from $S \subset \mathsf{K}$, the probability that equation (2.4) is correct while $H \neq \mathsf{SP}_{\mathsf{lo}}(F,G)$ or that equation (2.5) is correct while $H \neq \mathsf{SP}_{\mathsf{hi}}(F,G)$ is less than $\frac{n}{\#S}$.*

2.2.2 Generalization to matrix coefficients

It turns out that all the previous results hold also if we consider polynomial matrices rather than polynomials with scalars. However, while for the scalar case our verification procedure is optimal, this is not true for polynomial matrices. Indeed, let $F \in \mathsf{K}[X]^{m\times k}_{<d}, G \in \mathsf{K}[X]^{k\times n}_{<d}$. Our verification protocol for either FG, $\mathsf{SP}_{\mathsf{lo}}(F,G)$ and $\mathsf{SP}_{\mathsf{hi}}(F,G)$ will cost $O(d\mathsf{MM}(m,k,n))$ while the amount of data involved in these computation is in $O(d(mk+kn+mn))$.

In a recent work with Neiger [Cpub-GN18], we have shown that we can make this protocol optimal even for polynomial matrices. Our observation here is classical and makes use of the Freivalds technique [Fre79], the idea being to verify a matrix equation by projection on a random vector: let $u \in \mathsf{K}^{1\times m}$ with entries chosen uniformly and independently from $S \subset \mathsf{K}$, the verification amounts to checking $uH = \mathsf{SP}_{\mathsf{lo}}(uF,G)$, or similarly for the other operations. Incorporating this approach within our truncated product leads to verify the following equation:

$$\vec{\alpha}_d \cdot \begin{pmatrix} u\times F_0 & & & \\ u\times F_1 & \ddots & & \\ \vdots & \ddots & \ddots & \\ u\times F_{d-1} & & u\times F_1 & u\times F_0 \end{pmatrix} \times \begin{pmatrix} G_0 \\ G_1 \\ \vdots \\ G_{d-1} \end{pmatrix} = \vec{\alpha}_d \cdot \begin{pmatrix} u\times H_0 \\ u\times H_1 \\ \vdots \\ u\times H_{d-1} \end{pmatrix} \tag{2.6}$$

The cost of evaluating equation (2.6) is $O(d(mk+kn+mn))$ operation in K by applying a Horner scheme that is similar to the one used in the scalar case. The difference here is that coefficients are just row vectors and the final comparison is made component-wise. The same approach remains valid for other products: full product and short products.

Note that the probability of error is given by the probability that either α is a root of at least one coefficient in the polynomial matrix $\Delta = (FG-H) \bmod X^d$, or it is not and u is in the kernel of $\Delta(\alpha)$. Hence, we obtain a probability of error that is less than $d/\#S$ with $S \subset \mathsf{K}$ from where α and the coefficients of u are chosen uniformly at random.

Unbalanced degree. In order to capture a more general setting that is useful in the framework of linear algebra on polynomial matrices [Zho12; Nei16a] , it is classical to adapt the low short product on polynomial matrices to the case where the truncation order is not identical on all columns. We denote the column degree of a polynomial matrix $P \in \mathsf{K}[X]^{m\times n}$ as $\text{cdeg}(P) = (c_1, \ldots, c_n)$ where $c_j = \deg(P_{*,j})$ is the degree of the column j of P for $1 \leq j \leq n$. The row degree $\text{rdeg}(P)$ is defined similarly for the rows of P.

Let us consider the tuple $\mathbf{t} = (t_1, \ldots, t_n) \in \mathbb{Z}^n_{\geq 0}$ called *truncation order*. In what follows, given a matrix $A \in \mathsf{K}[X]^{m\times n}$ and a truncation order $\mathbf{t} \in \mathbb{Z}^n_{\geq 0}$, we write $A \text{ rem } X^{\mathbf{t}}$ for the unique matrix $B \in \mathsf{K}[X]^{m\times n}$ such that $B = A \bmod X^{\mathbf{t}}$ and $\text{cdeg}(B) < \mathbf{t}$.

We can now define the following truncated polynomial matrix product problem. Given a truncation order $\mathbf{t} \in \mathbb{Z}^n_{\geq 0}$ and the polynomial matrices $F \in \mathsf{K}[X]^{m\times m}$ and $G \in \mathsf{K}[X]^{m\times n}$ with $\text{cdeg}(G) < \mathbf{t}$ compute the unique matrix $H \in \mathsf{K}[X]^{m\times n}$ such that $FG = H \text{ rem } X^{\mathbf{t}}$. We remark that taking $\mathbf{t} = (d, \ldots, d)$ allows us to describe the classical problem with balanced degrees. In that case, assuming $n \in O(m)$ it is clear that the complexity is $\tilde{O}(m^{\omega-1}nd)$.

Let us now denote $|\mathbf{t}| = \sum_{i=1}^n t_i$. Assuming $|\text{rdeg}(F)| \in O(|\mathbf{t}|)$, it has been shown in [Jea+17; Zho12; ZLS12] that the more general case of unbalanced degree polynomial matrix multiplication can be done in $\tilde{O}(m^{\omega-1}|\mathbf{t}|)$. Consequently, this complexity remains the best estimate we have for truncated polynomial matrix products.

Our problem now is to provide an optimal verification algorithm that has linear complexity in the size of F, G and H. More formally, by size of a matrix we mean the number of field elements used for its dense representation. Hence, we define the quantity

$$\text{Size}(P) = m^2 + \sum_{1\leq i,j\leq m} \max(0, \deg(p_{i,j}))$$

for a matrix $P = (p_{i,j}) \in \mathsf{K}[X]^{m\times m}$.

For the full product or the truncated product of polynomial matrices, it is hard in general to pin down in advance the size of the result and take this into account. The more we can do, as described above, is to use the column (or row) degree to estimate an upper bound on degrees of the result. When dealing with the verification problem, we do not have similar issue as we already know the output and its size. Therefore, using equation (2.6) we can tackle the verification problem with an optimal complexity that is in $O(\text{Size}(F) + \text{Size}(G) + \text{Size}(H))$. In particular, according to the previous setting, one can verify a truncated polynomial matrix product in $O(m|\mathbf{t}|)$ while the product itself costs $\tilde{O}(m^{\omega-1}|\mathbf{t}|)$.

The algorithm for verifying truncated polynomial matrix product is given in [Cpub-GN18, Section 3, Algorithm 2] and its complexity estimate has resulted in the following precise bound: $2\text{Size}(P) + (6m+1)|\mathbf{t}| + 2n\log(\delta) \in O(\text{Size}(P) + m|\mathbf{t}| + n\log(\delta))$ operations in K where $\delta = \max(\mathbf{t})$. In that specific case, the probability of error is less than $\delta/\#S$, with $S \subset \mathsf{K}$ is the set from which random element are chosen.

2.2.3 Generalization to the middle product

Going back to the problem of middle product, as shown in [Jpub-Gio18], it is straightforward from the result of the section 2.2.1 that one can reach a probabilistic linear time verification algorithm. Indeed, by definition the middle product operation $\mathsf{MP}(F, G)$ corresponds to the middle part of the full product FG. In the most common case, when F is of size $2n-1$ while G is of size n [HQZ04], we have that the middle product corresponds to two short products, see equation (2.1).

Therefore, by applying our previous result on low and high short products we can design a similar procedure to verify a middle product. Let $F = \sum_{i=0}^{2n-2} f_i X^i$ and $G = \sum_{i=0}^{n-1} g_i X^i$ and define the Toeplitz matrix $T \in \mathsf{K}^{n\times n}$ as follow:

$$T = \begin{pmatrix} f_{n-1} & f_{n-2} & \cdots & f_0 \\ f_n & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & f_{n-2} \\ f_{2n-2} & \cdots & f_n & f_{n-1} \end{pmatrix}.$$

The linear map corresponding to the middle product of F by any size-n polynomial G is given by the matrix T. Let $H \in \mathsf{K}[X]^n$, the following method allows to verify $H = \mathsf{MP}(F, G)$:

1. choose $\alpha \in \mathsf{K}$ uniformly at random from $S \subset \mathsf{K}$ and set $\vec{\alpha}_n = (1, \alpha, \ldots, \alpha^{n-1})$.
2. $y \leftarrow (\vec{\alpha}_n T) \cdot (g_0, \ldots, g_{n-1})^T$
3. return true if $H(\alpha) = y$, false otherwise.

Using Lemma 2.2.3 and Corollary 2.2.2 one can demonstrate that this procedure costs $O(n)$ and its probability of success is more than $1 - \frac{n}{\#S}$.

Extension to any partial product computation Without loss of generality, we assume that $\deg F = m \geq \deg G = n$ and $s \mid n$. We define a partial product operation on F and G as $\mathsf{PP}_s(F, G, i) = (FG \text{ quo } X^i) \bmod X^s$. This operation corresponds to extracting the s consecutive terms of the product FG starting from the monomial X^i. Assuming F is fixed, this operation is a linear application from $\mathsf{K}^n \to \mathsf{K}^s$ where its matrix has the form

$$\mathscr{C}_F = \begin{pmatrix} T_{\bar{F}_0} & T_{\bar{F}_1} & \cdots & T_{\bar{F}_{n/s-1}} \end{pmatrix} \in \mathsf{K}^{s\times n} \tag{2.7}$$

such that each $T_{\bar{F}_k} \in \mathsf{K}^{s\times s}$ for $k \in [0, \ldots, n/s-1]$ is a Toeplitz matrix formed from the coefficients of the polynomial $\sum_{i=0}^{m} f_i X^i$. More precisely, we have

$$T_{\bar{F}_k} = \begin{pmatrix} f_{i-ks} & f_{i-ks-1} & \cdots & f_{i-(k+1)s+1} \\ f_{i-ks+1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & f_{i-ks-1} \\ f_{i-(k-1)s-1} & \cdots & f_{i-ks+1} & f_{i-ks} \end{pmatrix}$$

with $f_j = 0$ when $j < 0$ or $j > m$. Let $H = \mathsf{PP}_s(F, G, i)$ and v_G, v_H be the vector of the coefficients of the polynomials G and H. By definition of $\mathscr{C}_F$ we have $v_H = \mathscr{C}_F \cdot v_G$. Here again, applying the vector $\vec{\alpha}_s$ to this equality provides us a way to certify the partial product operation. Let $H \in \mathsf{K}[X]$, the following method allows to verify $H = \mathsf{PP}_s(F, G, i)$:

```
1. choose $\alpha \in \mathsf{K}$ uniformly at random from $S \subset \mathsf{K}$ and set $\vec{\alpha}_s = (1, \alpha, \ldots, \alpha^{s-1})$.
2. for $k$ from 0 to $n/s$
       $y_k \leftarrow (\vec{\alpha}_s T_{\bar{F}_k}) \cdot [g_{ks}, \ldots, g_{(k+1)s-1}]^T$
3. return true if $H(\alpha) = \sum_{k=0}^{n/s-1} y_k$, false otherwise
```

Lemma 2.2.4. *The previous algorithm ensures that $H = \mathsf{PP}_s(F, G, i)$ with a probability greater or equal to $1 - s/\#S$ and it uses $O(n)$ operations in K.*

For some specific cases, one is able to reduce the complexity of $\mathsf{PP}_s(F, G, i)$. Indeed, depending on the value of i some Toeplitz matrices $T_{\bar{F}_k}$ will be zero. Using the structure of $\mathscr{C}_F$, one can prove that the number of non zero matrices is given by $\lceil \frac{i}{s} \rceil$ if $i < n$ and $\lceil \frac{m-i}{s} \rceil$ if $i > m$. For such cases, the complexity drops down to $O(s\lceil \frac{i}{s} \rceil)$ and $O(s\lceil \frac{m-i}{s} \rceil)$ which are below $O(n)$.

2.3 In-place polynomial multiplications

As for time, space complexity is another measure of efficiency of algorithms. In particular, optimizing the space complexity of an algorithm while keeping its original time complexity is relevant for challenging computations requiring huge resources: e.g. breaking DLP or RSA [Kle+12; Bar+14]. As mentioned in the book of Brent and Zimmermann [BZ10, Section 1.3.2], "The efficiency of an implementation of Karatsuba's algorithm depends heavily on memory usage." Of course, this assertion remains valid for any multiplication algorithm. Many attempts have been done to improve the space complexity of the main practicable algorithms for polynomial or integer multiplication: Karatsuba on polynomials [Tho02b; Roc09]; Karatsuba on integers [Che16]; FFT/TFT on polynomials [Roc09; HR10]. From these works, one now is able to perform almost[2] "in-place" multiplication, meaning that no more than $O(1)$ temporary space is needed apart from the input and output spaces. While these results are only valid for a particular set of algorithms, the large zoology of multiplication algorithms implies to design a more general result. In a recent work with Grenet and Roche we tackle this more general question for both the full product and its variants (short and middle product). In particular we provide generic algorithmic reductions that can turn any polynomial multiplication algorithm using $O(n)$ temporary space into an algorithm with only $O(1)$ temporary space while keeping its original asymptotic time complexity. Furthermore, we also extend this result to the cases of the short and the middle product which seems harder as the output space is more constrained. The results of this section are based on [Cpub-GGR19] and on-going works with Grenet and Roche.

2.3.1 Revisiting existing full product algorithms

Apart from the quadratic naive algorithm that is explicitly using only $O(1)$ temporary space, the other existing algorithms are basically using $O(n)$ temporary space for multiplying two size-n polynomials. In this section, we recall all the results that improved

[2]This is not yet true for Karatsuba since extra space for the call stack is necessary

this space complexity and we give their exact time complexity together with the exact number of temporary registers that they require. Note that these complexity bounds were never provided explicitly and we thought this manuscript is a good place to provide this overview. Furthermore, we provide a discussion on Toom-3 method where we give a new description that minimizes the number of additions and which allows to reach a better space complexity than the one used for integer multiplication in GMP library [Gt16].

Karatusba's method. This method invented by A. Karatsuba and Y. Ofman [KO63] uses the bilinearity of the multiplication to compute

$$h = f \cdot g = f_l \cdot g_l + ((f_l + f_h) \cdot (g_l + g_h) - f_l \cdot g_l - f_h \cdot g_h) X^{\frac{n}{2}} + f_h \cdot g_h X^n \tag{2.8}$$

where $f = f_l + x^{\frac{n}{2}} f_h \in \mathsf{K}[X]$ and $g = g_l + x^{\frac{n}{2}} g_h \mathsf{K}[X]$ are size-n polynomials. Hence, the complexity of this algorithm is given by $T(n) = 3T(\frac{n}{2}) + O(n) = O(n^{\log_2(3)})$. Deriving the constant of this complexity is given by the exact number of additions hidden behind the $O(n)$ and also the value of the last step of the recursion. The approach remains valid for odd n using unbalanced splitting of f and g. We refer to [BZ10, Section 1.3.2] for a more general description. Here, we mainly consider the case where n is a power of two.

Following the original description above, the number of additions in the recurrence is $4n-4$, yielding a complexity of $T(n) = 7n^{\log_2(3)} - 8n + 2$. As remarked by several authors [Roc09; GG13], this number can be reduced to $7n/2 - 3$ yielding a total complexity of exactly $6.5n^{\log_2(3)} - 7n + 1.5$. Indeed, let $k = \frac{n}{2}$ and define

$$\alpha = \alpha_0 + X^k \alpha_1 = f_l \cdot g_l,$$

$$\beta = \beta_0 + X^k \beta_1 = f_h \cdot g_h,$$

$$\gamma = \gamma_0 + X^k \gamma_1 = (f_0 + f_l) \cdot (g_0 + g_l).$$

We have from Equation (2.8) that

$$h = \alpha_0 + X^k(\gamma_0 - \alpha_0 + (\alpha_1 - \beta_0)) + X^{2k}(\gamma_1 - \beta_1 - (\alpha_1 - \beta_0)) + X^{3k}\beta_1 \tag{2.9}$$

and the term $(\alpha_1 - \beta_0)$ needs to be computed only once. The extra space needed in this formula corresponds to the storage of α_1 and β_0 since all other values can be stored in the output space before to be summed up properly. There, the space complexity is given by $S(n) = S(n/2) + n - 1 = 2n - 1 - \log(n)$. Thomé proves in [Tho02b] that Equation (2.8) can be evaluated with only $n - 1 + (n \bmod 2)$ extra registers. Since Thomé's approach uses more operations than Equation (2.9), one can use the latter to improve it further. Indeed, using Equation (2.9) and computing first α_1 then adding it to the output and do the same thing for β_0. This yields a space complexity of $S(n) = S(n/2) + n/2 - 1 = n - 1 - \log(n)$. When n is odd, following the analysis of Thomé [Tho02b] one can derive a space complexity of respectively $2n$ for Equation (2.9) and n for Equation (2.8). Remark all these space complexities do not take into account the management of the stack which incurs a penalty of at most $5\log(n)$: 4 for the inputs/output/temporary pointers and 1 for the degree.

Roche provides in [Roc09] a better space complexity by designing a version that allows to be half-additive, meaning that the main operation is $h = h_l + f \cdot g$ where h_l already occupied half of the output space. In particular, he obtains a space complexity that is no more than $5\log(n)$: this corresponds only to the management of the stack. The downside is that the time complexity is increased to $T(n) = 10n^{\log_2(3)} - 10n - \frac{n}{4}\log(n)(1+\log(n))$.

Toom-Cook variant Another well known method improving the complexity from Karatsuba's idea is due to Toom in [Too63] and is also discussed in Cook's PhD thesis [Coo66]. It is basically a generalization of Karatsuba's idea where each polynomial input is split in more than two pieces: $f = f_0 + f_1X^k + \cdots + f_{r-1}X^{(r-1)k} \in \mathsf{K}[X]$ and $g = g_0 + g_1X^k + \cdots + g_{r-1}X^{(r-1)k} \in \mathsf{K}[X]$ where $n = k \times r$ and $\deg f_i, \deg g_i < k$. Defining $F(X,Y) = \sum_{i=0}^{r-1} f_iY^i \in \mathsf{K}[X,Y]$ and $G(X,Y) = \sum_{i=0}^{r-1} g_iY^i \in \mathsf{K}[X,Y]$ such that $f = F(X,X^k)$ and $g = G(X,X^k)$, we have $f \cdot g = H(X,X^k)$ where $H(X,Y) = F(X,Y) \times G(X,Y)$. The latter polynomial multiplication can be done through evaluation/interpolation on Y at $2r-1$ points. Applying this method recursively provides an algorithm of time complexity $O(n^{\log_r(2r-1)})$. To have more details on this general result, one may look at [BZ10].

In the case of $r = 3$, the Toom-3 method can use the five points: $0, 1, -1, 2, \infty$. Hence, evaluation and interpolation are linear maps defined by Vandermonde matrices Δ and Γ. As described in [BZ10, Section 1.3.3], by using the factorization of these two matrices, given below, one can minimize the number of operations required for the interpolation and evaluation steps.

- evaluation matrix:

$$\Delta = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 1 & 2 & 4 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{2.10}$$

- Interpolation matrix:

$$\Gamma = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}}_{\Gamma_1} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \frac{3}{6} & 0 & \frac{2}{6} & \frac{1}{6} & -2 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{bmatrix}}_{\Gamma_2}. \tag{2.11}$$

Let us assume that $n = 3^k$ and that $\deg f, \deg g < n$. Using Equation (2.10) to evaluate $F(X,Y)$ and $G(X,Y)$ requires 10 additions and 4 multiplications by a constant with polynomials of sizes 3^{k-1}. Then the five products $F(X,\alpha) \cdot G(X,\alpha)$ with $\alpha \in \{0, 1, -1, 2, \infty\}$ are done through recursive calls. Finally, the coefficients of $f \cdot g$ are recovered by interpolation on $H(X,\alpha)$ using the matrix Γ and evaluating the result at $Y = X^{3^{k-1}}$. Using Equation (2.11) for interpolation leads to a cost of 8 additions, 3 multiplications by a constant and 2 divisions by a constant with polynomials of sizes $2 \times 3^{k-1} - 1$. For the evaluation of $H(X, X^{3^{k-1}})$, this amounts to adding 5 polynomials that overlap at most on $3^{k-1} - 1$ positions. This leads to a cost of $4 \times 3^{k-1} - 4$ addition in K.

The presentation given above is classical and one can achieve even better complexities. Bodrato and Zanoni provide in [BZ07] optimal factorization for the matrix Γ according to different cost models for the operations in $\mathbb{K}$, for instance when $\mathbb{K} = GF(2^k)$. Using one of these factorizations, given below, one can obtain a better complexity since the interpolation costs only 8 additions, 1 multiplication by a constant and 3 divisions by a constant with polynomials of sizes $2 \times 3^{k-1} - 1$.

$$\Gamma = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\Gamma'_1} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\Gamma'_2} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\Gamma'_3} \tag{2.12}$$

Using this description, we separate the interpolation and the last recombination phase which we can be merged together for saving redundant operations, as seen previously for Karatsuba. Surprisingly, we haven't found any references in the literature that provides similar improvement for Toom-3. Looking deeply in the code of GMP library [Gt16], we find out that the Toom-3 interpolation given by Equation (2.12) plus the recombination phase has been optimized in that way. Below, we recall briefly this optimization. Let us define

$$H' = \begin{bmatrix} \alpha_0 + X^{\frac{n}{3}}\alpha_1 \\ \beta_0 + X^{\frac{n}{3}}\beta_1 \\ \gamma_0 + X^{\frac{n}{3}}\gamma_1 \\ \delta_0 + X^{\frac{n}{3}}\delta_1 \\ \zeta_0 + X^{\frac{n}{3}}\zeta_1 \end{bmatrix} = \Gamma'_2 \times \Gamma'_3 \times \begin{bmatrix} F(X,0)\cdot G(X,0) \\ F(X,1)\cdot G(X,1) \\ F(X,-1)\cdot G(X,-1) \\ F(X,2)\cdot G(X,2) \\ F(X,\infty)\cdot G(X,\infty) \end{bmatrix}.$$

Then, we have that $f \cdot g = [1, x^{\frac{n}{3}}, X^{\frac{2n}{3}}, X^n, X^{\frac{4n}{3}}] \times \Gamma'_1 \times H'$ which can be rewritten as

$$\begin{aligned} f \cdot g = \alpha_0 + X^{\frac{n}{3}}(\alpha_1 + \gamma_0 - \delta_0) + X^{\frac{2n}{3}}(\beta_0 + \gamma_1 - (\zeta_0 + \delta_1)) \\ + X^n(\beta_1 + \delta_0 - \zeta_1) + X^{\frac{4n}{3}}(\zeta_0 + \delta_1) + X^{\frac{5n}{3}}\zeta_1. \end{aligned} \tag{2.13}$$

There, we can see that $\zeta_0 + \delta_1$ is calculated twice and thus $3^{k-1} - 1$ additions in $\mathbb{K}$ can be discarded. In particular, this gives a total number of 6 additions, 3 divisions by a constant and 1 multiplication by a constant with polynomials of sizes $2 \times 3^{k-1} - 1$ for computing H'; and $7 \times 3^{k-1} - 5$ additions in $\mathbb{K}$ for computing Equation (2.13).

One may remark that a similar approach can be done for the interpolation using the original Equation (2.11) from [BZ10]. Indeed, let H be such that

$$H = \begin{bmatrix} \alpha_0 + X^{\frac{n}{3}}\alpha_1 \\ \beta_0 + X^{\frac{n}{3}}\beta_1 \\ \gamma_0 + X^{\frac{n}{3}}\gamma_1 \\ \delta_0 + X^{\frac{n}{3}}\delta_1 \\ \zeta_0 + X^{\frac{n}{3}}\zeta_1 \end{bmatrix} = \Gamma_2 \times \begin{bmatrix} F(X,0)\cdot G(X,0) \\ F(X,1)\cdot G(X,1) \\ F(X,-1)\cdot G(X,-1) \\ F(X,2)\cdot G(X,2) \\ F(X,\infty)\cdot G(X,\infty) \end{bmatrix}.$$

Then, we have that $f \cdot g = [1, x^{\frac{n}{3}}, X^{\frac{2n}{3}}, X^n, X^{\frac{4n}{3}}] \times \Gamma_1 \times H$ which can be rewritten as

$$f \cdot g = \alpha_0 + X^{\frac{n}{3}}(\alpha_1 - \delta_0 + \beta_0) + X^{\frac{2n}{3}}(\zeta_0 + \beta_1 - \alpha_0 - (\gamma_0 + \delta_1))$$
$$+ X^n(\zeta_1 + \zeta_0 + \gamma_1 - (\alpha_1 - \delta_0)) + X^{\frac{4n}{3}}(\zeta_1 + \gamma_0 + \delta_1) + X^{\frac{5n}{3}}(\gamma_1). \qquad (2.14)$$

In that case, the gain is even better since both $\alpha_1 - \delta_0$ and $\gamma_0 + \delta_1$ are computed twice. Therefore, $2 \times 3^{k-1} - 2$ additions in K can be discarded. In particular, this gives a total number of 4 additions, 3 multiplication by a constant and 2 divisions by a constant with polynomials of sizes $2 \times 3^{k-1} - 1$ for computing H; and $10 \times 3^{k-1} - 6$ additions in K for computing Equation (2.14). Comparing with the optimized version using Equation (2.13), this does not improve the total number of operations. However, the number of additions has been reduced and this might effect the overall count when addition over K is twice more costly than the other ones. We summarize all the complexities we have seen for TOOM-3 method in the table below:

Tab. 2.1: Detailed time complexity of Toom-3 variants

Toom-3 version	# additions in K	# mul. by cst in K	# div. by cst in K
original [BZ10]	$30 \times 3^{k-1} - 12$	$10 \times 3^{k-1} - 3$	$4 \times 3^{k-1} - 2$
optimized eq. (2.14)	$28 \times 3^{k-1} - 10$	$10 \times 3^{k-1} - 3$	$4 \times 3^{k-1} - 2$
original [BZ07]	$30 \times 3^{k-1} - 12$	$6 \times 3^{k-1} - 1$	$6 \times 3^{k-1} - 3$
optimized eq. (2.13)	$29 \times 3^{k-1} - 11$	$6 \times 3^{k-1} - 1$	$6 \times 3^{k-1} - 3$

To the best of our knowledge, we haven't found any references in the literature that try to optimize the space complexity of Toom-3. The only relevant information has been extracted again from the GMP library, where the size of the temporary buffer for Toom-3 n-words integers multiplication is expected to be $2.5n + 10\log(n)$ words. As a matter of comparison, we provide below an estimate of the complexity of our method using (2.14) for the case of polynomials. Let us denote $S(n)$ this complexity.

First, it is easy to remark that the evaluations of the inputs using matrix-vector product with Δ (Equation (2.10)) does not require more space than the results from all the recursive calls. Thus, we need space for each result from recursive call: $5 \times (2 \times 3^{k-1} - 1)$. Using Equation (2.11), we have that the matrix/vector product with Γ_1 can be done without any extra space. Indeed, only the last two rows are computing new values while the other three ones are just identity. Finally, ordering the computation such that Equation (2.14) starts to compute the following values in the output space

$$\alpha_0 + X^{\frac{n}{3}}\alpha_1 + X^{\frac{2n}{3}}\zeta_0 + X^n\zeta_1 + X^{\frac{4n}{3}}\gamma_0 + X^{\frac{5n}{3}}\gamma_1$$

allows us to write the space complexity as $S(3^k) = S(3^k - 1) + 2 \times (2 \times 3^{k-1} - 1)$ which gives $S(n) = 2n - 2\log_3(n) - 1$. Here again, this complexity does not take into account the space for the management of the stack that should be $5\log_3(n)$ similarly to Karatsuba. It seems plausible following the idea of Thomé to show that even when n is not a power of three that the space should be no more than $2n + 5\log_3(n)$.

FFT based multiplication Since Cooley and Tukey result of FFT [CT65], the use of FFT algorithm within polynomial multiplication allows to reach a quasi-linear time complexity of $O(n\log(n))$. More precisely, assuming $n = 2^k$, $\deg f + \deg g < 2n$ and K embeds a $2n$-th primitive root of unity ω, the time complexity for computing the product

$f \cdot g$ is $9n\log(2n)+4n-2$ operations in K [GG13, Section 8.2]. In order to reach such a complexity, one needs to transform the two input polynomials into their evaluations at the first $2n$ powers of ω. This requires to have at least an extra space of $2n$ algebraic registers as one transform can be done into the output space. As all remaining steps can be done with $O(1)$ extra space, the space complexity for polynomial multiplication with FFT is exactly $2n$.

In [Roc09], an adaptation of this algorithm is proposed to decrease the space complexity to $O(1)$. The idea is not to compute the full product but only half of it using half size FFT, which can be done completely within the output space with only $O(1)$ temporary space. Then using iteratively the same process on smaller and smaller parts of the product makes it possible to perform the whole computation within the output space. While decreasing the space complexity to $O(1)$ this algorithm keeps the same asymptotic complexity of the classical FFT approach with an extra constant factor. In particular the time complexity becomes $11n\log(2n)-8n-\log(2n)+6$ operations in K. One may notice this approach works for any polynomial degrees but the time and space complexities are only valid for power of 2. Later, Harvey and Roche[HR10] designed an in-place version of van der Hoeven's truncated FFT method [Hoe04] to perform FFT-based multiplication in-place even when the size n is not a power of two.

Summary of complexities for polynomial multiplication. The table 2.2 below provides an overview of the space and time complexities for the most classical algorithms for computing polynomial products over $\mathsf{K}[X]$.

Tab. 2.2: Space and time complexity for full product algorithms with size-n polynomials inputs.

Algorithms	Time complexity	Space complexity	Reference
naive	$2n^2+2n-1$	$O(1)$	folklore
Karatsuba	$< 6.5n^{\log(3)}$	$\leq 2n+5\log(n)$	[KO63]
Karatsuba	$< 7n^{\log(3)}$	$\leq n+5\log(n)$	[Tho02b]
Karatsuba	$< 10n^{\log(3)}$	$\leq 5\log(n)$	[Roc09]
Toom-3	$< \frac{73}{4}n^{\log_3(5)}$	$\leq 2n+5\log_3(n)$	[Too63; Coo66]
FFT[1]	$9n\log(2n)+O(n)$	$2n$	[CT65]
FFT[1]	$11n\log(2n)+O(n)$	$O(1)$	[Roc09]
TFT[2]	$O(n\log(n))$	$O(1)$	[HR10]

One should notice that among these results, the one from [Roc09] achieving $O(\log(n))$ space provides a stronger operation than just a full product, that is called a half-additive full product. Since this stronger operation will play a major role in the next section, we provide a formal definition for it:

Definition 1 (Half-additive full product)**.** Let f and g be two size-n polynomials, and h be a polynomial of size $n-1$. 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 $\mathsf{FP}^{+}_{\mathsf{hi}}(f,g,h)=X^n h+fg$. 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. From the space complexity, the problem of in-place half-additive full product might be harder than full product since the available output space is further restricted: only n

[1] n must be a power of 2 and K must contain a $2n$-th primitive of unity.
[2] K must contain a $2n$-th primitive of unity.

free algebraic registers compared to $2n-1$. With no distinction we will refer to one of these two operations with FP^+.

2.3.2 In-place polynomial multiplication

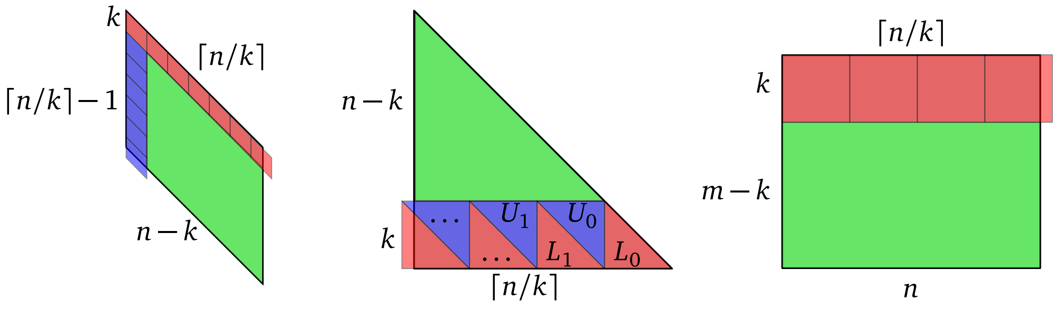
As seen in Table 2.2 many algorithms do not have yet an in-place variant but they need at most a linear amount of extra memory. Our main result in [Cpub-GGR19] is to provide generic reductions that allow to derive in-place polynomial multiplication algorithms from algorithms that use no more than a linear space complexity. The following theorem states formally this result, giving precisely the constants in the reductions:

Theorem 2.3.1.

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* $\leq \mathsf{M}(n)\log_{\frac{c+2}{c+1}}(n)+O(\mathsf{M}(n))$ *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 $s(n)$.

Fig. 2.1: 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).



Main idea. Our technique to reach in-place algorithms is to use an out-of-place algorithm for the problem as building block. There, we employ a strategy that divides the computation in such a way one can compute only a part of the result using the remaining part of the output as temporary. Doing this process in tail recursive manner yields our in-place reductions. We shall mention this technique has been already used in [Boy+09] to provide in-place variant of Strassen-Winograd fast matrix multiplication.

The (constant) amount of space needed in our in-place reduction corresponds to the space needed to process the base cases. Our technique expresses naturally when seeing polynomial product as a linear application. There, the idea is to split the matrix in a way that enable to both compute only part of the result, using other part as temporary, and to offer tail recursion. For full product, we will have to use an half-additive variant in order to be tail recursive[3]. Figure 2.1 exhibits the choice of our splitting and tiling for the three polynomial products (full product, (low) short product and middle product) that exhibit the expected

In-place full product. As precised above, our aim is to build an in-place (low-order) half-additive full product algorithm $\texttt{iFP}^{+}_{\texttt{hi}}$ based on an out-of-place full product algorithm $\texttt{oFP}$ that has space complexity cn. That is, we are given two n-size polynomials f and g in the input space and a $(n-1)$-size polynomial h 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 Figure 2.1 (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

$$h+fg=h+f_0g+\hat{f}g_0X^k+\hat{f}\hat{g}X^{2k}. \tag{2.15}$$

Recall that the output $\mathtt{R}$ has size $2n-1$ with its $n-1$ lowest registers containing h. Then Equation (2.15) can be evaluated with the following three steps:

```
1: $\mathtt{R}_{[0..n+k-1[} \leftarrow h+f_0g$                                   (red part in Figure 2.1)
2: $\mathtt{R}_{[k..n+k-1[} \leftarrow \mathtt{R}_{[k..n+k-1[}+\hat{f}g_0$    (blue part in Figure 2.1)
3: $\mathtt{R}_{[2k..2n[} \leftarrow \mathtt{R}_{[2k..2n[}+\hat{f}\hat{g}$    (green part in Figure 2.1)
```

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 $\texttt{oFP}^{+}_{\texttt{u}}$ for this task, based on a (standard) full product algorithm $\texttt{oFP}$: If f has degree $<k$ and g has degree $<n$, $n>k$, we write $g=\sum_{i=0}^{\lceil n/k\rceil-1} g_iX^{ki}$ 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 $\texttt{oFP}$ has time complexity $\mathsf{M}(n)$ and space complexity cn, the time complexity of $\texttt{oFP}^{+}_{\texttt{u}}$ 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, $\deg(h+f_0g+\hat{f}g_0X^k)<n+k-1$. This last step is thus a tail 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 $\texttt{oFP}^{+}_{\texttt{u}}$ fits exactly in the unused part of $\mathtt{R}$. This is the case when

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

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

The time complexity of Algorithm 1 is completely driven by the constant c and it satisfies the recurrence $T(n)=T(n-k)+(2\lceil n/k\rceil-1)(\mathsf{M}(k)+2k-1)$ where the out-of-place full product cost $\mathsf{M}(n)$ and has a space complexity $\le n$. We prove in [Cpub-GGR19,

[3]Tail recursiveness is mandatory in order to avoid the call stack which would incur a penalty of $O(\log(n))$ in the space complexity.

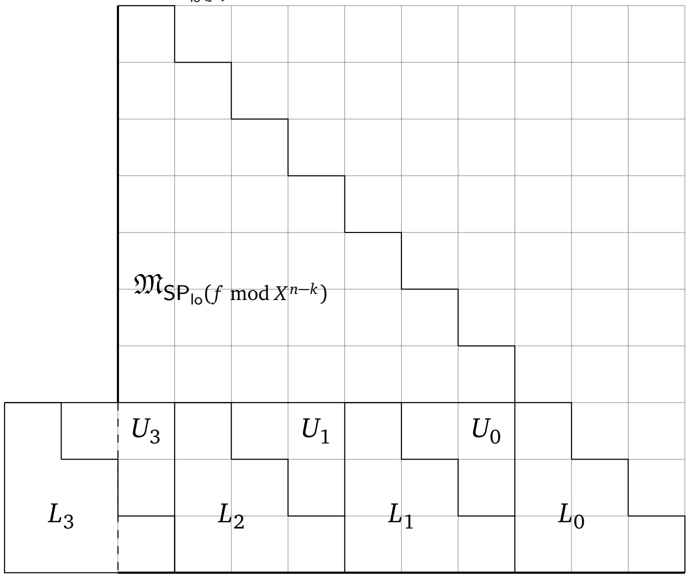
Algorithm 1 iFP$^+_{\mathtt{hi}}$_from_oFP

```
Input: $f$ and $g$ of size $n$ in the input space, $h$ of size $n-1$ in the output space R
Output: R contains $fg+h$
Required: Full product algorithm oFP of space complexity $\le cn$
1: if $n < c+2$ then
2:     R ← R + $fg$                                        ▷ using a naive algorithm
3: else
4:     $k \leftarrow \lfloor (n+1)/(c+3) \rfloor$
5:     $R_{[0..n+k-1[} \leftarrow$ oFP$^+_u(h, f_0, g)$           ▷ work space: $R_{[n+k-1..2n[}$
6:     $R_{[k..n+k-1[} \leftarrow$ oFP$^+_u(h + f_0 g, f, g_0)$         ▷ same work space
7:     $R_{[2k..2n[} \leftarrow$ iFP$^+_{hi}$_from_oFP($f$ quo $X^k$, $g$ quo $X^k$)
```

section 5.1] that the complexity $T(n)$ for in-place half additive full product is no more than $(2c+7)\mathsf{M}(n)+o(\mathsf{M}(n))$ operations.

In-place short product Similarly, we can design an in-place variant for the short product operation. For simplicity we only describe the case of the low short product since the high short product is equivalent by transposition, see Equation (2.2). Our approach is using the tiling of $\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}$ depicted on Figure 2.1 (center) and recalled below:

Fig. 2.2: Decomposition of $\mathfrak{M}_{\mathsf{SP}_{\mathsf{lo}}(f)}$.



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. In particular, 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, the coefficient of U_i are the coefficients of degree $ki+1$ to $k(i+1)-1$ of f which in fact corresponds to $U_i = \mathfrak{M}_{\mathsf{SP}_{\mathsf{hi}}(f_{ki,k(i+1)})}$. Remark that the matrices $L_{\lceil n/k \rceil -1}$ and $U_{\lceil n/k \rceil -2}$ must be padded with zero 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.

The second phase corresponds to the top $(n-k)$ rows and is a tail recursive call to compute h_0 to h_{n-k-1}: Indeed, $h \bmod X^{n-k} = \mathsf{SP}_{\mathsf{lo}}(f \bmod X^{n-k}, g \bmod X^{n-k})$.

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 $k \leq n/(c+2)$. The resulting algorithm is depicted in Algorithm 2 given below.

Algorithm 2 `iSP_lo_from_oSP`

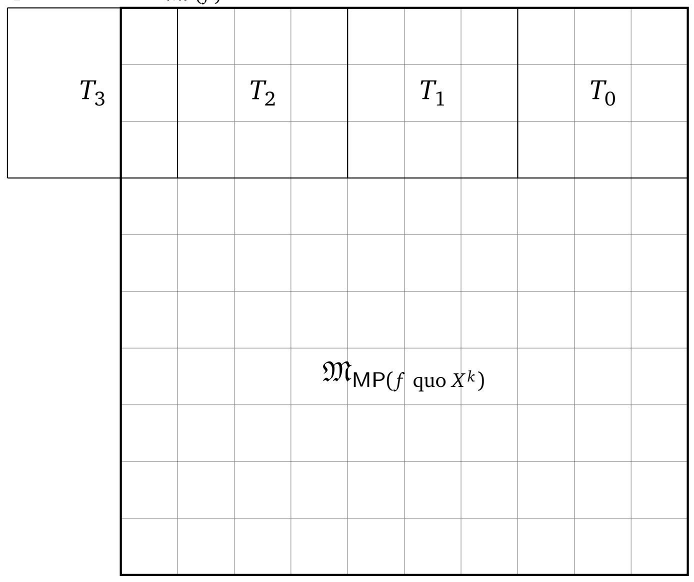
```
Input: $f$ and $g$ of size $n$
Output: R contains $\mathsf{SP}_{\mathsf{lo}}(f,g)$
Required: Two short product algorithms $\mathtt{oSP_{lo}}$ and $\mathtt{oSP_{hi}}$ of space complexity $\leq cn$
1: if $n < c+2$ then
2:     $\mathtt{R} \leftarrow \mathsf{SP}_{\mathsf{lo}}(f,g)$                ▷ using 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:     $\mathtt{R}_{[0..n-k[} \leftarrow \mathtt{iSP_{lo}\_from\_oSP}(f \bmod X^{n-k}, g \bmod X^{n-k})$
```

Here again, the time complexity of the algorithm is driven by the constant c and the cost of the in-place short product that we denote $\mathsf{M}(n)$. In particular, let $T(n)$ be the complexity of our in-place short product algorithm, we have that $T(n) = T(n-k) + \lceil n/k \rceil \mathsf{M}(k) + (\lceil n/k \rceil - 1)\mathsf{M}(k-1) + 2k(\lceil n/k \rceil - 1)$. We prove in [Cpub-GGR19, section 5.2] that the complexity for the in-place low (or high) short product is then no more than $(2c+7)\mathsf{M}(n) + o(\mathsf{M}(n))$ operations.

In-place middle product 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): polynomials inputs are of sizes $n+m-1$ and n, respectively and the polynomial output is of size m. The idea for the in-place algorithm is quite similar as for the short product: first compute k of the m coefficients of the result using a calls to the out-of-place algorithm in size (n,k) using $m-k$ free cells of the output space as work space and then use a recursive call. The in-place algorithm is again based on the tiling given in Figure 2.1 (right) that is recalled below in Figure 2.3.

The top k rows corresponds to the matrix $\mathfrak{M}_{\mathsf{MP}(f \text{ quo } X^{n+k})}$ and the bottom $n-k$ rows to the matrix $\mathfrak{M}_{\mathsf{MP}(f \text{ quo } X^{k})}$. The algorithm consists in computing $\mathfrak{M}_{\mathsf{MP}(f \text{ quo } X^{n+k})}[g_0,\dots,g_{n-1}]^T$ using the out-of-place algorithm and then compute the remaining $\mathfrak{M}_{\mathsf{MP}(f \text{ quo } X^{k})}$ as a tail recursive call. Since our assumption on out-of-place algorithm is that cn extra space is needed, there is no chance to have $cn < m-k$, especially when $m < n$. Therefore,

Fig. 2.3: Decomposition of $\mathfrak{M}_{\mathsf{MP}(f)}$.



we only consider to use a balanced out-of-place middle product (e.g. $m = n$). One can easily compute an unbalanced middle product by $\lceil n/m \rceil$ calls to the balanced case with the right coefficients, padding with zeroes when necessary and summing up the intermediate results. This situation is exactly depicted in Figure 2.3 where $\mathfrak{M}_{\mathsf{MP}(f \text{ quo } X^{n+k})}$ is decomposed with balanced middle products given by the Toeplitz matrices T_i defined with the coefficients of degree ki to $k(i+2)-1$ in f, i.e. $T_i = \mathfrak{M}_{\mathsf{MP}(f_{ki,k(i+2)})}$. We shall mention that $T_{\lceil n/k \rceil}$ might be padded with zeros when k does not divide n.

To make our algorithm to 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 size 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. The resulting algorithm depicted below in Algorithm 3 achieved the desired space complexity of $O(1)$.

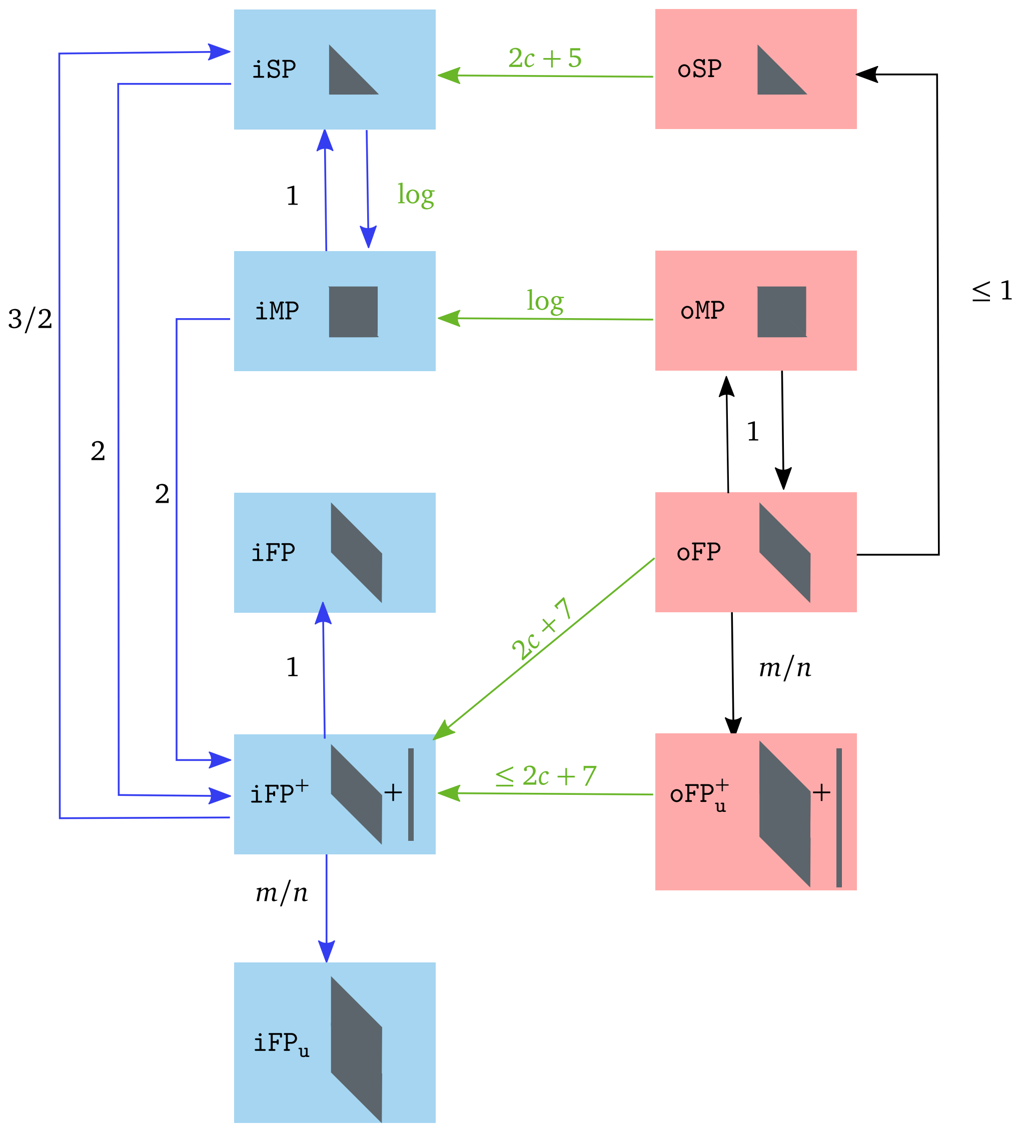
Algorithm 3 `iMP_from_oMP`

```
Input: $f$ and $g$ of size $n+m-1$ and $n$ respectively
Output: R contains $\mathsf{MP}(f,g)$
Required: Middle product oMP of space complexity $\leq cn$
1: if $m < c+2$ then
2:     R ← oMP($f, g$)                                   ▷ using a naive algorithm
3: else
4:     $k \leftarrow \lfloor m/(c+2) \rfloor$
5:     for $i = 0$ to $\lceil n/k \rceil - 1$ do                  ▷ work space: $R_{[k..m[}$
6:         $R_{[0..k[}$ += oMP($(f_{ki,k(i+2)}, g_{n-k(i+1),n-ki})$)
7:     $R_{[k..m[}$ ← iMP_from_oMP($f$ quo $X^k$, $g$)              ▷ recursive call
```

The time complexity of our in-place middle product is trickier than previous results on full and short product. Indeed, in this case the recursive call is no more balanced as the degree of one operand remains constant. In particular, let $T(m,n)$ be the time complexity of our in-place middle product for input (m,n) and let $\mathsf{M}(n)$ be the time complexity of the out-of-place balanced middle product. Then we have $T(m,n) = T(m-k,n)+\lceil n/k \rceil \mathsf{M}(k)$. One main difficulty of the analysis is that n remains constant and that k is varying at each level of the recurrence. i.e. it is getting smaller and smaller. We prove in [Cpub-GGR19, section 5.2] that the complexity for the in-place middle product is then no more than $T(n,n) \leq \mathsf{M}(n)\log_{\frac{c+2}{c+1}}(n) + O(\mathsf{M}(n))$ for $m = n$. We also show that the increase by $\log_{\frac{c+2}{c+1}}(n)$ in the time complexity is only inherent to quasi linear time algorithms and that in other cases the complexity remains $O(\mathsf{M}(n))$. We have not yet provided the constant in the latter case.

2.3.3 Space preserving reductions

Fig. 2.4: Overview of time and space relations between polynomial product operations. Arrows between algorithms indicate problem reductions that either preserve space (blue arrows), make it in-place (green arrows) or possibly increase space (black arrows); the label λ exhibits the increase in the corresponding time complexity.



As seen in the previous section, we obtain in-place algorithms for the full and short products that completely preserve the asymptotic time complexity while this is not completely true for the middle product. Therefore, we are interested to harness this difficulty. For this, 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* [Vas18, Definition 2.1] adapted to time-space fine-grained complexity classes [Lin+16].

The following theorem provided in [Cpub-GGR19] establishes the difficulty relation between problems.

Theorem 2.3.2. *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* $\mathsf{SP}_{\mathsf{hi}}$*,*

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

Note. As given in Equation (2.1), the middle product of f and g can be computed as the sum of the low short product of f quo X^{n-1} with g and the high short product of $X(f \bmod X^{n-1})$ 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 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 $\mathsf{MP} \leq_{\mathsf{TISP}} \mathsf{SP}$.

Figure 2.4 given above provides a general overview on what we know today about the space and time complexity relation between all variants of univariate polynomial product. The green values are obtained trough the results presented in this section, while the other ones are either classical or obtained by applying simple rewriting rule using a notion of fake padding. For instance, one of the easy reduction is obtained with the relation

$$\mathsf{SP}_{\mathsf{lo}}(f,g) = \mathsf{MP}(0 + X^{n-1} f, g).$$

Of course, we cannot store the zeros otherwise this would change the space complexity. Instead, our fake padding technique uses the fact that with only $O(1)$ non-algebraic registers one can embed those zeros into the structure of the polynomials: storing the indices range is sufficient. The structure is then in charge to return a zero whenever needed. Note, it is easy to show that structure does not increase the space and time complexity while it is compliant with recursivity.

2.4 Multiplication in Chebyshev basis

Polynomials are a fundamental tool in mathematics and especially in approximation theory where mathematical functions are approximated using truncated series. One can think of the truncated Taylor series to approximate a function as a polynomial expressed in the monomial basis. In general, many other series are preferred to the classical Taylor series in order to have better convergence properties. For instance, one would prefer to use the Chebyshev series in order to have a rapid decrease in the expansion coefficients which implies a better accuracy when using truncation [MH02; Boy01]. One can also use other series such as Legendre or Hermite to achieve similar properties. It is therefore important to have efficient algorithms to handle arithmetic on polynomials in such bases and especially for the multiplication problem [BT04; BJ10]. This is for instance of great interest to rigorous computing which aims to guarantee numerical approximation of functions through certified bounds on the results [BJ10].

Despite the zoology of algorithms for multiplication in monomial basis seen in Section 2.1, it is yet not clear how to directly adapt these algorithms to work with any other basis. Nevertheless, it has been shown in [BSS08; BSS10] and some earlier work [PST98] that change of basis between the monomial basis and any orthogonal polynomial bases can be done in time $O(\mathsf{M}(n)\log(n))$ for n-size polynomials. Converting the initial problem to the monomial basis and reverting back the result in the original basis yields immediately a cost of $\tilde{O}(\mathsf{M}(n))$ for the multiplication in all these bases. Even if this result is satisfactory from a theoretical point of view, the hidden constant and the logarithmic factor may incur a non-negligible gap in practical performance, especially due to the use of power series compositions [BSS08]. Reducing this gap is necessary to provide the most efficient implementations. In the case of Chebyshev basis, the complexity of the conversions is even lower down to $O(\mathsf{M}(n))$ [BSS08] but the constant remains non-negligible. For Chebyshev basis, some dedicated multiplication algorithms have been also proposed for $\mathbb{R}[X]$ or $\mathbb{C}[X]$. In particular, fast discrete cosine transform provides an algorithm with a complexity of $O(n\log(n))$ operations in $\mathbb{R}$ [BT97; CH90]. Note, this can be seen as the counterpart of the use of fast Fourier transform in the monomial bases case. This result is interesting since it provides a more practicable algorithm, avoiding conversions between bases. However, no other subquadratic approaches exist in the literature and this incurs a disparity with the monomial case. Since it is well know that quasi-linear time algorithms are not always preferred in practice, it is of interest to design other subquadratic variants.

Definition 2. Chebyshev polynomials are polynomials defined by the following recurrence relation:

$$\begin{cases} T_k(X) = 2xT_{k-1}(X) - T_{k-2}(X), \\ T_0(X) = 1, \\ T_1(X) = X. \end{cases}$$

Another possible definition over $\mathbb{R}$, is to define the Chebyshev polynomials of the first kind on the interval $[-1,1]$ by

$$T_k(X) = \cos(k \arccos(X)), \quad k \in \mathbb{Z}_{\geq 0} \text{ and } X \in [-1,1].$$

Problem 1. *Given two polynomials $a, b \in \mathbb{R}[X]$ such that $a(X) = \frac{a_0}{2} + \sum_{k=1}^{d} a_k T_k(X)$ and $b(X) = \frac{b_0}{2} + \sum_{k=1}^{d} b_k T_k(X)$ compute the polynomial $c(X) = a(X)b(X) \in \mathbb{R}[X]$ such that $c(X) = \frac{c_0}{2} + \sum_{k=1}^{2d} c_k T_k(X)$.*

The main difference between the Chebyshev basis and the monomial basis comes from the span of basis' elements product. Indeed, we have that

$$x^i \times x^j = x^{i+j} \quad \text{while} \quad T_i(X) \times T_j(X) = \frac{T_{i+j}(X) + T_{|i-j|}(X)}{2}.$$

It is obvious that classical algorithms discussed in Section 2.1 do not apply directly to solve Problem 1. In [Jpub-Gio12], we propose a novel approach for problem 1 using a reduction to the monomial case that improves the constant in the complexity of the generic approach using changes of bases [BSS08; BSS10]. In particular, our method avoids the use of change of bases. The following theorem states our main result.

Theorem 2.4.1. *The complexity of multiplication of polynomials in Chebyshev basis of degree less than n is no more than $2\mathsf{M}(n) + O(n)$. Furthermore, if we denote $\mathsf{M}_T(n)$ the complexity of such a multiplication, we have that the multiplication in monomial basis is no more than $2\mathsf{M}_T(n) + O(n)$.*

In the following sections we develop more on the main ingredient behind this result and we show the effectiveness of our approach for polynomial over $\mathbb{R}$ by comparing with the best existing method [BT97] which is based on the discrete cosine tranform (DCT-I). We shall mention that similar results on polynomials operations in Chebyshev basis have also been done in a similar way in [Tra15; Ben12], especially for the Euclidean division.

Note. Our method is presented in the context of polynomials in $\mathbb{R}[X]$ but it also works for any ring that would have basis' element following similar recurrence relation,, e.g. Dickson polynomial over a finite fields [LMT93].

2.4.1 Reduction to classic polynomial multiplication

Let two polynomials $a, b \in \mathbb{R}[X]$ of degree $d = n-1$ expressed in Chebyshev basis :

$$a(X) = \frac{a_0}{2} + \sum_{k=1}^{d} a_k T_k(X) \text{ and } b(X) = \frac{b_0}{2} + \sum_{k=1}^{d} b_k T_k(X).$$

The polynomial $c(X) = a(X)\, b(X) = \frac{c_0}{2} + \sum_{k=1}^{2d} c_k T_k(X) \in \mathbb{R}[X]$ is given by the following formula from [BT97]:

$$2c_k = \begin{cases} \displaystyle\sum_{l=0}^{k} a_{k-l} b_l + \sum_{l=1}^{d-k} (a_l b_{k+l} + a_{k+l} b_l), & \text{for } k = 0, \ldots, d-1, \\ \displaystyle\sum_{l=k-d}^{d} a_{k-l} b_l, & \text{for } k = d, \ldots, 2d. \end{cases} \tag{2.16}$$

Our main observation in [Jpub-Gio12] is to remark that all terms appearing in Equation (2.16) can be obtained from convolutions on coefficients of $a(X)$ and $b(X)$. Indeed, let

$$\begin{aligned}
\bar{a}(X) &= a_0 + a_1X + a_2X^2 + \ldots + a_dX^d, \\
\bar{b}(X) &= b_0 + b_1X + a_2X^2 + \ldots + b_dX^d, \\
\bar{r}(X) &= \text{rev}_{d+1}(a(X)) = a_d + a_{d-1}X + a_{d-2}X^2 + \ldots + a_0X^d. \qquad (2.17)
\end{aligned}$$

These polynomials are given in the monomial basis, and their products can computed in $\mathsf{M}(n)$. In particular, let

$$\begin{aligned}
\bar{f}(X) &= \bar{a}(X) \times \bar{b}(X) = f_0 + f_1X + f_2X^2 + \ldots + f_{2d}X^{2d}, \\
\bar{g}(X) &= \bar{r}(X) \times \bar{b}(X)) = g_0 + g_1X + g_2X^2 + \ldots + g_{2d}X^{2d}. \qquad (2.18)
\end{aligned}$$

We can rewrite Equation (2.16) as follow

$$2c_k = \begin{cases} f_k + g_{d-k} + g_{d+k} - a_0b_k - a_kb_0, & \text{for } k = 0, \ldots, d-1, \\ f_k, & \text{for } k = d, \ldots, 2d. \end{cases} \qquad (2.19)$$

From this equation it is straightforward to derive an algorithm of complexity $2\mathsf{M}(n)+O(n)$ for the multiplication of polynomials in Chebyshev basis, recalling that $n = d + 1$.

Improving the constant for DFT approach. We propose in [Jpub-Gio12] to improve this approach when the underlying multiplication algorithm is based on DFT approach. Indeed our approach needs two multiplications in monomial basis using operands $\bar{a}(X), \bar{b}(X)$ and $\bar{r}(X) = \bar{a}(X^{-1})X^d$. Somehow, we compute twice the DFT transforms of $\bar{b}(X)$ and $\bar{a}(X)$ on $2n$ points. Let ω be a $2n$-th primitive root of unity, we have

$$\begin{aligned}
\mathtt{DFT}_{2n}(\bar{a}) &= [\ \bar{a}(\omega^k)\ ]_{k=0\ldots2n-1}, \\
\mathtt{DFT}_{2n}(\bar{r}) &= [\ \bar{a}(\omega^{2n-k})\ \omega^{kd}\ ]_{k=0\ldots2n-1}.
\end{aligned}$$

This implies that the coefficients of $\mathtt{DFT}_{2n}(\bar{r})$ correspond to the ones of $\mathtt{DFT}_{2n}(\bar{a})$ in reversed order scaled out by the adequate powers of ω, and this can be done with only $O(n)$ operations. We can even go further and remove the multiplications by powers of ω. Indeed, one can calculate the product $X\bar{r}(X)\bar{b}(X)$ instead of the product $\bar{r}(X)\bar{b}(X)$. Note that the degree of the product remains less than $2n$ and can be done through $2n$ points DFT. There, recalling that $d = n - 1$ and ω is a $2n$-th primitive root of unity it is sufficient to remark that

$$\mathtt{DFT}_{2n}(X\bar{r}) = [\ \bar{a}(\omega^{2n-k})\ (-1)^k\ ]_{k=0\ldots2n-1}$$

which removes the multiplications by powers of ω. Using all these considerations, one can modify our approach to save exactly 2 DFTs. Saying it more generally, the complexity of our approach has exactly the same asymptotic complexity as the multiplication in monomial bases, for algorithms using a DFT approach.

Estimation of the arithmetic complexity. Since our reduction is generic, we will denote it PM-Chebyshev followed by the underlying multiplication method in monomial bases.

i.e. PM-Chebyshev (Karatusba) for using Karatsuba algorithm. We will also denote "DCT-based" the fast method of [BT97] and "Direct" the naive method given by Equation (2.16). Table 2.3 below summarizes the exact number of operations in $\mathbb{R}$ needed to multiply two n-size polynomials. Note we also provide in this table the estimate for the fast method of [BT97] when the DCT-I transform is achieved through DFT. Indeed, one can calculate up to some scaling a N-size DCT-I transform by a 2N-size DFT transform. This estimate makes sense for practical purpose since software might use DFT for computing DCT-I transforms [FJ05]. We remark from the values that the faster dedicated method of [BT97] provides the best possible complexity in theory. Our approach with DFT remains close to it and even outperforms it if DCT is achieved through DFT.

Tab. 2.3: Number of real operations for multiplication of size-n polynomial in Chebyshev basis over $\mathbb{R}[X]$.

Algorithm	**nb. of operations in $\mathbb{R}$**
Direct	$2.5n^2 - 0.5n$
DCT-based [BT97]	$(12n+3)\log 2n - 14n + 15$
DFT-based [BT97]	$24n \log 2n - 42n + 18$
PM-Chebyshev (Schoolbook)	$4n^2 - 1$
PM-Chebyshev (Karatsuba)	$13n^{\log 3} - 10n$
PM-Chebyshev (DFT-based)	$16n \log 2n - 12n + 21$

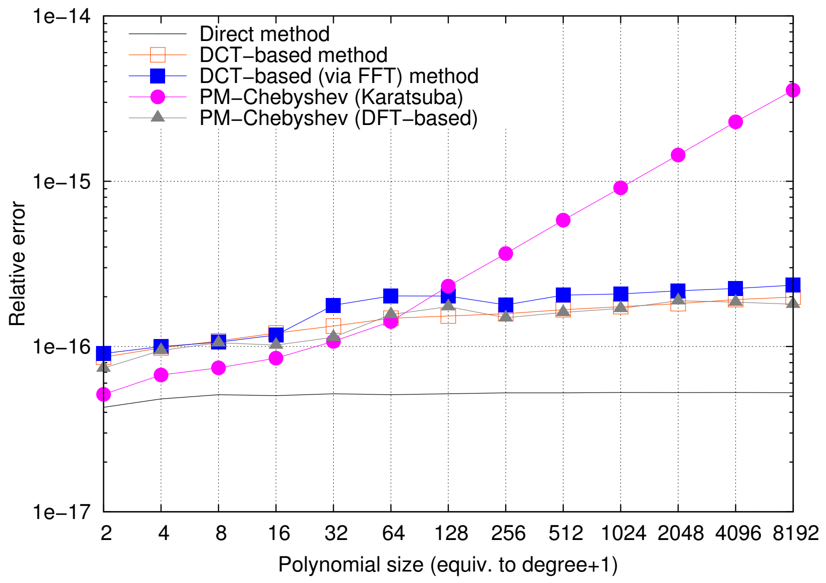
2.4.2 Practical experiments.

In order to further emphasize the benefit of our approach from the existing methods, we proposed in [Jpub-Gio12] to evaluate both its performance and its numerical accuracy. First, we remarked from the standard numerical library for fast discrete Fourier transform, namely FFTW [FJ05], that standard code for the DCT-I transform has serious numerical problems, in particular loosing accuracy in the computed result[4]. There, for improving the numerical accuracy it is preferred to rely on DFT code for computing DCT-I and then relax the performance by a factor of two. In this context, our approach might reveal competitive as it offers a lower count of operations, as depicted in Table 2.3.

Numerical accuracy. To emphasize the "good" numerical behavior of our method we provide in [Jpub-Gio12] practical experiments. More precisely, we have computed approximations of the relative error by computing the exact solution with multiprecision rational numbers from the GMP library [Gt16]. The coefficients of the polynomial are random floating point numbers where the exponents have a rather small norm, e.g. a fraction of two integers. The latter choice ensures that DFT transforms remains in a favorable stability range [Hoe08]. Note that all complex DFTs are computed with FFT through FFTW library. To confirm the implementation of the DCT-I code in FFTW, we also wrote a dedicated code that compute DCT-I with double length DFT. Figure 2.5 illustrates the stability results on average for 50 polynomial product for sizes ranging from 2 to 2^{13}.

[4]see http://www.fftw.org/fftw3_doc/Real-even_002fodd-DFTs-_0028cosine_002fsine-transforms_0029.html

Fig. 2.5: Experimental measure of the relative error in double precision. Polynomial coefficients lying in $[-50, 50]$.



From Figure 2.5 we can conclude few thoughts. Our method PM-Chebyshev (DFT-based) seems to offer the same numerical behavior as the DCT based one, and thus offer a concrete alternative in practice. As expected, the code using DCT-I is as stable as the one using a double length-DFT since they are indeed similar code. Finally, the use of our approach together with Karatsuba technique is not worth it as it introduces too many numerical errors. This is certainly due to the high number of additions which often cause numerical errors. From a theoretical point of view, our approach with FFT-based polynomial multiplication should be similar to the one using DCT (with double length FFT) as they both use FFT and perform some extra additions/scaling that are similar. Of course, this rough analysis need to be confirmed by a dedicated work that would study the backward stability of the multiplication problem in Chebyshev basis, as done for the monomial case in [Hoe08].

Performances The table 2.4 reports the computation time for the same set of parameters as for the stability evaluation. As expected, our approach using reduction to the monomial case with FFT is competitive and even outperforms the DCT based approach of [BT97]. In particular, the practical performance confirms the estimation based on the number of operations given in Table 2.3. These timings also confirm that the DCT-I code in FFTW is using a double length DFT. Finally, Karatsuba approach does not bring any benefit in practice neither for its performance nor for its numerical stability, and it must be avoided. As a general conclusion, our approach might be worthwhile for any efficient polynomial product algorithm that has good stability behavior.

Tab. 2.4: Times of polynomial multiplication in Chebyshev basis (given in μs) on Intel Xeon 2GHz platform.

n	Direct	DCT-based	DCT-based with DFT	PM-Chebyshev Karatsuba	PM-Chebyshev with DFT
2	0.23	1.25	0.43	0.41	0.49
4	0.43	1.32	0.57	0.62	0.58
8	0.52	1.80	0.81	1.04	0.86
16	1.28	3.11	1.56	1.91	1.28
32	4.33	4.88	2.84	5.11	2.58
64	15.73	8.82	9.55	16.83	5.41
128	56.83	18.08	17.07	70.54	13.37
256	211.34	35.97	31.42	195.85	24.41
512	814.71	82.47	68.67	554.36	49.53
1024	3219.13	160.81	159.21	1618.37	109.74
2048	12800.30	346.95	364.82	4749.63	236.01
4096	54599.10	741.38	740.31	14268.10	556.83
8192	220627.00	1613.43	1625.43	40337.20	1176.00

3 Contributions to integer arithmetic

3.1 Introduction

Historically, integer arithmetic received more attention than its polynomial analogue. Indeed, the major results for the multiplication problem [KO63; Coo66; Too63; SS71] have been primarily designed for the integer case, and then were further adapted to the much simpler case of polynomials. Indeed, integer carries make things more complicated. While being quite similar, new breakthrough ideas are usually done for integers. This has been again verified with the work of Fürer in 2007 [Für07] which improves the upper bound $O(n\log(n)\log(\log(n)))$ of [SS71] to $\mathsf{I}(n) = O(n\log(n)2^{O(\log^*(n))})$, where $\log^*$ is the iterated logarithm function. It is only ten years later that the polynomial case followed [HHL17]. Since [Für07] many other improvements have been done to reduce the constant value in the exponent of $2^{O(\log^*(n))}$. In a recent report [HH19a] van der Hoeven and Harvey provide a stronger result where they demonstrate an unconditional upper bound of $O(n\log n)$ for the problem. Note that this time, they also provide a similar result for the polynomial case [HH19b] but under some mild hypothesis on the coefficient ring. Interested readers will find in the two latter references a really nice exposition of historical advances on the multiplication problem. To a major extent, this new upper bound is rather theoretical and it should not yet have a large impact on the practical performance of multiplication of large integers. Schönhage-Strassen algorithm's implementation is still as of today the most efficient in practice for really large integers [GKZ07].

Let A and P be respectively m-word and a n-word integers with $n \le m$. The quotient $(A \operatorname{quo} P)$ and the remainder $(A \bmod P)$ from the Euclidean division of A by P can be computed in $O(\mathsf{I}(m))$ word operations; the algorithm goes back to [Coo66], see also [AHU74, Section 8.2] and [BZ10, Section 2.4.1]. As a consequence, modular multiplication modulo P with n-words integers can also be done using $O(\mathsf{I}(n))$ word operations: first do the product over the integers and then reduce the result using a division by P. The rather more complicated operation of inversion modulo P can be done using $O(\mathsf{I}(n)\log(n))$ word operations using fast gcd algorithms [Knu70; Sch71]. All these results provide the foundation for the most classical method dealing with arithmetic over prime finite fields $\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z}$. When several modular multiplications are needed, as for instance in the exponentiation used by RSA algorithm [RSA78] or simply in matrix multiplication in $\mathbb{F}_p$, two major techniques based on precomputation improve the latter result. In [Bar86; Mon85] it is shown that modular multiplication can avoid division when either β^{2n}/P or $1/P \bmod \beta^n$ are known in advance. There the modular reduction only requires two short products and costs $2\mathsf{I}(n)$. Of course, the precomputations cost $O(\mathsf{I}(n))$ using fast Newton iteration [Coo66] and they are easily amortized when several multiplications are needed.

In the following sections, we present two of our results that aim to perform modular integer computations [Cpub-GII13; Jpub-Dol+18]. The first section gives a new algorithm for modular multiplications based on [Bar86; Mon85] that enables better practical performance on multi-core processors by exhibiting lower synchronization barriers than any previous algorithm. The second section focuses on basis conversions arising within multi-modular technique. Similarly to Barret and Montgomery algorithms [Bar86; Mon85] that use precomputation to improve efficiency of modular multiplication, we provide a precomputation technique that enables the use of matrix multiplication for integer conversions with the Residue Number System (RNS) [BZ10, Section 2.1.3]. Exploiting this technique leads us to practical improvements over any existing methods, even with the asymptotically faster ones.

3.2 Modular multiplication with better parallelism

Barrett's method [Bar86] precomputes $\nu = \lfloor \beta^{2n}/P \rfloor$ to easily compute an approximated quotient $Q = \lfloor \lfloor AB/\beta^n \rfloor \nu/\beta^n \rfloor$ such that $C = AB - QP$ with $C < 3P$. Thanks to this precomputation, the cost of the modular multiplication $AB \bmod P$ reduces to three n-word integer products: $T_0 + \beta^n T_1 = AB = \mathsf{FP}(A,B)$; $Q = \lfloor T_1 \nu/\beta^n \rfloor = \mathsf{SP}_{\mathsf{hi}}(T_1, \nu)$ and $C = T_0 + \mathsf{SP}_{\mathsf{lo}}(Q,P)$ plus at most two subtractions by P since $0 \leq C < 3P$. Similarly, Montgomery's method [Mon85] precomputes $\mu = -1/P \bmod \beta^n$ so that we can compute $C = AB/\beta^n \bmod P$ with almost the same operations: $T_0 + \beta^n T_1 = AB = \mathsf{FP}(A,B)$; $Q = T_0\mu \bmod \beta^n = \mathsf{SP}_{\mathsf{lo}}(T_0,\mu)$ and $C = T_1 + \mathsf{SP}_{\mathsf{hi}}(Q,P)$[1]. Here again, subtracting P might be necessary since $0 \leq C < 2P$. This method can be turned into a real modular multiplication if all operands are assumed to be already in the Montgomery representation, that is $A = A'\beta^n \bmod P$ and $B = B'\beta^n \bmod P$. There, the multiplication $AB \bmod P$ with Montgomery's method gives $A'B'\beta^n \bmod P$. The complexity of these two methods are similar and it is no more than $3\mathsf{I}(n)$ word operations (assuming the precomputation is already done). The interested reader can find an interesting discussion on the similitude of these two methods in [BZ10, Section 2.4] together with improved variants for some specific integer multiplication algorithms. Note that when subquadratic multiplication algorithms are not needed, e.g. when the number of words remains small (less than 6 in GMP library [Gt16]), one would prefer to use the Finely Integrated Operand Scanning (FIOS) of [KAK96]. This method processes sequentially each word of one operand and performs unbalanced multiplications (one word by n-word) followed by reduction implying a single word quotient. This method is of interest if the output space is restricted, meaning for instance that the full product cannot be stored. This is typically the case of interest for hardware implementations. Further insights on hardware implementation of modular multiplication can be found in [Neg16].

During the supervision of Thomas Izard's PhD thesis, we have worked on these methods to incorporate more parallelism. In particular, we achieve in [Cpub-GII13] a modular multiplication algorithm that exhibits better independent computations while minimizing the number of thread synchronization. We also provide in [Cpub-GIT09] a first insight on using GPU for performing basic operations over prime fields. More precisely, we managed to show that clever implementation of FIOS method can compete with the best available software that is the MpFQ library [GT07]. The following sections discusses only about our first contribution.

[1]Here, the short product must compute the high n-word of the product, not the $n-1$ as defined in $\mathsf{SP}_{\mathsf{hi}}$

3.2.1 Partial reductions

We introduce a partial Montgomery reduction and a partial Barrett reduction that only reduce the operand by t words for a given integer $t \geq 0$. In particular, let P be an n-word integer and C be an m-word integer such that $C < P^2$, Algorithm 4 below computes $R \equiv C\beta^{-t} \pmod{P}$ such that $0 \leq R < \beta^{m-t}$. Similarly, Algorithm 5 computes $R \equiv C \pmod{P}$. It is assumed that $t \leq m - n$ to guarantee the algorithms to be correct.

Algorithm 4 PMR (Partial Montgomery Reduction)

```
Input: integers $P, C, t, \mu$ such that $\beta^{n-1} < P < \beta^n$, $\gcd(P, \beta) = 1$, $0 \leq C < P^2$, $C < \beta^m$,
  $0 \leq t \leq m - n$ and $\mu = -P^{-1} \bmod \beta^t$ (precomputed)
Output: $R \equiv C\beta^{-t} \pmod{P}$ with $0 \leq R < \beta^{m-t}$
  $Q \leftarrow \mu C \bmod \beta^t$
  $R \leftarrow (C + QP)/\beta^t$
  if $R \geq \beta^{m-t}$ then
      $R \leftarrow R - P$
  return $R$
```

Algorithm 5 PBR (Partial Barrett Reduction)

```
Input: integers $P, C, t, \nu$ such that $\beta^{n-1} < P < \beta^n$, $0 \leq C < P^2$, $C < \beta^m$, $0 \leq t \leq m - n$
  and $\nu = \lfloor \beta^{n+t}/P \rfloor$ (precomputed)
Output: $R \equiv C \pmod{P}$ with $0 \leq R < \beta^{m-t}$
  $Q \leftarrow \lfloor C_1 \nu / \beta^t \rfloor \beta^{m-n-t}$ where $C = C_1\beta^{m-t} + C_0$ with $0 \leq C_0 < \beta^{m-t}$
  $R \leftarrow C - QP$
  while $R \geq \beta^{m-t}$ do $R \leftarrow R - \beta^{m-n-t}P$
  return $R$
```

The complexities of these partial reductions are identical. Let us denote their cost $T(t, n, s)$ where t is the number of words to reduce, n is the number of words of P. Here, s is used to express that less than t non-zero words appear in the left-most digits of C for **PBR**; and the right-most digits of C for **PMR**. Thus, we have

$$T(t, n, s) = \begin{cases} \mathsf{I}(n, t) + \mathsf{I}(t, s) & \text{if } s < t \\ \mathsf{I}(n/t) + \mathsf{I}(t) & \text{otherwise} \end{cases} \tag{3.1}$$

These two algorithms will serve as building blocks to derive our modular multiplication with better parallelism in the next section.

3.2.2 Multipartite modular multiplication

Classical modular multiplication use either Barrett or Montgomery method. In [KT08], it has been proposed to use them together to reduce both ends of the product, and therefore exhibit a natural two fold parallelism. Let $A = A_0 + \beta^{n/2}A_1$, B and P be n word integers, then bipartite method of [KT08] computes a $AB\beta^{-n/2} \bmod P$ through $A_1B \bmod P$ and $A_0B\beta^{-n/2}$ which be can reduced using **PMR** and **PBR** algorithms with $t = n/2$. This result is improved in [Sak+11] by splitting both operands and use Karatsuba technique to offer a three fold parallelism. In our paper [Cpub-GII13], we further pursue this idea by splitting both operands in k parts in order to reach a higher level of parallelism.

Let $0 \leq A, B < P < \beta^n$ and $\gcd(P, \beta) = 1$. Splitting A, B into k parts each as $A = \sum_{i=0}^{k-1} A_i \beta^{ni/k}$ and $B = \sum_{i=0}^{k-1} B_i \beta^{ni/k}$, assuming k divides n, the modular product shifted to the right by $n/2$ digits rewrites

$$AB\beta^{-n/2} \bmod P = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} A_i B_j \beta^{d_{i,j}} \bmod P, \tag{3.2}$$

where $d_{i,j} = n(i+j)/k - n/2$ and $-n/2 \leq d_{i,j} \leq 3n/2 - 2n/k$. The figure 3.1 illustrates the splitting of the multiplication for $k = 5$. In such a case, it is easy to see that only some of the partial products must be reduced. In fact, the product above β^n must be reduced with **PBR** while the one below 0 must be reduced with **PMR**.

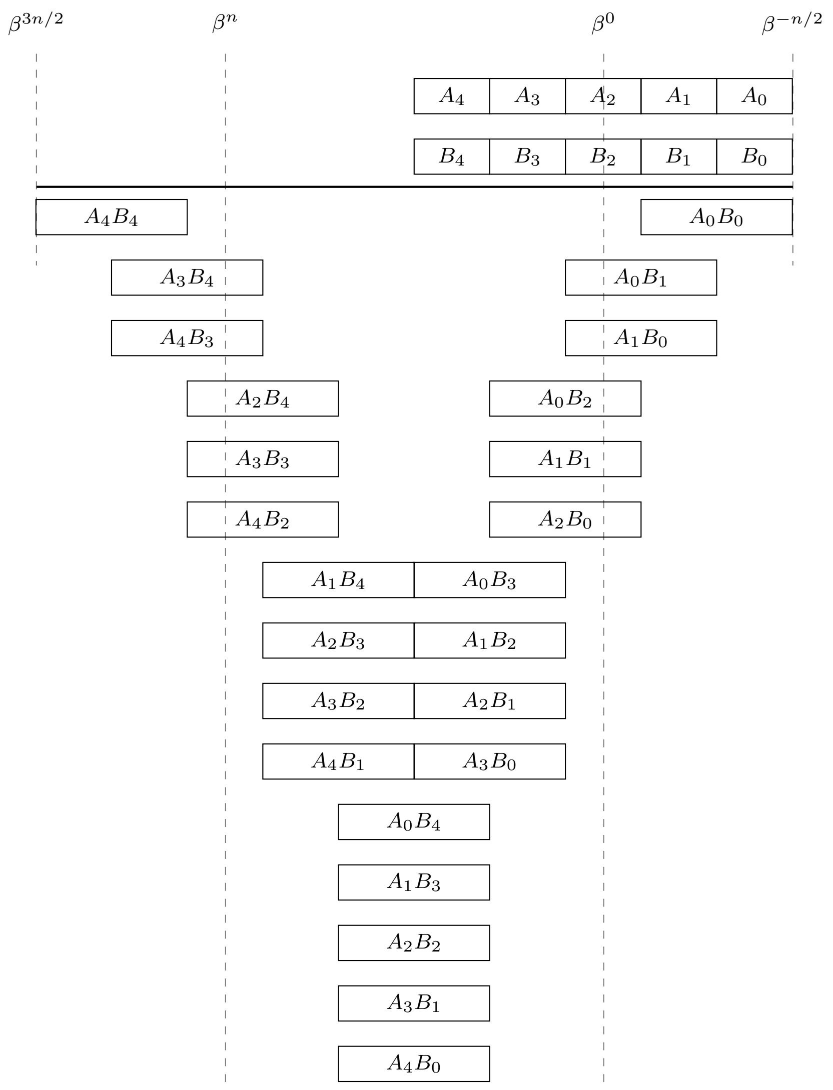


Fig. 3.1: Illustrating the 5-ary multipartite multiplication

The number of such independent reductions is roughly $k^2/4 + O(k)$. Of course summing up all partial products that are aligned to the same digit can reduce this number to exactly $2\lceil k/2 \rceil$. This approach is summarized in the following algorithm

Algorithm 6 k**-ary modular multiplication**

```
Input: integers $k, P, A, B, \mu, \nu$ such that $k \geq 2$, $0 < P < \beta^n$, $\gcd(P, \beta) = 1$, $0 \leq A, B < P$,
    $\mu = -1/P \bmod \beta^{n/2}$, $\nu = \lfloor \beta^{3n/2}/P \rfloor$ where $A = \sum_{i=0}^{k-1} A_i \beta^{in/k}$ and $B = \sum_{i=0}^{k-1} B_i \beta^{in/k}$
Output: $R \equiv AB\beta^{-n/2} \pmod{P}$ with $0 \leq R < P$
  for each $h \in \{0, \dots, 2k-2\}$ do
      $C_h \leftarrow \sum_{i+j=h} A_i B_j$
      if $h \leq \lceil k/2 \rceil - 1$ then
          $R_h \leftarrow$ PMR$(P, C_h, t_h, \mu \bmod \beta^{t_h})$,          where $t_h = n/2 - h(n/k)$
      if $h \geq 2k - 1 - \lceil k/2 \rceil$ then
          $R_h \leftarrow$ PBR$(P, C_h \beta^{hn/k}, t_h, \lfloor \nu / \beta^{n/2 - t_h} \rfloor)$,    where $t_h = (h+2)n/k - 3n/2$
      else
          $R_h \leftarrow C_h \beta^{hn/k}$
  $R \leftarrow \sum_{i=0}^{2k-2} R_h$
  while $R \geq P$ do $R \leftarrow R - P$
  return $R$
```

Lemma 3.2.1. *The complexity of the* **PMR** *or* **PBR** *reduction in the for each loop of algorithm* 6 *is*

$$\begin{cases} \mathsf{I}(\frac{n}{2} - h\frac{n}{k}) + \mathsf{I}(n, \frac{n}{2} - h\frac{n}{k}) & \textit{when } \lceil k/2 \rceil - 2 \leq l \leq 2k - \lceil k/2 \rceil \\ \mathsf{I}(\frac{n}{2} - h\frac{n}{k}, \frac{2n}{k}) + \mathsf{I}(n, \frac{n}{2} - h\frac{n}{k}) & \textit{otherwise} \end{cases}$$

While the overall sequential complexity of algorithm 6 is beyond the classical approach of Montgomery and Barret, its usage becomes more interesting with parallelism. In [Cpub-GII13] and in Izard's PhD [Iza11] it is shown that the parallel complexity is

$$\begin{cases} 2\mathsf{I}(n/2) + I(n, n/2) & \text{for } k = 2, \\ \mathsf{I}(n/3) + \mathsf{I}(n/2) + \mathsf{I}(n, n/2) & \text{for } k = 3, \\ \mathsf{I}(n/k) + \mathsf{I}(n/2, 2n/k) + I(n, n/2) & \text{for } k > 3. \end{cases}$$

The latter results compute each part of the *for each* loop of algorithm 6 in parallel and execute all integer multiplications sequentially. In comparison, Barrett and Montgomery have a parallel complexity of $3\mathsf{I}(n)$ while the bipartite method of [KT08] achieves $\mathsf{I}(n/2) + 2\mathsf{I}(n, n/2)$ and tripartite method of [Sak+11] reaches a parallel complexity of $\mathsf{I}(n/2) + \mathsf{I}(n, n/2)$.

Let us assume a more realistic model where integer multiplication can be parallelized on $T = \theta_1\theta_2$ processors at a parallel cost of $\mathsf{I}(n/\theta_1, n/\theta_2)$ with 2 synchronizations (one after partial products and one after their summation). Of course, we assume that integer multiplication is parallelized naively but this is a rather classical approach for considered cryptographic bit length [SP04]. Applying this technique, we summarize the obtained complexities for parallel modular multiplication in Table 3.1.

The last entry of this table refers to a variant of the k-ary modular multiplication presented above and described in [Cpub-GII13]. The latter approach computes in parallel all the quotients Q arising in the **PMR** and **PBR** calls but do not perform each reduction

Algorithm	Parallel complexity on $\theta_1\theta_2$ cores	# synch.
Montgomery/Barrett	$3\mathsf{I}\left(\frac{n}{\theta_1},\frac{n}{\theta_2}\right)$	6
Bipartite	$2.5\mathsf{I}\left(\frac{n}{\theta_1},\frac{n}{\theta_2}\right)$	(∗) 6
k-ary multipartite ($k=2\theta_1\theta_2/3$)	$(1.5+\frac{9}{4\theta_1\theta_2}+\frac{\theta_1\theta_2}{2})\mathsf{I}\left(\frac{n}{\theta_1},\frac{n}{\theta_2}\right)$	2
k-ary multipartite* ($k=2\theta_1\theta_2/3$)	$(2.5+\frac{9}{4\theta_1\theta_2})\mathsf{I}\left(\frac{n}{\theta_1},\frac{n}{\theta_2}\right)$	3

Tab. 3.1: Parallel complexity of various modular multiplication algorithms. (∗) when $\theta_1\theta_2=2$, the bipartite algorithm requires only 2 synchronizations

independently. Instead, it adds a synchronization and sum all these quotients together to perform only one parallel integer multiplication with P at a cost of $\mathsf{I}(n/\theta_1, n/\theta_2)$. Of course this last optimization drastically improves the complexity of our algorithm but it slightly increases the number of synchronizations. Even with this last approach, the bipartite modular multiplication still provides the lowest count of operations. However, our two methods allow to divides by at least two the number of thread synchronizations. We have done some benchmarks to illustrate the benefit of our approach with two Intel processors X5650 with six 2.66GHz cores each. Our implementation relied on GMP library 5.0.2 and OpenMP with no explicit SIMD instructions. As confirmed by these experiments reported in Table 3.2, the synchronization bareers are significant in practice and they could be dominant when integers size is not large, i.e. less than a hundred words. For instance, for 1024-bits integers (16 words) it seems unlikely that thread parallelisms is worth it since threads management and synchronization incur a too large penalty compared to the sequential execution. When the bit length is larger our multipartite approach remains interesting to lower down the effect of thread synchronization, which is still being significant. However, when the arithmetic count is overshooting this effect, bipartite method might be preferred.

		sizes (in bits)								
	Algorithm	1024	1536	2048	3072	4096	6144	8192	12288	16384
1 Thread	Best seq.	1.32	2.53	4.13	8.18	13.06	26.03	41.10	79.76	125.18
3 Threads	Montgomery	3.63	4.15	4.95	6.71	8.84	13.42	20.11	33.96	50.50
	2-ary multi*	2.59	3.23	3.83	5.49	7.43	12.24	18.26	32.17	48.40
	2-ary multi	1.47	2.04	2.86	4.84	7.45	12.91	19.95	37.59	58.48
4 Threads	Montgomery	3.73	4.34	4.94	6.54	9.60	13.05	19.09	33.09	49.77
	Bipartite	3.93	4.08	4.69	6.27	7.94	12.13	17.12	29.60	44.65
	4-ary multi*	2.75	3.05	3.68	5.06	7.07	11.40	17.38	31.04	48.16
	4-ary multi	1.68	2.14	2.90	4.74	7.20	13.43	20.60	37.88	60.99
6 Threads	Montgomery	4.66	5.10	5.71	6.70	8.72	12.18	17.30	26.82	41.94
	Bipartite	4.82	5.20	5.39	6.45	7.88	10.99	15.16	22.92	34.58
	4-ary multi*	3.32	3.47	3.83	5.13	6.56	9.72	14.48	24.13	36.22
	4-ary multi	1.95	2.42	3.03	4.96	6.91	12.00	17.82	32.08	49.84
8 Threads	Montgomery	7.62	7.99	8.59	10.51	13.01	16.39	20.18	30.59	42.36
	Bipartite	10.12	9.98	10.33	11.05	12.25	15.45	18.90	26.61	36.58
	8-ary multi*	5.85	6.03	6.44	7.57	8.87	12.18	16.06	25.43	37.80
	8-ary multi	3.98	4.29	4.92	6.59	8.84	13.81	19.88	33.92	51.10

Tab. 3.2: Timings (in μs) for several parallel modular multiplication algorithms on 1, 3, 4, 6 and 8 cores, for operands ranging from 1024 to 16384 bits. For a given number of cores and a given size, the gray cells represent the most efficient algorithm

3.3 Simultaneous conversion with residue number system

As previously mentioned, integer division reduces to integer multiplication to achieve the best complexity. This reduction is mostly well understood in the case of the division of a $2n$-word integer by a n-word integers, and it costs $O(\mathsf{I}(n))$ words operations. However in the multi-modular technique that we will develop in the following sections, one of the basic operation is rather to divide an n-word integer by a smaller t-word integer where possibly $n \gg t$. In that specific case, more refined estimates for Euclidean division can be given. Let A and P be respectively n-word and t-word integers then:

- if $n \gg t$, the computation of $(A \text{ quo } P)$ and $(A \text{ rem } P)$ can be done in $O(n/t\,\mathsf{I}(t))$ word operations,
- if we suppose more precisely that $\beta^{t-1} \leq P < \beta^t$ and $n - t \leq t$, the remainder $(a \text{ rem } m)$ can be computed in time $O((n-t)\mathsf{I}(n-t))$.

3.3.1 The Residue Number System

The Residue Number System (RNS) is a non-positional number system that allows to represent finite sets of integers. Let $m_1, m_2, \ldots, m_s \in \mathbb{Z}_{\geq 0}$ be pairwise distinct primes, and let $M = m_1 m_2 \cdots m_s$. Then any integer $A \in \{0, \ldots, M-1\}$ is uniquely determined by its residues $(a_1, a_2, \ldots, a_s)$ where $a_i = (A \text{ rem } m_i)$; this residual representation is called a Residue Number System [BZ10, Section 2.1.3]. Operations such as addition, subtraction, multiplication and exact division are straightforward in this residual representation: by doing the operation separately in each residual class. Conversions between the classical word representation of A and the residual one are potentially costly. For the following discussion, we assume that all m_i's have length at most t; in particular, any A in $\{0, \ldots, M-1\}$ has length at most st. The purpose of the multi-modular approach is to find a sufficiently large integer $M = \Pi_{i=1}^{s} m_i$ such that the sought integer solution lies in $\{0, \ldots, M-1\}$. Hence, the initial problem over the integers can be mapped into each $\mathbb{Z}/m_i\mathbb{Z}$. Therefore, finding the solutions in each residual class $\mathbb{Z}/m_i\mathbb{Z}$ provides a solution to the initial problem as we have have the ring isomorphism $\mathbb{Z}/M\mathbb{Z} \cong \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_s\mathbb{Z}$. The remaining difficulty lies in the conversions between word representation of integers and RNS.

The conversion to the RNS representation (also called multi-modular reduction) amounts to taking an st-word integer A and computing all remainders $(A \text{ rem } m_i)$. Using classical division, we can compute each of them in $O(s\,\mathsf{I}(t))$ word operations, for a total time in $O(s^2\mathsf{I}(t))$. The classical divide-and-conquer approach of [BM74] improves the complexity to $O(\mathsf{I}(st)\log(s))$ word operations (see also [GG13, Theorem 10.24]). The idea is basically to compute separately the remainders $A \bmod (m_1 \ldots m_{s/2})$ and $A \bmod (m_{s/2+1} \ldots m_s)$ and apply the process recursively. The key point is then the fast precomputation of the induced subproduct tree of the m_i's.

The inverse operation that converts integers from RNS to word representation is achieved through Chinese Remaindering Theorem (CRT)[GG13, Section 5.4]. Let A be an integer in $\{0,\ldots,M-1\}$ given by its RNS representation $(a_1,\ldots,a_s)$; then, A is given by

$$A = \left[\sum_{i=1}^{s} (a_i u_i \text{ rem } m_i) M_i\right] \text{rem } M, \tag{3.3}$$

where $M_i = M/m_i$ and $u_i = (1/M_i \text{ rem } m_i)$ for all i. The naive algorithm to recover A from its residues is easily deduced from this expression and it costs $O(s^2\,\mathsf{l}(t)+s\,\mathsf{l}(t)\log(t))$ word operations. The cost is dominated by computing the u_i's and the inner product of the summand. As previously, a similar divide-and-conquer approach is presented in [BM74] to recover A with only $O(\mathsf{l}(st)\log(s)+s\,\mathsf{l}(t)\log(t))$ word operations. The idea is to reconstruct separately $A \bmod (m_1 \ldots m_{s/2})$ and $A \bmod (m_{s/2+1}\ldots m_s)$ which of course can be done recursively, and then apply equation (3.3) to reconstruct A. The only difficulty is to get all the M_i rem m_i necessary for computing the u_i's. In fact, this can be done efficiently by re-using the fast multi-modular reduction to compute each $\lambda_i = M \text{ rem } m_i^2$ as it immediately follows that $M_i \text{ rem } m_i = \lambda_i/m_i$. Finally, the u_i's are computed with s integer gcds.

In terms of implementation, multi-modular reductions using the naive algorithms can benefit from vectorized (SIMD) instructions to save constant factors, and can be easily parallelized. On the other hand, the quasi-linear approaches do not benefit from SIMD instructions in a straightforward manner. Furthermore it does require $O(s\log(s))$ extra memory while the naive approach needs only $O(s)$ extra memory, and the latter can probably be done in-place. Finally, for the sake of completeness, we shall point to a recent variant of these conversions that improves the complexities by a factor $\log\log(s)$ based on FFT-trading [Hoe17].

3.3.2 On simultaneous conversions

For classical problems in linear algebra, such as matrix multiplication, determinant or rank, it is often a good choice to use a multi-modular approach. Even if this kind of approach does not yield the asymptotically fastest ones, their simplicity and their intrinsic parallelism often make them more efficient in practice. In such a situation, the conversions between word representation of integers and RNS have to be done for several entries, usually N^2 for $N \times N$ matrices. For that specific case, the classical fast multi-modular reduction [BM74] cannot be further reduced and it costs $O(N^2\mathsf{l}(st)\log(s))$, contrary to the classical fast multi-modular reconstruction [GG13, Section 5.4] that can share the precomputation of the u_i's to reach a complexity of $O(N^2\mathsf{l}(st)\log(s) + s\mathsf{l}(t)\log(t))$. When the naive approach is used, it seems not trivial to show that one can reduce the quadratic dependency in s. In [Jpub-Dol+18] we propose a precomputation technique that aims to break this dependency. In particular we demonstrate the following theorem

Theorem 3.3.1. *Let $\mathbf{A} = (A_1,\ldots,A_r)$ be a vector of r integers lying in $\{0,\ldots,M-1\}$ where $M = m_1 m_2 \ldots m_s$ with coprime integers $m_i < \beta^t$ for $1 \le i \le s$. Assuming $r \ge s$, the linear map*

$$\phi : \mathbb{Z}_M^r \longrightarrow \mathbb{Z}_{m_1}^r \times \cdots \times \mathbb{Z}_{m_s}^r \text{ such that } \phi(\mathbf{A}) = (\mathbf{A} \text{ rem } m_1, \ldots \mathbf{A} \text{ rem } m_s)$$

can be computed in $O(rs^{\omega-1}\mathsf{I}(t))$ word operations and the inverse map ϕ^{-1} can be computed in $O(rs^{\omega-1}\mathsf{I}(t)+s\mathsf{I}(t)\log(t))$ word operations.

Of course, this theorem does not improve the complexity when naive matrix multiplication ($\omega = 3$) is used, but when $\omega < 3$ this yields a $O(s^{3-\omega})$ speed-up from the naive multi-modular reduction and reconstruction. Besides, the expression of the algorithm in term of matrix multiplication makes it possible to re-use very efficient matrix multiplication libraries (i.e. BLAS). This will improve considerably the practical performance, especially due to cache memory optimization technique [GG08]. When parameters values are favorable, the use of faster matrix multiplication such as Strassen [Str69] will be an opportunity for further practical speed-ups [DGP02; Jpub-DGP08].

The main ingredient behind our result is simply to trade unbalanced divisions, i.e. $A \text{ rem } m_i$, with a matrix-vector product involving integers smaller than the m_i's. Then, we show that the remaining step is just balanced divisions with integers of $O(t\log(s))$ words. For multi-modular reconstruction the trick is quite similar.

Multi-modular reduction: Given the vector **A**, the decomposition of **A** in base β^t is obtained without any operations as it is given by the encoding of its coefficients :

$$\begin{bmatrix} A_1 & \dots & A_r \end{bmatrix} = \begin{bmatrix} 1 & \beta^t & \beta^{2t} & \dots & \beta^{(s-1)t} \end{bmatrix} \begin{bmatrix} a_{1,1} & \dots & a_{r,1} \\ \vdots & & \vdots \\ a_{1,s} & \dots & a_{r,s} \end{bmatrix}.$$

Using the notation $|\cdots|_{m_i}$ for the operation $\text{rem } m_i$, the application of ϕ on that equation gives the following matrix relation

$$\begin{bmatrix} |A_1|_{m_1} & \dots & |A_r|_{m_1} \\ \vdots & & \vdots \\ |A_1|_{m_s} & \dots & |A_r|_{m_s} \end{bmatrix} = \begin{bmatrix} |1|_{m_1} & |\beta^t|_{m_1} & \dots & \left|\beta^{(s-1)t}\right|_{m_1} \\ \vdots & \vdots & & \vdots \\ |1|_{m_s} & |\beta^t|_{m_s} & \dots & \left|\beta^{(s-1)t}\right|_{m_s} \end{bmatrix} \times \begin{bmatrix} a_{1,1} & \dots & a_{r,1} \\ \vdots & & \vdots \\ a_{1,s} & \dots & a_{r,s} \end{bmatrix} - \text{diag}_s(m_i) \times Q$$

where $Q \in \mathbb{Z}_{\geq 0}^{s\times r}$ and $\text{diag}_s(m_i)$ is the $s \times s$ diagonal matrix made from the m_i's. It is not difficult to see that the coefficients of Q are such that $Q_{i,j} < sm_i\beta^t < \beta^{3t}$, recalling that m_i's are distinct primes smaller than β^t.

Hence, the computation of $\phi(\mathbf{A})$ amounts to a matrix multiplication of dimensions $s \times s$ by $s \times r$ with their entries bounded by β^t; and to rs modular reductions of integers with $O(t)$ words. The precomputation phase for a given ϕ corresponds to the computation of $(\beta^t)^j \text{ rem } m_i$ for $1 \leq i, j \leq s$ that is achieved at a cost of $O(s^2\mathsf{I}(t))$. Altogether, one can deduce the claimed cost.

Multi-modular reconstruction (CRT): the inverse of ϕ can be computed similarly. Given the residues of the vector **A**, we define the matrix T as the scaling of the residues by $\text{diag}_s(u_i)$ that is

$$T = \begin{bmatrix} |A_1u_1|_{m_1} & \dots & |A_ru_1|_{m_1} \\ \vdots & & \vdots \\ |A_1u_s|_{m_s} & \dots & |A_ru_s|_{m_s} \end{bmatrix}.$$

Using the decomposition of the M_j's in base β^t such that $M_j = \sum_{i=0}^{s-1} \gamma_{i,j}\beta^{it}$, for $1 \le j \le s$, we can write the inner sum of Equation (3.3) that is $\begin{bmatrix} M_1 & \dots & \mathsf{M}_s \end{bmatrix} T$ as

$$\begin{bmatrix} 1 & \beta^t & \beta^{2t} & \dots & \beta^{(s-1)t} \end{bmatrix} \begin{bmatrix} \gamma_{1,1} & \dots & \gamma_{1,s} \\ \vdots & & \vdots \\ \gamma_{s,1} & \dots & \gamma_{s,s} \end{bmatrix} \times \begin{bmatrix} |A_1 u_1|_{m_1} & \dots & |A_r u_1|_{m_1} \\ \vdots & & \vdots \\ |A_1 u_s|_{m_s} & \dots & |A_r u_s|_{m_s} \end{bmatrix}. \qquad (3.4)$$

The computation of ϕ^{-1} is then achieved by evaluating the expression given in Equation (3.4) followed by a modular reduction by M component-wise. This amounts to exactly rs modular reductions of integers in $\beta^{O(t)}$ for computing T; then to a matrix multiplication of dimensions $s \times s$ by $s \times r$ with their entries bounded by β^t; and finally to r modular reductions by M of integers in $O(\beta^{st})$. Note that the final multiplication by $\begin{bmatrix} 1 & \beta^t & \beta^{2t} & \dots & \beta^{(s-1)t} \end{bmatrix}$ in Equation (3.4) only requires $3r$ additions of st-word integers since the computed coefficients are smaller than β^{3t}. The precomputation phase for a given ϕ is similar to the classical approach that is to compute the u_i's and the M_i's. As previously, assuming $r \ge s$ ensures the validity of our theorem.

For more readability we have rather simplified the presentation of our method given in [Jpub-Dol+18]. In particular, the choice of the decomposition in base β^t leads to the best complexities but our analysis has been done for any decomposition in base $\beta^{t'}$ with $t' \le t$. One of the reason is that for practical purpose it may be more relevant to use a splitting that guarantees that the matrix multiplication result holds within a word-size machine. This is indeed ensured when $s'\beta^{t+t'}$ holds in a machine word, with $s' \ge s$ such that $s' = \lceil st/t' \rceil$. In that cases, the complexities become $O(\mathsf{MM}(r,s,s')\mathsf{I}(t))$ for the multi-modular reduction and $O(\mathsf{MM}(r,s,s')\mathsf{I}(t) + s\mathsf{I}(t)\log(t))$ for the multi-modular reconstruction.

Practical performance: We designed a fine-tuned implementation of the previous approaches in our FFLAS-FFPACK library [Jpub-DGP08]. Here again, our implementation re-uses the efficiency of numerical BLAS libraries for the matrix multiplication involved in the RNS conversions. In particular, this implies some choices for the parameters t and t'. The restriction given by double precision floating point numbers in BLAS libraries sets $s'\beta^{t+t'} < 2^{53}$. The representation of multi-precision integers usually uses 64-bits words. Choosing $\beta = 2$ and a value of t' to be a multiple of 8 ensures that getting the decomposition of integers in base $\beta^{t'}$ has a constant cost[2]. The value of t is usually guided by the multi-modular approach that has to be done. In our case, our motivations comes from linear algebra and matrix multiplication. Following our results in [Jpub-DGP08] this implies that $t < 27$. Hence, we decided to fix the value of t' to be 16 and then t is chosen maximal so that $16 \le t \le 27$. Note that taking $t' = 24$ would limit too much the matrix dimension in the targeted linear algebra operation. Using our parameters, we are able to handle integers up to 189 KBytes that is roughly a million of bits. This is a mild restriction as the fast RNS conversions would be definitively faster in practice for smaller bit length.

We now recall timings obtained in [Jpub-Dol+18] for the computation of RNS conversions in the flagship computer algebra libraries : FLINT [Har10] and Mathemagix [Hoe+12];

[2]In practice this is achieved with only a cast of the pointer to the integer limbs.

and our code in FFLAS-FFPACK. Since all implementations do not share the same value for t and the underlying datatype, we first recall the choices made by each library:

Library	max moduli bit length	datatype
FLINT	59	64-bit integers
Mathemagix	31	32-bit integers
FFLAS	26	64-bit floating point

The benchmarks are made for the parameters of a multi-modular matrix multiplication of matrix dimension 128. This implies that the RNS basis is twice larger than necessary for the conversion from the β-adic representation of the input integer matrices, but it is tight for the reconstruction of the output integer matrix. The choice of matrix dimension 128 is made so that the effect of the precomputation is amortized. In particular, this allows to extract average timings for single integer conversion.

Tab. 3.3: Simultaneous conversions to RNS (time per integer in μs) on Intel Xeon E5-2697 (multi-thread off)

RNS bit length	**FLINT**	**MMX (naive)**	**MMX (fast)**	**FFLAS** [speedup]
2^{8}	0.17	0.34	1.49	0.06 [x 2.8]
2^{9}	0.35	0.75	3.07	0.13 [x 2.7]
2^{10}	0.84	1.77	6.73	0.27 [x 3.1]
2^{11}	2.73	4.26	14.32	0.75 [x 3.6]
2^{12}	7.03	11.01	30.98	1.92 [x 3.7]
2^{13}	17.75	29.86	72.42	5.94 [x 3.0]
2^{14}	50.90	88.95	183.46	21.09 [x 2.4]
2^{15}	165.80	301.69	435.05	80.82 [x 2.0]
2^{16}	506.91	1055.84	1037.79	298.86 [x 1.7]
2^{17}	1530.05	3973.46	2733.15	1107.23 [x 1.4]
2^{18}	4820.63	15376.40	8049.31	4114.98 [x 1.2]
2^{19}	13326.13	59693.64	20405.06	15491.90 [none]
2^{20}	37639.48	241953.39	54298.62	55370.16 [none]

Tables 3.3 and 3.4 report the RNS conversion time on average per integer matrix entry. Each integer matrix conversion is done several time to reach a total time of few seconds that ensures more confidence in the timings. The MMX columns correspond to Mathemagix library where two implementations are available: the asymptotically fast divide-and-conquer approach and the naive one. The FLINT column corresponds to fast divide-and-conquer implementation available in FLINT version 2.5 while FFLAS corresponds to the implementation of our new algorithms with parameters $t' = 16$ and an adaptative value of $t < 27$ that is maximal for the given RNS bit length. For both tables, we add in the FFLAS column the speedup of our code against the fastest code among FLINT and Mathemagix.

One can see from Tables 3.3 and 3.4 that up to RNS bases with $\approx 260\,000$ bits (or equivalently ≈ 4100 words), our new method outperforms existing implementations, even when asymptotically faster methods are used. For an RNS basis with at most 2^{15} bits $= 4$ KBytes our implementation is at least twice as fast. If we just compare ourselves with

Tab. 3.4: Simultaneous conversions from RNS (time per integer in μs) on Intel Xeon E5-2697 (multi-thread off)

RNS bit length	FLINT	MMX (naive)	MMX (fast)	FFLAS [speedup]
2^8	0.63	0.74	3.80	0.34 [x 1.8]
2^9	1.34	1.04	7.40	0.39 [x 3.4]
2^{10}	3.12	1.86	15.53	0.72 [x 4.3]
2^{11}	6.92	4.29	30.91	1.57 [x 4.4]
2^{12}	16.79	12.18	63.53	3.94 [x 4.3]
2^{13}	40.73	43.89	139.16	12.77 [x 3.2]
2^{14}	113.19	144.57	309.88	43.13 [x 2.6]
2^{15}	316.61	502.18	687.45	161.44 [x 2.0]
2^{16}	855.48	2187.65	1502.16	609.22 [x 1.4]
2^{17}	2337.96	10356.08	3519.61	2259.84 [x 1.1]
2^{18}	7295.26	39965.23	9883.07	8283.64 [none]
2^{19}	18529.38	156155.06	22564.36	31382.81 [none]
2^{20}	48413.81	685329.45	59809.07	111899.47 [none]

Tab. 3.5: Precomputation for RNS conversions from/to together (total time in μs)

RNS bit length	FLINT	MMX (naive)	MMX (fast)	FFLAS
2^8	18.58	0.34	1.49	670.30
2^9	19.60	0.75	3.07	708.40
2^{10}	35.80	1.77	6.73	933.30
2^{11}	68.30	4.26	14.32	1958.60
2^{12}	148.10	11.01	30.98	5318.30
2^{13}	338.90	29.86	72.42	17568.60
2^{14}	765.10	88.95	183.46	65323.40
2^{15}	2013.70	301.69	435.05	250234.90
2^{16}	5392.80	1055.84	1037.79	987044.30
2^{17}	13984.80	3973.46	2733.15	4066830.50
2^{18}	35307.50	15376.40	8049.31	17133438.90
2^{19}	89413.10	59693.64	20405.06	69320436.10
2^{20}	243933.90	837116.30	805324.30	244552123.40

Mathemagix naive implementation, which basically has the same theoretical complexity since $\omega = 3$ in our case, we always get a speedup between 3 and 6. This is of course due to the use of better cache optimizations in BLAS.

Table 3.5 provides time evaluations of the precomputation phase for each implementation. As expected, one can see that our method relies on a long phase of precomputation. Despite this long precomputation, our method is really competitive when the number of elements to convert is sufficiently large. For example, if we use an RNS basis of 2^{15} bits, our precomputation phase needs $250ms$ while FLINT's code needs $2ms$. However, for such a basis bit length, our conversion to RNS (resp. from) needs $80\mu s$ ($161\mu s$) per element while FLINT's one needs $165\mu s$ and $316\mu s$. Assuming one needs to convert back and forth with such an RNS basis, our implementation will be faster when more than 1000 integers need to be converted. Note that the code of our precomputation phase is not fully optimized and we should be able to lower this threshold, which makes our approach even relevant for smaller numbers of simultaneous conversions.

3.3.3 Extending to larger moduli

The presented approach is limited by the use of the numerical BLAS to perform matrix multiplication implying at most 26-bit integer moduli. This restriction is not problematic since there are enough of those primes for most applications of multi-modular techniques. However, for the multiplication of polynomials with large integer coefficients the use of fast Fourier transform implies to have primitive 2^d-roots of unity in each $\mathbb{Z}/m_i\mathbb{Z}$, where 2^d bounds the degree of the product. To ensure such property, it is common to use FFT primes of the form $c2^d - 1$ for any integer $c > 1$. However, the number of FFT primes is much smaller than the number of primes: less by a factor of 2^{d-1} asymptotically. To handle larger integer coefficients with these specific moduli, there are two major directions: first, raise the prime size limit of our simultaneous RNS conversions; second, use 3-primes FFT when all FFT primes have been exhausted [Pol71] (see also [GG13, Chapter 8]). Since the latter choice at least triples the runtime per bit, we chose to investigate the first solution. We shall remind that the multi-modular technique is not asymptotically the most efficient solution to this problem. As we already mentioned in 2.1, Kronecker substitution together with integer multiplication yields the best asymptotic method for polynomial multiplication with large integer coefficients. However, for some range of integers and degrees, multi-modular technique might be really competitive in practice, especially if parallelism is used.

We provide in [Jpub-Dol+18] an extension of our approach to handle larger moduli while still relying on BLAS matrix multiplication. Note that already slightly larger moduli will substantially increase the possible bit length of the coefficients: with our forthcoming technique, we will be able to multiply polynomials up to degree 2^{22} and of coefficient with bit length 2^{20} using primes of 42 bits. This is particularly interesting since FFT performances are almost not penalized when 53-bit primes are used instead of 32-bit primes as demonstrated in [HLQ16].

The main observation is that when the moduli m_i are too large, the following restriction from BLAS matrix multiplication is no more true: $s'\beta^{t+t'} < 2^{53}$. Therefore, the solution is to express all m_i's in their expansion in base β^δ such that $s'\beta^{\delta+t'} < 2^{53}$. This means that the integers modulo the m_i's are split in κ chunks of size δ.

Multi-modular reduction: For multi-modular reduction this implies the following rewriting

$$\begin{bmatrix} |1|_{m_1} & \left|\beta^{t'}\right|_{m_1} & \cdots & \left|\beta^{(s'-1)t'}\right|_{m_1} \\ \vdots & \vdots & & \vdots \\ |1|_{m_s} & \left|\beta^{t'}\right|_{m_s} & \cdots & \left|\beta^{(s'-1)t'}\right|_{m_s} \end{bmatrix} = B_0 + \beta B_1 + \cdots + \beta^{\kappa-1} B_{\kappa-1}$$

where each matrix B_i are in $\mathbb{Z}^{s\times s'}_{<\beta^\delta}$. Hence, our previous method for multi-modular reduction is now working individually for each B_i's as δ has been chosen so that $s'\beta^{\delta+t'} < 2^{53}$. This particularly means that only a factor of κ will appear in the complexity. We have demonstrated that our method requires $O(\mathsf{MM}(\kappa s, s', r)\mathsf{I}(\delta))$ word operations. The most remarkable property of this approach is to be as efficient as the one when the moduli are restricted to be less than β^{26}. Indeed, in such a case the complexity is of $O(\mathsf{MM}(s, s', r)\mathsf{I}(t))$. With larger moduli we obtain $O(\mathsf{MM}(\kappa s, s', r)\mathsf{I}(\delta))$ which can be reformulated as $O(\mathsf{MM}(s, s', r)\kappa\mathsf{I}(\delta))$. Using the super-linearity assumption on the complexity function of integer multiplication that is $\kappa\mathsf{I}(\delta) \leq \mathsf{I}(t)$, we obtain exactly the same complexity bound.

Multi-modular reconstruction: Similarly, the multi-modular reconstruction implies the following rewriting of T as

$$T = \begin{bmatrix} |A_1u_1|_{m_1} & \dots & |A_ru_1|_{m_1} \\ \vdots & & \vdots \\ |A_1u_s|_{m_s} & \dots & |A_ru_s|_{m_s} \end{bmatrix} = T_0 + \beta T_1 + \dots + \beta^{\kappa-1}T_{\kappa-1}$$

where each matrix T_i are in $\mathbb{Z}^{s\times r}_{<\beta^\delta}$. Here again, our previous method for multi-modular reconstruction is working individually for each T_i's using the same argument on δ. In that case, we have also demonstrated that our method requires $O(\mathsf{MM}(\kappa s, s', r)\mathsf{I}(\delta)) + s\mathsf{I}(t)\log(t)$ word operations, which boils down to $O(\mathsf{MM}(r, s, s')\mathsf{I}(t) + s\mathsf{I}(t)\log(t))$ using the same argument as before.

Practical performance: This extension of our approach has been implemented in the FFLAS-FFPACK Library. The implementation is restricted to $\kappa = 2$ as it offers a sufficient range of values for integer polynomial multiplication as already mentioned. As before, we chose $\beta = 2$ and $t' = 16$. The value of δ is dynamically chosen to ensure $s'\beta^{\delta+16} \leq 2^{53}$. Since we chose $\kappa = 2$, this means our primes must not exceed $2^{2\delta}$. For small RNS basis of bit length less than 2^{15} ($\simeq$ 4 KBytes), we can choose the maximum value $\delta = 26$. For larger RNS basis, we need to reduce the value of δ, *e.g.* with $\delta = 21$ one can use 42-bit primes and reach an RNS basis of 2^{20} bits ($\simeq$ 131 KBytes). Our implementation is similar to the one with moduli less than 2^{26}, which is in fact the case $\kappa = 1$. In order to speed-up our conversions with larger primes, we always stack the matrices to compute the κ matrix multiplication altogether using only one larger matrix multiplication. In practice, it is best to have such a larger matrices because the peak performance of BLAS is attained above a certain matrix dimension. Furthermore, doubling a matrix dimension may offer an extra level of sub-cubic matrix multiplication in FFLAS-FFPACK.

Table 3.6 is reporting the benchmarks from [Jpub-Dol+18] using similar settings as in the previous section. As matter of comparison, it reports the time of our RNS conversions when $\kappa = 1$, corresponding to the "small" prime of Section 3.3.2. Comparison with the FLINT library is also provided as it is the fastest contestant offering similar prime bit length when $\kappa = 2$ (e.g. 59 bits). The values reported in Table 3.6 confirm our conclusion that the performance is asymptotically similar for different values of κ. However, for small RNS bit length, one may remark slight differences between $\kappa = 1$ and $\kappa = 2$. For this size, the matrix multiplication is not dominant in the complexity and the constant behind second order terms roughly double the cost. Furthermore, our implementation with $\kappa = 2$ does not benefit from all the SIMD vectorization code that has been done with $\kappa = 1$, explaining the variation. Note that for larger RNS bit length, the conversion to RNS with $\kappa = 2$ is faster than the one with $\kappa = 1$. This is of course due to larger matrix multiplications which benefits from the sub-cubic matrix multiplications of FFLAS-FFPACK. This is no longer true for conversions from RNS as the last division step is almost twice as costly as for the method with $\kappa = 1$. Finally, compared to FLINT, our approach can improve performances up to a factor of two. For very large bit length, the fast divide-and-conquer approach of FLINT kicks in and becomes indeed more advantageous.

Tab. 3.6: Simultaneous RNS conversions (time per integer in μs)

	To RNS			From RNS		
RNS bit length m_i	**FLINT** $< 2^{59}$	**FFLAS** $(\kappa = 1)$ $< 2^{27}$	**FFLAS** $(\kappa = 2)$ $< 2^{53}$	**FLINT** $< 2^{59}$	**FFLAS** $(\kappa = 1)$ $< 2^{27}$	**FFLAS** $(\kappa = 2)$ $< 2^{53}$
2^{8}	0.17	0.06	0.15	0.63	0.34	0.63
2^{9}	0.35	0.13	0.24	1.34	0.39	0.70
2^{10}	0.84	0.27	0.53	3.12	0.72	1.39
2^{11}	2.73	0.75	1.20	6.92	1.57	2.46
2^{12}	7.03	1.92	2.92	16.79	3.94	5.15
2^{13}	17.75	5.94	8.01	40.73	12.77	14.98
2^{14}	50.90	21.09	25.05	113.19	43.13	47.54
2^{15}	165.80	80.82	85.38	316.61	161.44	167.93
2^{16}	506.91	298.86	299.11	855.48	609.22	629.69
2^{17}	1530.05	1107.23	1099.52	2337.96	2259.84	2375.98
2^{18}	4820.63	4114.98	4043.68	7295.26	8283.64	8550.81
2^{19}	13326.13	15491.90	15092.94	18529.38	31382.81	33967.42
2^{20}	37639.48	55370.16	67827.24	48413.81	111899.47	121432.66

As we will see in Section 4.2, this improvement for RNS conversions will be important to reach efficient dense linear algebra over multiprecision prime fields that heavily relies on integer matrix multiplication.

Contributions to linear algebra

4

4.1 Introduction

As presented in the main introduction of this manuscript, the history of exact linear algebra has been first impacted by the work of Strassen [Str69] showing first sub-cubic algebraic complexity for matrix multiplication and matrix inversion. Since then matrix multiplication are a key tool to derive algorithm with a complexity upper bounds of $O(n^{\omega})$ or $\tilde{O}(n^{\omega})$ to most of the problems. The last fifty years have seen many algorithms improving the value of ω, the best value being so far $\omega < 2.3729$ [Le 14]. This clearly improves on the naive approach of $\omega = 3$ and also on the first result of Strassen $\omega = 2.807$. While all these results improve theoretically the complexity of dense linear algebra, Strassen's algorithm remains the most efficient sub-cubic approach in practice, and it is now widely available in software. Even if some efforts to implement theoretically faster algorithms have been done, their superiority remain seldom visible in practice [BD16].

Similarly to the Strassen breakthrough, the work of Storjohann [Sto03] impacted the complexity of dense linear algebra over principal rings, more precisely over univariate polynomial rings. In that case, the complexity is shown to have the same cost as matrix multiplication, $\tilde{O}(n^{\omega}d)$ operations over the base ring, while any other approaches could not amortize expression swell, yielding a complexity of $\tilde{O}(n^{\omega+1}d)$: $O(nd)$ being generally the bound on the degree of matrix entries during the calculation. Clearly, this work opened a new breach for faster linear algebra over univariate polynomial or integer rings. The last two decades have seen new efficient algorithms providing non-trivial reductions to polynomial (or integer) matrix multiplication.

In this chapter, we describe two of our contributions that either show practical efficiency of the algebraic reductions or provide new tools to derive reductions to polynomial matrix multiplication.

Section 4.2 is devoted to efficient implementations of dense linear algebra over finite fields, and more specifically to prime fields. While Strassen's approach is relevant when field's operations have almost constant cost, for larger fields the cost of the arithmetic leaves some room for further improvement. Indeed, in that case the problem falls naturally in the framework of dense linear algebra over integers, and its cost is then related to integer matrix multiplication. As we will see, our proposal to use delayed modular reduction leads to great improvement both in practice and in theory. This description is based on the work [Jpub-DGP08; Cpub-DGP04] that gave birth of the FFLAS-FFPACK[1] Library and some of its improvements [Jpub-Dol+18] .

[1] `https://linbox-team.github.io/fflas-ffpack/`

Section 4.3 deals with dense linear algebra with univariate polynomials. Here we focus on the problem of minimal approximant bases as it influences the design of the most efficient algorithms as shown in Figure 1.1. In particular, we describe our result that exhibits the first reduction to polynomial matrix multiplication for that problem, and we provide a quick overview on what has be done so far to extend it to more general cases. We will also give some insights on a novel description of our reduction fitting the online computational model and proving practical benefit for early termination within the block Wiedemann approach. Lastly, we provide an almost optimal certification procedure for the minimal approximant bases problem. All these descriptions are based on [Cpub-GJV03; Cpub-GL14a; Cpub-GN18].

4.2 Efficient dense linear algebra over prime fields

When dealing with linear algebra problems over prime field $\mathbb{F}_p$, there is no problem of expression swell as the elements remain in the integer range $[0, p-1]$ or $[-\frac{p-1}{2}, \frac{p-1}{2}]$ if signed representation is used. As seen in section 3.2, it is classical to perform arithmetic of $\mathbb{F}_p$ through integer operations followed by a modular reduction. Hence, depending on the bit length of the prime p the implementations and the algorithms may vary according to machine word precision.

4.2.1 Half word-size prime fields

Extending the representation beyond the canonical representation, that lets elements of $\mathbb{F}_p$ to be larger than p, is classical and leads to better performance in practice than reducing modulo p after each operation. As proposed in [DGP02], let δ be the word-size machine precision (usually $\delta \leq 64$), if p is such that

$$n\frac{(p-1)^2}{2} < 2^\delta$$

then a matrix multiplication over $\mathbb{F}_p$ with inner dimension n can be done over the integers followed by n^2 divisions by p without any overflow. Note that applying division for each multiplication would increase this number to n^3 and penalize the overall performance as division is usually very expensive on modern processor. As an example, on latest Skylake X Intel architecture unsigned 64-bit integers division requires more than 21 cycles while multiplication and addition need respectively 1 and 0.5 cycle (see Agner Fog instructions table[2]). Of course, one can circumvent the divisions using any of the Barrett or Montgomery methods presented in section 3.2. Nevertheless, this still provides at least a factor of two of slowdown. In [Har14] a similar technique of delayed modular reduction is used to speed-up FFT transforms over prime field in practice. Using similarity with BLAS convention, we call `fgemm` the general matrix multiplication over $\mathbb{F}_p$.

Another key idea in [DGP02] to improve `fgemm` performance is to couple delayed modular reduction with hybrid fast matrix multiplication, that is to use Strassen-Winograd algorithm [Str69; Win71] for few levels of recursion and then switch to the naive algorithm. The latter cubic matrix multiplications are done very efficiently using numerical

[2] https://www.agner.org/optimize/instruction_tables.pdf

libraries (GOTO BLAS/OPEN BLAS [GG08], MKL [Int07], ATLAS [WPD01]) that take advantage of cache optimization and pipelined vector instructions. This choice leads to fix $\delta = 53$ or $\delta = 24$ as either `double` or `float` are used, the latter being twice faster in practice. Using `int64_t` would allow $\delta = 64$ but the performance of integer units are never better their floating point counterpart: sometimes halved and sometimes equivalent (see [Ppub-Gio14]). Furthermore, no standard equivalence to BLAS library exists for integer types yet.

Using Strassen-Winograd with delayed arithmetic incurs intermediate growth in the matrix coefficients that affects the threshold of the modular reductions. If t levels of recursion are used, we show in [Jpub-DGP08] that the reductions can be totally delayed as soon as

$$9^t \left\lfloor \frac{n}{2^t} \right\rfloor \left(\frac{p-1}{2} \right) < 2^\delta.$$

As mentioned in [Per14], a better adaptation of this threshold can be done as only few of the recursive calls are really matching this bound. In practice, the FFLAS-FFPACK code for `fgemm` is using such adaptive criteria by keeping track of matrix coefficient bounds through all recursive calls. From that approach one is able to reach performance for `fgemm` that are similar to or better than its numerical analogues: `dgemm` for double or `sgemm` for float. Nevertheless, the larger the p the higher the number of modular reductions. When n is too large, splitting the matrix in blocks of smaller dimension is used so as to ensure the validity of delayed reduction. As seen in [Per14], the overhead of modular reductions remains limited in practice and it does not prevent from a sub-cubic time behavior. In particular, our work to provide SIMD implementation of Barrett's method for matrices and vector in FFLAS-FFPACK has been crucial for reducing the impact of these reductions. We have conducted this work in FFLAS-FFPACK with our PhD student B. Vialla and also with B. Boyer. Note that similar effort have been done independently in the Mathemagix library [HLQ16].

Reduction to matrix multiplication: Following the work of Strassen [Str69] providing sub-cubic matrix inversion of dense matrices over a finite field, several other problems have been reduced similarly to matrix multiplication: LUP factorization, determinant [BH74]; LQUP factorization, rank and kernel basis [IMH82]; characteristic polynomial [PS07], rank profile [DPS15]. All these problems rely mainly on matrix multiplication and matrix inversion. However, in most cases, inversion can be replaced with triangular system solving to lower the constant in the reduction [Jpub-DGP08]. The latter operation, called `ftrsm`, consists in finding the unique solution matrix $X \in \mathbb{F}_p^{m\times n}$ of the triangular linear system $TX = B$ where $B \in \mathbb{F}_p^{m\times n}$ and $T \in \mathbb{F}_p^{n\times n}$ have a triangular shape. The reduction to matrix multiplication is easily achieved with a recursive divide-and-conquer strategy. In [Jpub-DGP08] we provide a hybrid recursive algorithm for `ftrsm` that enables delayed modular reduction. In particular, we show that delaying completely the reduction incurs an exponential growth of the coefficients, and it is only relevant for small p. Nonetheless, we prove that this growth can be better controlled by performing the reduction only before and after the last recursion step, obtaining then the same bound as for matrix multiplication. Building the LQUP factorization of [IMH82] on top of our delayed `fgemm` and `ftrsm`, and managing the operation in such a way the input matrix is overwritten with the result matrix, allows to reach high efficiency for linear algebra operation over $\mathbb{F}_p$. In particular, this achieves performance similar to the ones LAPACK numerical libraries [And+99]. Since then, many other efforts have been done to provide faster `fgemm` code : improving memory management [Boy+09] or exploiting asymptotically faster algorithm [BD16]; and also to design better matrix factorization algorithms: revealing more structural information [DPS15] or allowing better parallelism [Dum+16a; Dum+14]. A survey on most of these improvements is provided in Pernet's HDR manuscript [Per14].

4.2.2 Multi-precision prime fields

Of course, the limitation of that method is that the prime p must satisfy $p < 2^{26}$. While interesting work is done in the numerical community to provide BLAS and LAPACK libraries for extended precision (double-double, quad-double, ...), the relative performance of these libraries (XBLAS[3], MPACK[4]) are far from being linear with the precision. In fact, it is almost quadratic as subquadratic techniques often incur too many numerical errors. Furthermore, these libraries might use extended precision only for internal computation. It is then neither appropriate nor efficient to rely on such software for computing with prime p that are beyond 2^{26}.

Integer matrix multiplication: Extending the bit length of p beyond the word-size makes the algebraic complexity of matrix multiplication not as tight as it could be. In such a case, the cost of arithmetic with matrix coefficients is no more unitary and depends on the length of the prime p. In particular, with the classical approach we get a complexity of $O(\mathsf{MM}(n)\mathsf{I}(k)) = O(n^\omega \mathsf{I}(k))$ word operations for multiplication of matrices of dimension n over $\mathbb{F}_p$ with $p < 2^{k\delta}$. This complexity can be improved by looking at the problem of integer matrix multiplication. Table 4.1 below summarizes the main known complexity results for the multiplication of $n \times n$ matrices with their entries having at most k words. We consider $d = O(k + \log(n)/\delta)$ that is roughly the number of words for the coefficients in the matrix product. Note that mapping back the result to $\mathbb{F}_p$ does not change these complexities as the modular reductions need $O(n^2\mathsf{I}(d))$ word operations.

Tab. 4.1: Summary of asymptotic complexity for integer matrix multiplication

Method	**Word operations**
algebraic algorithm	$n^\omega \mathsf{I}(d)$
multi-modular naive	$n^\omega d + n^2 d^2$
multi-modular [Jpub-Dol+18]	$n^\omega d + n^2 d^{\omega-1}$
multi-modular [BM74]	$n^\omega d + n^2 \mathsf{I}(d)\log(d)$
Kronecker segmentation and [BS05]	$\left(n^\omega k + n^2 \mathsf{M}(k)\right)\mathsf{I}(\log(kn))$
Kronecker segmentation and [HH18]	$n^\omega k \mathsf{I}(\log(kn)) + n^2 \mathsf{I}(k)$

The multi-modular method aims to compute the matrix product using the RNS representation. Assuming the moduli of the RNS basis hold in a machine word-size, that is below 2^δ, the number of moduli needed to recover the integer matrix product is $O(d)$. Therefore, the modular matrix multiplications cost $O(n^\omega d)$ word operations. The remaining part of the complexity depends on the choice of the algorithm for the RNS conversions, see Section 3.3. The Kronecker segmentation method aims to use polynomial arithmetic.

[3] `https://www.netlib.org/xblas/`
[4] `http://mplapack.sourceforge.net`

The technique is to split the k-word integers into size-k polynomials having their coefficients less than 2^δ, and perform the polynomial matrix product over $\mathbb{Z}_m[X]$, with $m > 2^{2\delta+\log(kn)}$. Using the fast polynomial evaluation/interpolation technique of [BS05] to perform the polynomial matrix product yields the announced complexity. The technique developed in [HH18] is to use a splitting that gives a smaller polynomial degree of $O(k/\log(kn))$, and to replace the composite integer m with a prime $p = O(k/\log(kn))$. In particular, it performs the product modulo p^λ using lifting technique and it uses Bluestein's FFT trick [Blu70] to reduce the evaluations/interpolations of the polynomial matrices to integer matrix products. Choosing good parameters makes the latter integer matrix product to have integer entries with roughly $\log(kn)$ words and it allows to remove the logarithmic factors of the polynomial matrix evaluation/interpolation phase. For practical applications, the Kronecker approach can be simplified since the memory constraint implies that $\log(kn) < \delta$. Indeed, the maximal number of words that can be stored on a single computer is bounded by 2^δ which means that $n^2k < 2^\delta$. Therefore, using three-prime FFT technique [Pol71] with $m = p_1p_2p_3$ such that $p_1, p_2, p_3 < 2^\delta$ allows to hide the factor $\mathsf{I}(\log(kn))$. Of course, the latter approach still incurs a constant overhead.

When the number of words is larger than the matrix dimension, say $n = O(\log(k))$, the best theoretical algorithm is the one in [HH18]. However, it is better in practice to rely on Kronecker segmentation and the three-prime FFT to provide a more efficient computation. While no implementation of [HH18] exist yet, similar performances might be hard to reach: the authors already suggest not to use their algorithm in practice. We shall mention that the fast multi-modular approach incurs a $\log(d)$ factor that cannot be optimized out with CRT as for Kronecker, which makes the latter more suitable in practice. When the matrix dimension is larger than the integers length, i.e. $k = O(\log(n))$, any of the multi-modular approaches would be the most efficient since the cost is dominated by the modular matrix product. When $k = O(n)$ the situation is less clear and all algorithms can be competitive. This clearly depends on the level of optimizations in the implementations.

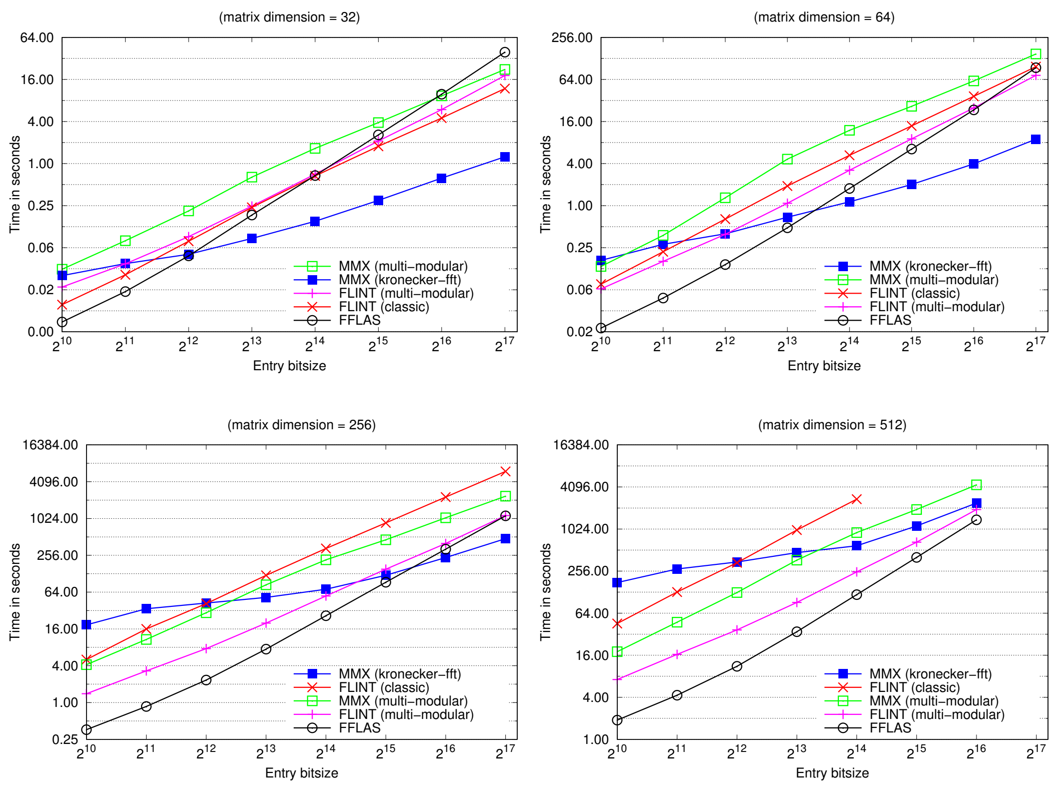
Our work in [Jpub-Dol+18] illustrates that such optimizations can make a difference in practice. In particular, we use our simultaneous conversions with RNS, presented in Section 3.3.2, together with our highly efficient `fgemm` code to provide efficient integer matrix multiplication in FFLAS-FFPACK. In particular, we re-use similar parameters as in Section 3.3.2. We set $\beta = 2$, $t' = 16$ and choose the maximum value of t such that $n\ 2^{2t} \leqslant 2^{53}$ (modular matrix multiplication constraint) and $s'2^{t+t'} \leqslant 2^{53}$ (RNS constraint). As an example, with $t = 20$ our code is able to handle matrices up to dimension 8192 with entries up to 189 KBytes ($\simeq 2^{20.5}$ bits).

Figure 4.1 reports the timings of integer matrix multiplication for different matrix entries bit length (abscissa of plots) and different matrix dimensions (4 different plots) that are representative of all different ranges of parameters. We compared ourselves with FLINT [Har10], and Mathemagix [Hoe+12] :

- FLINT provides two implementations: the algebraic algorithm `FLINT (classic)` with some hand-tuned inline integer arithmetic, and the multi-modular algorithm `FLINT (multi-modular)` which uses the fast divide-and-conquer techniques [BM74] for conversions to and from the RNS.

- The `Algebramix` package of the Mathemagix library provides three implementations: the algebraic algorithm, the multi-modular one and the Kronecker segmentation with FFT. `MMX (kronecker-fft)` reduces multi-precision integers to polynomials over single-precision integers via Kronecker substitution, and then performs FFTs on these polynomials [HLQ16, Section 5.3]. `MMX (multi-modular)` plot corresponds to a hybrid multi-modular approach that uses both quadratic and fast RNS conversions [BM74].
- The `FFLAS` entry corresponds to our multi-modular implementation [Jpub-Dol+18].

Fig. 4.1: Timings for multi-precision integer matrix multiplication (Intel Xeon E5-2697 2.6 GHz, using single thread)[5]



From Figure 4.1, we see that our method improves performance in every cases for some initial range of bit length. However, when the matrix dimension increases, the benefit of our method also tends to increase. We should mention that Kronecker method has the best asymptotic complexity in terms of integer bit length but it turns out to be the worst with respect to matrix dimension. This is confirmed in Figure 4.1 where for $k = 2^{12}$, `MMX (kronecker-fft)` is the fastest code for small matrix dimensions (*e.g.* $n = 32$) while it becomes the worst for larger ones (*e.g.* $n = 512$). As mentioned above, the level of optimizations brought by the extensive use of efficient matrix multiplication, even for RNS conversions, gives a little advance to our approach for the setting $k = \theta(n)$.

Reduction to matrix multiplication: Integer matrix multiplication and modular reduction are the main core operations for fast algorithms in dense linear algebra over multi-precision prime fields. In particular, it is sufficient to plug any integer matrix multiplication algorithms presented above in the classical reduction to yield an efficient method. While with word-size primes this approach leads to the most efficient algorithm, for multi-precision ones it seems not to be the case. To illustrate this, we consider the problem of `ftrsm` for an upper triangular matrix $A \in \mathbb{F}_p^{n\times n}$ and a matrix $B \in \mathbb{F}_p^{n\times n}$. The following splitting of the problem $AX = B$ gives the reduction to matrix multiplication complexity in the algebraic setting:

$$\underbrace{\begin{bmatrix} A_1 & A_2 \\ & A_3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} B_1 \\ B_2 \end{bmatrix}}_{B}$$

with $A_1, A_2, A_3 \in \mathbb{F}_p^{\frac{n}{2}\times\frac{n}{2}}$ and $X_1, X_2, B_1, B_2 \in \mathbb{F}_p^{\frac{n}{2}\times n}$.

Indeed, it is sufficient to solve :

- $A_3X_2 = B_2$ with a recursive call
- $D = B_1 - A_2X_2$ with `fgemm`
- $A_1X_1 = D$ with a recursive call

The complexity satisfies the recurrence $T(n,n) = 2T(n/2,n) + \mathsf{MM}(n/2,n/2,n)$. Assuming $n = 2^t$, we have $T(n,n) = O(n^\omega)$ operations in $\mathbb{F}_p$. Let us now assume that p is a k-word prime (i.e. $p < \beta^k$) and `fgemm` costs $O(n^\omega d + n^2d^2)$ word operations with $d = O(k + \log n)$. The resolution of the recurrence gives $T(n,n) = O(n^\omega d + n^2d^2 \log(n))$ word operations, which is slower than matrix multiplication by a factor $\log(n)$ when $k = O(n^{\omega-2})$. This situation arises for every integer matrix multiplication where a $\log(n)$ shows up in the complexity terms that are quadratic in the matrix dimension. The classical approach that maps the whole original problem to a sufficiently large RNS basis does not give any improvement as it would imply to recover a rational solution that has entries of $O(nk)$ words, see [Jpub-DGP08, Theorem 4.2].

Nevertheless, if `fgemm` is based on multi-modular approach, our delayed approach of [Jpub-DGP08] allows to remove this $\log(n)$ factor. Indeed, the integer growth within `trsm` is coming mainly from the linear combinations of the previous computed result: this corresponds to the computation $A_1^{-1}D = A_1^{-1}(B_1 - A_2X_2)$. As remarked in [Jpub-DGP08, Section 4.3], reducing both $(B_1 - A_2X_2)$ rem p and $A_1^{-1}D$ rem p allows to keep the coefficients below $n(p-1)^2$. Using this remark, we can almost perform the whole computation of `ftrsm` within an RNS basis larger than $n(p-1)^2$. Indeed, all internal calls to integer matrix multiplication will fit this setting except for each final recursive level. For these latter cases, the row matrices D and X_1 of dimension $1 \times n$ have to be reduced modulo p. The latter operations amount to $2n$ conversions between integers and RNS representations for total cost of $O(n^2d^2)$ word operations, thus removing the $\log(n)$ factor. A similar remark has been done in [DPS13] to reduce the number of delayed modular reductions in `ftrsm`. Of course, we can reach $O(n^2\mathsf{I}(d)\log(d))$ using fast RNS conversions, or $O(n^2d^{\omega-1})$ with our simultaneous approach of Section 3.3.

In practice, it is even possible to obtain a better speed-up by avoiding the use of RNS conversions inside the algorithm; only the inputs and the output are converted. Indeed, the problem of modular reduction within an RNS basis, and more generally the extension of RNS bases was well studied [ST67]. The main idea is based on the CRT. Any integer $0 \le A < M = \Pi_{i=1}^s m_i$ for co-prime m_i's is uniquely determined from its residues $a_i = |A|_{m_1} = A$ rem m_i:

$$A = \left(\sum_{i=1}^{s} |a_iu_i|_{m_i} M_i\right) - \alpha M,$$

where $M_i = M/m_i$, $u_i = (1/M_i \text{ rem } m_i)$ for $1 \le i \le s$, and $0 \le \alpha < s$.

Reducing the right hand side of this equation modulo p gives a method for a partial modular reduction in RNS basis. Let $(z_1, \ldots, z_s) \in \mathbb{Z}^s_{\ge 0}$ and $p \in \mathbb{Z}_{>0}$ with

$$z_i = \left| \sum_{i=1}^{s} \left| |a_i u_i|_{m_i} \right|_p |M_i|_p - |\alpha M|_p \right|_p \text{ rem } m_i \text{ for } 1 \le i \le s. \tag{4.1}$$

The unique integer Z satisfying the relation $z_i = Z \text{ rem } m_i$ for $1 \le i \le s$ is such that $Z \equiv A \text{ rem } p$ and $0 \le Z < p \sum_{i=1}^{s} m_i$. Assuming $p < \beta^k$ and the m_i's to be less than β^t this approach yields a partial reduction modulo p: going from β^{st} to $\beta^{\log_\beta(s)+k+t}$.

The main difficulty in Equation (4.1) is to know the value of α from the CRT without computing the integer Z. However, α being a quotient, it can be determined by

$$\alpha = \left\lfloor \sum_{i=1}^{s} \frac{|a_i u_i|_{m_i}}{m_i} \right\rfloor$$

While this computation remains costly using rational numbers, some speed-ups are obtained using either modular arithmetic with $m_{s+1} > s$ [SK89], or floating point arithmetic [MS90] or fixed point arithmetic [Ber95]. The two latter approximated methods are of course faster in practice. Note that $\alpha < s$ which means that all the s^2 values of $|\alpha M \text{ rem } p|_{m_i}$ for $0 \le \alpha, i < s$ can be easily precomputed and stored in a table.

Assuming that we are given r integers $(A_1, \ldots, A_r)$ by their residues $|A_j|_{m_i}$ for $1 \le j \le r$ and $1 \le i \le s$, the summation in Equation (4.1) corresponds to the matrix multiplication

$$H = \begin{bmatrix} |M_1|_p \text{ rem } m_1 & \ldots & |M_s|_p \text{ rem } m_1 \\ \vdots & & \vdots \\ |M_1|_p \text{ rem } m_s & \ldots & |M_s|_p \text{ rem } m_s \end{bmatrix} \times \begin{bmatrix} |A_1 u_1|_{m_1} & \ldots & |A_r u_1|_{m_1} \\ \vdots & & \vdots \\ |A_1 u_s|_{m_s} & \ldots & |A_r u_s|_{m_s} \end{bmatrix}.$$

Computing $\left| |H_{i,j}|_p - |\alpha_j M|_p \right|_{m_i}$ for $1 \le i \le s$ and $1 \le j \le r$ allows to get the evaluation of Equation (4.1). Here, α_j stands for the value α in the CRT for A_j. Assuming all possible constant values are precomputed and $r = O(s)$, this approach costs $O(rs^{\omega-1}\mathsf{I}(t))$ word operations. This approach is similar to the simultaneous conversions with RNS given in Section 3.3.2.

The classical approach for modular reduction in RNS would use back and forth conversions interleaved with a reduction modulo p. In particular, this would induce: r integer reconstructions from RNS, r integer reductions modulo p and r multi-modular reductions to RNS. In fact, our approach trades the r reductions modulo p to only s of them, for $|M_i \text{ rem } p|$, and the r multi-modular reductions to RNS to s of them, corresponding to $|M_i \text{ rem } p|$. Furthermore, if several modular reductions within RNS representation must be done, the latter precomputations are amortized. This is typically the case for `ftrsm` where n simultaneous RNS modular reductions are done n times. This will provide some gain as soon as $s < n^2$.

This approach can be used in the delayed `ftrsm` with n equations over $\mathbb{F}_p$, as soon as the RNS basis satisfies the following constraint:

$$m(p-1)^2 < \frac{\Pi_{i=1}^{s} m_i}{\sum_{i=1}^{s} m_i}$$

Some preliminary results of this method in the FFLAS-FFPACK library shows practical improvement in many cases against the classical reduction scheme to `fgemm`.

PLUQ factorization: Following similar thoughts, the fast algorithms for classical matrix factorizations (LQUP [IMH82], PLE/CUP [JPS13], Gauss Jordan [Sto00], ...) incur a similar logarithmic factor in their complexity over multi-precision prime fields. This result is precisely established for Gauss Jordan algorithm in [Sto00, Corollary 2.12], where the complexity is $O(n^\omega d + n^2 \log(n) \mathsf{I}(d) \log(d))$ word operations for n by n matrices.

The fast PLUQ factorization provided in [DPS13] is a good candidate to avoid the spurious $\log(n)$ factor and to make the complexity exactly the one of `fgemm`. Indeed, as remarked in [DPS13, Theorem 4], using delayed modular reductions allows to reduce the number of modular reductions from $O(n^2 \log(n))$ to only $O(n^2)$ for factoring $A \in \mathbb{F}_p^{n\times n}$. As for `ftrsm`, delaying the modular reduction in PLUQ factorization makes the matrix coefficients bounded by $n(p-1)^2$ during the course of the algorithm. Therefore, using an RNS basis larger than this bound and converting only the coefficients that have to be reduced modulo p, or the ones that have to be checked to be non-zero (pivot search) yields an algorithm of complexity $O(n^\omega d + n^2 \mathsf{I}(d) \log(d))$ word operations, matching with the one of `fgemm`.

Similarly to `ftrsm`, we can use our fast simultaneous modular reduction in RNS to further speed-up the latter approach. In that case, the RNS basis must be chosen so that

$$m(p-1)^2 < \frac{\Pi_{i=1}^{s} m_i}{(\sum_{i=1}^{s} m_i)^2}.$$

Indeed, the matrix coefficients during the course of the algorithm are now bounded by $m(p-1)^2(\sum_{i=1}^{s} m_i)^2$. Note that checking divisibility by p in RNS remains hard to improve and we revert to back and forth conversions. A more precise study of the number of these conversions must be done as it could be as large as n^2 in the worst case. Here again, preliminary results in FFLAS-FFPACK library shows good improvement in practice.

4.3 On the computation of approximant bases

Approximant bases, also called order bases or sigma-bases, are a central tool in the recent advances in linear algebra over $\mathsf{K}[X]$. Indeed, many problems have been reduced to polynomial matrix multiplication through the use of minimal approximant bases, see Figure 1.1.

Let $F = \sum_{i\geq 0} F_i X^i \in \mathsf{K}[[X]]^{m\times n}$ be a matrix of power series and σ a positive integer. A vector $p \in \mathsf{K}[X]^{1\times m}$ is called an approximant of F at order σ if

$$pF \equiv 0 \bmod X^{\sigma}.$$

The set of all such approximant vectors forms a free $\mathsf{K}[X]$-module of rank m, which we define as $\mathscr{A}_\sigma(F) = \{p \in \mathsf{K}[X]^{1\times m} \mid pF \equiv 0 \bmod X^\sigma\}$. An approximant basis for (F,σ) is then a basis of $\mathscr{A}_\sigma(F)$. In order to allow the efficient computation of such bases, algorithms have been designed to find the minimal bases according to their row degree (using lexicographic order). The latter quantity, corresponding to the vector of maximal degree of each row, aims at bounding the "size" of a polynomial matrix, i.e. the number of coefficients in K. More precisely, minimal approximant bases are the ones that are row-reduced [VB92]. In order to provide a better control of this row degree during the course of the algorithms, the notion of minimality is extended to shifted row degree. This notion only amounts to taking the row degree where each row entry is shifted by a power of X accordingly a vector $s \in \mathbb{Z}^m$: $\operatorname{rdeg}([a_1,\ldots,a_m]) = \operatorname{rdeg}_s([a_1X^{s[1]},\ldots,a_mX^{s[m]}])$. This new quantity is compliant with matrix multiplication and it is useful for bounding intermediate matrices during the course of the algorithms. Consequently, shifted minimal approximant bases, or s-minimal approximant bases are the ones that have minimal s-row degree, or similarly that are s-row reduced. Note that this degree extension captures a more general framework that allows to compute classical normal forms, such as Hermite or Popov form, as a specific shifted minimal approximant basis [BLV99; BLV06].

Fast algorithms: Beckermann and Labahn provide in [BL94] the very first efficient algorithm that uses a divide-and-conquer approach on the order σ, yielding a complexity of $\tilde{O}((m^2n+n^2m)\sigma)$ operations in K. In [Cpub-GJV03] we adapt their algorithm to incorporate fast matrix multiplication, thus obtaining a complexity of $\tilde{O}(m^\omega\sigma)$. While being faster for $n = O(m)$, our result could not handle shifted minimal basis and the case $n \ll m$ provides an overshoot in the complexity, especially with $n = 1$ that encompasses Hermite-Padé approximants. The latter case has been solved partially in [Sto06] where only terms of the basis of degree no more than $O(\lceil \frac{n\sigma}{m}\rceil)$ are computed in time $\tilde{O}(m^\omega\lceil\frac{n\sigma}{m}\rceil)$. The salient idea is to transform the vector problem with order σ to a roughly square matrix problem of smaller order $O(\lceil\frac{n\sigma}{m}\rceil)$, and re-use [Cpub-GJV03]. [ZL12] follows this approach to derive an algorithm for computing minimal basis of any n and m at a cost of $\tilde{O}(m^\omega\lceil\frac{n\sigma}{m}\rceil)$. Furthermore, their algorithm is able to find shifted minimal bases for balanced shifts (those having their values in an interval of size $O(\lceil\frac{n\sigma}{m}\rceil)$), and small unbalanced shifts (those having their values in an interval of size $O(n\sigma)$. The latter restrictions are due to the size of minimal approximant bases that can be in $\Theta(m^2n\sigma)$ for very unbalanced shifts [Jea+16, App. B]. Of course, the latter cases are not reachable in the targeted complexity. In such a case [Jea+16] proposes to use further restriction on the minimal bases, that is to compute their shifted Popov canonical form that is always of size $O(mn\sigma)$. The algorithm provided in [Jea+16] achieves a complexity of $\tilde{O}(m^\omega\lceil\frac{n\sigma}{m}\rceil)$ for arbitrary shifts. In fact, this algorithm solves the more general case of approximants that are defined such that $pF \equiv 0 \bmod X^{\mathbf{d}}$, where $\mathbf{d} = (d_1,\ldots,d_n) \in \mathbb{Z}^n_{\geq 0}$ and the modular reduction $X^{\mathbf{d}}$ is done columnwise (i.e. $(pF)_{1,j} \bmod X_j^d$). The obtained complexity is $O(m^{\omega-1}D)$ with $D = \sum_{i=1}^n d_i$. Note that the previous case of $\mathscr{A}_\sigma(F)$ is encompassed by setting $\mathbf{d} = (\sigma,\ldots,\sigma)$. Interested reader on the details of all these results can find a nice survey in [JNV19] and [Nei16a].

4.3.1 Reduction to matrix multiplication

In [Cpub-GJV03], we provide an algorithm called `PM-Basis` that reduces the computation of minimal approximant basis of $\mathscr{A}_\sigma(F)$ to the product of polynomial matrices. The main ingredients behind our approach can be outlined with the following two lemmas.

Lemma 4.3.1. *Let $A, B \in \mathsf{K}[X]^{m\times m}$ with B invertible, $\vec{u}$-row reduced and of $\vec{u}$-row degree $\vec{v}$. Then the $\vec{u}$-row degree of AB is equal to the $\vec{v}$-row degree of A and $AB \in \mathsf{K}[x]^{m\times m}$ is $\vec{u}$-row reduced if and only if $A \in \mathsf{K}[x]^{m\times m}$ is $\vec{v}$-row reduced.*

Lemma 4.3.2. *Let $P^{(1)} \in \mathsf{K}[X]^{m\times m}_{\le\delta_1}$ be a minimal approximant basis of $\mathscr{A}_{\delta_1}(F)$ with row degree $\vec{v}$ and $P^{(2)} \in \mathsf{K}[X]^{m\times m}_{\le\delta_2}$ be a $\vec{v}$-minimal approximant basis of $\mathscr{A}_{\delta_2}(X^{-\delta_1}F \bmod X^{\delta_2})$ with row degree $\vec{u}$, then $P = P^{(2)}P^{(1)} \in \mathsf{K}[X]^{m\times m}_{\le\delta_1+\delta_2}$ is a minimal approximant basis of $\mathscr{A}_{\delta_1+\delta_2}(F)$ and $\vec{u}$ is its row degree.*

Lemma 4.3.1 is a consequence of [Zho12, Lemmas 2.14, 2.18] and it is a key observation to prove Lemma 4.3.2, which appears first in [BL94, Theorem 6.1]. Using these two lemmas, one can derive the divide-and-conquer approach on σ by setting $\delta_1 = \delta_2 = \sigma/2$ and reduce the computation to multiplication of polynomial matrices in $\mathsf{K}[X]^{m\times m}_{\le\sigma/2}$. The remaining ingredient is the final recursion step that aims to computing a minimal approximant basis at order 1. This problem is equivalent to finding some particular row echelon form of a matrix kernel in $\mathsf{K}^{m\times n}$. In [Cpub-GJV03] we show this can be done using any matrix factorization revealing the low row rank profile of the constant term of the given matrix, after some row permutations related to the shift. The complexity is $O(\mathsf{MM}(m,n,r)) = O(mnr^{\omega-2})$ operations in K where r is the rank of the matrix. Applying the divide-and-conquer strategy (DAC) yields the algorithm `PM-Basis` of [Cpub-GJV03] and gives the complexity $\tilde{O}(m^\omega\sigma)$.

4.3.2 Fast iterative algorithms

Let us assume that algorithm $\texttt{Basis}(F_0,\vec{s})$ computes a minimal basis for $\mathscr{A}_1(F_0)$ for given $F_0 \in \mathsf{K}^{m\times n}$ and $\vec{s} \in \mathbb{Z}^m$. The algorithm `M-Basis` of [Cpub-GJV03] is a slow iterative variant of `PM-Basis`. It computes a minimal basis of $F \in \mathsf{K}[[X]]^{m\times n}$ at order σ by computing iteratively all minimal basis of order $0 < i \le \sigma$ of F. This approach is equivalent to using $\delta_2 = 1$ in lemma 4.3.2. This approach has a complexity of $O(m^\omega\sigma^2)$ due to the unbalanced degrees in the polynomial matrix multiplication $p^{(2)}p^{(1)}$ and $X^{-i}p^{(1)}F \bmod X$.

The efficiency of `PM-Basis` is indeed obtained exclusively by balancing the degree of $p^{(2)}$ and $p^{(1)}$, the calls of `Basis` remaining identical. This approach of balancing the degree is classical in computer algebra to get quasi-linear time complexity: see fast multi-modular reduction/reconstruction [BM74] or fast gcd [Knu70; Sch71]. We now provide a variant of `M-Basis` that achieves similar complexity to `PM-Basis` by deferring polynomial matrix multiplication to only the balanced degree cases [Cpub-GL14a]. It can be seen as the derecursivation of `PM-Basis` where the stack of recursive calls are handled by hand.

Let $\nu_2(k)$ be the 2-valuation of a positive integer k that is $\nu_2(k) = n_1$ if $k = \sum_{i=1}^r 2^{n_i}$ is the binary decomposition of k with $n_1 < \cdots < n_r$. By convention $\nu_2(0) = +\infty$. The main idea behind our fast iterative variant is to link the recursive levels of PM-Basis to the binary decomposition of k. Indeed, an approximant basis at order k is made from k products of polynomial matrices of degree at most 1. However, using the binary decomposition of k such an approximant basis is also the product of at most r polynomial matrices of degree 2^{n_i} where $k = \sum_{i=1}^{r} 2^{n_i}$ as defined above. Such polynomials matrices are roughly the one computed by PM-Basis. Our delayed approach consists in keeping such product unevaluated and only perform polynomial matrix products of the same degree.

When computing a new approximant at order k, the value $\nu_2(k)$ tells us how many products can be merged with the unevaluated approximant basis at order $k-1$. In particular, this corresponds to the number of completed recursive levels in PM-Basis. Similarly, the value $\nu_2(k-2^{\nu_2(k)})$ indicates how many levels of recursion we have to go back to find the series used in this step k . The algorithm 7 (called iPM-Basis) given below implements this strategy. It performs exactly the same computation as PM-Basis when $\sigma = 2^t$ and it has a complexity of $O\tilde{}(m^\omega d)$ operations in K.

Algorithm 7 iPM-Basis $(F, \sigma, \vec{s})$

Input: $F \in \mathsf{K}[[X]]^{m\times n}$, $\vec{s} \in \mathbb{Z}^m$ and $\sigma \in \mathbb{Z}_{\geq 0}$
Output: $P \in \mathsf{K}[X]^{m\times m}$ and $\vec{u} \in \mathbb{Z}^m$

```
 1: $F^{(\infty)} = F$
 2: $M_0, \vec{u}_0 = \mathtt{Basis}(F \bmod X, \vec{s})$
 3: for $k = 1$ to $\sigma - 1$ do
 4:     $\nu = \nu_2(k)$, $\nu' = \nu_2(k - 2^\nu)$
 5:     $M^{(\nu)} = ((M_{k-1} \cdot M^{(0)}) \cdot M^{(1)}) \cdots M^{(\nu-1)}$
 6:     $F^{(\nu)} = X^{-2^\nu} M^{(\nu)} F^{(\nu')} \bmod X^{2^\nu}$
 7:     $M_k, \vec{u}_k = \mathtt{Basis}(F^{(\nu)} \bmod X,\ \vec{u}_{k-1})$
 8: Let $\sum_{i=1}^{r} 2^{n_i}$ be the binary decomposition of $\sigma$
 9: $M^{(n_1)} = ((M_{\sigma-1} \cdot M^{(0)}) \cdot M^{(1)}) \cdots M^{(n_1-1)}$
10: return $(M^{(n_1)} \cdot M^{(n_2)}) \cdots M^{(n_r)}, \vec{u}_{\sigma-1}$
```

Note that a similar iterative algorithm for fast multi-modular reconstruction has been proposed in [DGR10]. In the latter work, the recursive call stack is handle through a radix ladder rather than an explicit approach with the valuation in two of the indices. In particular, such iterative approach is used to design finer early termination strategy.

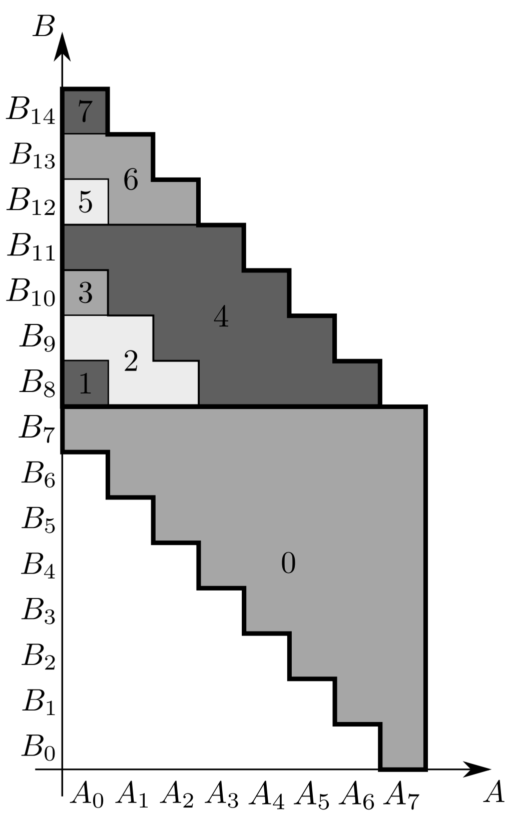
The design of iPM-Basis followed similar motivations, and especially in the context of block Wiedemann algorithm. Indeed, in that specific case, the coefficients of the matrix power series F are given by the computation of $(UA^iV)_{i\geq 1}$, that is done iteratively as $UA(A^{i-1}V)$. While it is expected the number of coefficients of F to be $2N/n + O(1)$, the argument remains probabilistic [Cop94]. In some specific cases, this bound might be loose as experimented in [Cpub-Dum+07]. Theorem 2.12 of [KV05] gives a tighter bound by looking at the invariant factors of the characteristic matrix $xI_N - A$. Let μ be the sum of the degrees of the n largest invariant factors of $xI_N - A$. Then, a minimal matrix generating polynomial of $\sum_{i\geq 1}(UA^iV)X^i$ can be derived from an approximant basis at precision $\lceil \mu/m \rceil + \lceil \mu/n \rceil + O(1)$ [KV05; Tur06]. When the values of μ are assumed to be smaller than the matrix dimension, early termination could be valuable. A classical heuristic termination method is to check if the meaningful rows of the approximant basis keep the same row degree during few consecutive orders. This condition is similar to getting consecutive zero discrepancies in the Berlekamp-Massey algorithm [KL03]. However, such condition does not allow to minimize the dependency on the coefficients of the series within fast algorithms. Indeed, step 6 of `iPM-Basis` (and similarly in `PM-Basis`) computes at step k a matrix middle product with the coefficient of F possibly up to $2k$, e.g. when $k = 2^{\nu_2(k)}$. It is then possible that the number of coefficient of the series is overshot by a factor two, which makes early termination not as efficient as it could be. We propose in [Cpub-GL14a] to use the online computation model to circumvent the latter problem.

Online computation The online computation model aims at providing a framework where the input (resp. output) symbols are read (resp. written) sequentially with the constraint that the k-th output symbol is issued before reading the $(k+1)$-th output symbol. This model is interesting to capture inputs that are implicitly defined by a recursive equation. The online computation model has been revived in computer algebra in 1997 by van der Hoeven for power series multiplication [Hoe97]. While the model has existed in computational complexity since the late '60s [Hen66], only [FS74] proposes a non trivial idea to achieve fast integer multiplication in this model. Of course, almost all naive algorithms for classical operations on integers or polynomials fit well this model: only the first k input coefficients are needed to provide the first k coefficient of the result. However, it is not trivial to find their fast equivalent in this model.

[Hoe97] uses this model to capture the dynamic precision of classical computation with power series and he provides a quasi-linear time algorithms for the truncated multiplication (short product). Since then, complexity refinements were obtained and few extensions to other problems were also proposed: improved truncated multiplications [Hoe02; Hoe03; Hoe07; LS16; Hoe14], resolution of system of equations (linear, algebraic) [Hoe02; BL12; Leb15] , (q)-differential) [Bos+12]. Many of these results are also valid with p-adic numbers; see Lebreton's PhD thesis [Leb12]. As a general remark, it is always the case that fast online algorithms incur a non-constant increase in their complexity compared to their offline quasi-linear variants. In particular the early result [Hoe97; Hoe03] on fast online power series multiplication at precision n gives a cost of $O(\mathsf{M}(n)\log(n))$ operations in K while the best known result to date is $O(\mathsf{M}(n)e^{2\sqrt{\log(2)\log\log(n)}})$ [Hoe14].

Usually online algorithms assume that all inputs follow the model: each input is discovered iteratively. However, an half-line variant is defined for functions having multiple arguments. The idea is that not all arguments are online, a few of them being known completely in advance. For power series product, this means one of the two operands is known in advance. We propose in [Cpub-GL14a] to adapt the half-line truncated power series of [Hoe03] to the need of `iPM-Basis` algorithm. To this end, we use the fact that the update of the series F in `iPM-Basis` (and also in `PM-Basis`) are just specific middle products: $X^{-2^\nu}M^{(\nu)}F^{(\nu')} \bmod X^{2^\nu}$. Here the coefficients of $M^{(\nu)}$ are completely known while the ones of $F^{(\nu')}$ can be discovered iteratively using half-line approach. Figure 4.2 illustrates such half-line middle product with a polynomial $A = \sum_{i=0}^{7} A_i X^i$ completely known in advance and a polynomial $B = \sum_{i=0}^{14} B_i X^i$ known partially up to the degree 7. The meaning of this figure is that sums of diagonal elements correspond to coefficients in the middle product. The integer values in the colored shapes indicate at which iterative step the corresponding products are computed. For instance the first coefficient of the middle product in computed directly at the beginning because all coefficients of

Fig. 4.2: Shifted half-line middle product

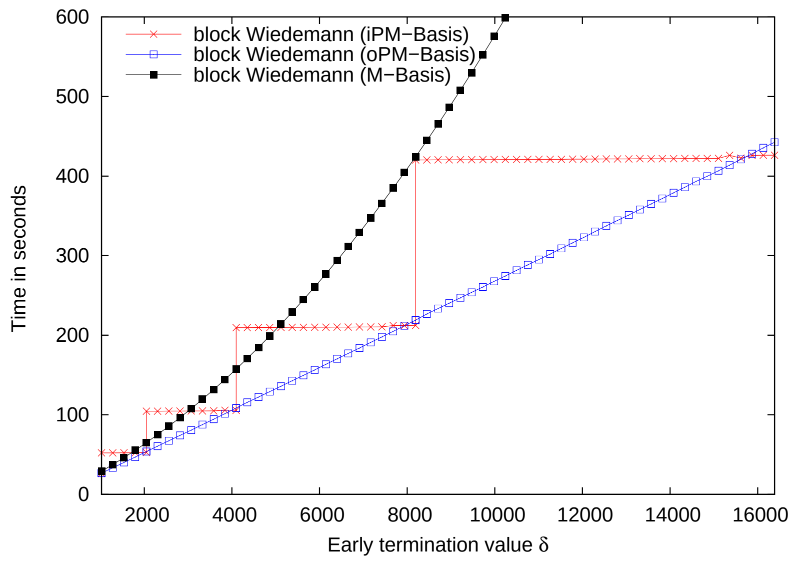


both operands are known. On the contrary, the last coefficient is only computed after 7 iterations when B_{14} is known: it is made from the sum of coefficients computed at iterations $0, 4, 6$ and 7. Remark that each step is made of classical middle-products or short-products.

Replacing line (6) in `iPM-Basis` algorithm with such half-line middle product yields our online algorithm, called `oPM-Basis`, for the computation of minimal approximant basis. Its complexity is $\tilde{O}(m^{\omega}\sigma)$ operations in $\mathbb{K}$. As originally mentioned in [Cpub-GL14a] an extra $\log(\sigma)$ factor is hidden in this notation compared to the complexity of `PM-Basis` and `iPM-Basis`. We shall mention that our latter result is in fact improved by the faster online power series multiplication in [Hoe14], thus reducing the factor to $e^{2\sqrt{\log(2)\log\log(n)}} = o(\log(n))$.

In [Cpub-GL14a], we demonstrate with some experiments that this extra log factor does not impact too much the performance in practice. We also emphasize its benefit for early termination in block Wiedemann algorithm. In particular, this means that we can expect at most a factor two speed-up when the cost of computing coefficients of the matrix power series $\sum_{i\geq 1}(UA^iV)X^i$ dominates. Figure 4.3 illustrates this with time comparisons of the first two parts of the block Wiedemann algorithm: computation of the series $\sum_{i\geq 1}(UA^iV)X^i$ and its minimal matrix generating polynomial through approximant basis. Here the matrix A is square of dimension 2^{17} with coefficients over $\mathbb{F}_p$ for a 23-bit prime p. The block dimensions are chosen to be 16 which means that the matrix power series is potentially computed up to precision 2^{14}. From this experiment we can see that `oPM-Basis` removes the staircase effect of `iPM-Basis` that need to double the knowledge of the series for each power of two. Of course, when the value of early termination is close to $\delta = 2^t - \epsilon$ for very small $\epsilon \geq 0$ the `iPM-Basis` algorithm might be more efficient.

Fig. 4.3: Timing of block Wiedemann with early termination.



4.3.3 Certification of minimal bases

While all fast known algorithms for minimal approximant bases are deterministic, it is not trivial to efficiently verify that a given polynomial matrix is indeed a minimal approximant basis. Such procedure might be considered useless for any deterministic algorithms. However, with the emergence of delegating computation [GKR08], and the use of more and more complex infrastructures and software to tackle challenging computations [Kle+12], it appears nowadays crucial to provide such an posteriori verification for giving more credits to the computed result. In that case, one can either use proof of program as in [GKR08], or use dedicated algorithm to do the verification according to the input and output. The latter approach can use extra information given together with the output, known as a certificate, to ease the verification. The goal is then to find optimal certificates, meaning that one can verify the solution within linear time in the size of the problem instance. Of course, the computation of the certificate must not dominate, and its size must remain minimal. As in delegating computation, an interactive protocol can be used between the "prover", who does the computation, and the "verifier" who wants to establish the correctness of the result (or the proof-of-work). The idea is that the verifier sends challenges to the prover in order to establish a publicly verifiable proof of the commitments. The goal in that context is then to minimize the number of rounds of the protocol and to minimize the communication cost.

Several optimal certificates exist for most linear algebra problems on matrices over a field or over the integer [KNS11; Kal+12; DK14; Dum+16b], but the case of univariate polynomial has been less studied. We shall mention [Luc+18] which provides some work in this direction. Most of these certificates are based on interactive protocols with possibly a constant increase of the prover time. We refer to [Dum18] for a recent survey on efficient certificates for exact linear algebra.

Our result in this section is an almost optimal certificate for the minimal approximant bases problem that can be verifiable within a non-interactive protocol. The time of the prover is not affected by the computation of the certificate, and the size of the latter is small. The main results and their proofs are provided in [Cpub-GN18]. Since minimal approximant bases play a central role in fast linear algebra with univariate polynomial, we believe our non-interactive certificate can help to design other efficient non-interactive verification protocols.

Let us first recall briefly the more general problem of minimal approximant basis at order $\mathbf{d} = (d_1, \dots, d_n) \in \mathbb{Z}^n_{\geq 0}$ together with some of their properties.

For a truncated power matrix series $F \in \mathsf{K}[X]^{m\times n}$, we consider the $\mathsf{K}[X]$-module $\mathscr{A}_{\mathbf{d}}(F) = \{p \in \mathsf{K}[X]^{1\times m} \mid pF = 0 \bmod X^{\mathbf{d}}\}$ where $X^{\mathbf{d}} = \mathrm{diag}(X^{d_1}, \dots, X^{d_n})$ is the $n \times n$ diagonal matrix made of powers of X accordingly to the d_i's. Here $pF = 0 \bmod X^{\mathbf{d}}$ means that $pF = qX^{\mathbf{d}}$ for some $q \in \mathsf{K}[X]^{1\times m}$, and the matrix power series is truncated similarly to get F. For a shift $s \in \mathbb{Z}^m_{\geq 0}$, a matrix $P \in \mathsf{K}[X]^{m\times m}$ is a s-minimal approximant basis of F if and only if it is a s-row reduced basis of $\mathscr{A}_{\mathbf{d}}(F)$ [VB92].

According to this definition, verifying that a matrix P is an s-minimal approximant basis for a given instance $(\mathbf{d}, F, s)$ boils down to the following three properties on P:

1. Minimal: P is s-row reduced. Assuming $T \in \mathsf{K}^{m\times m}$ such that $X^{-\vec{u}}PX^s = T + O(X^{-1})_{X\to\infty}$ with $\vec{u}$ be the shifted s-row degree of P, this corresponds to T having full rank [BLV99]. This can be assessed easily in $O(m^\omega)$ operations in K with fast dense linear algebra. The matrix T is the so-called s-leading row matrix of P.
2. Approximant: the rows of P are approximants. This corresponds to verify the truncated matrix polynomial product $PF = 0 \bmod X^{\mathbf{d}}$, or equivalently that $PF = QX^{\mathbf{d}}$ for a polynomial matrix $Q \in \mathsf{K}[X]^{m\times n}$. This can be done in optimal time $O(\mathsf{Size}(P) + \mathsf{Size}(F))$ operations in K using our result given in Section 2.2.2.
3. Basis: the rows of P generates the module $\mathscr{A}_{\mathbf{d}}(F)$.

While the first two properties can be verified efficiently, the last one seems harder without any other information than F and $\mathbf{d}$ to describe the module. The following lemma establishes a relation between approximant basis and kernel basis that serves to ease the verification of the latter point. It will naturally help to define our certificate.

Lemma 4.3.3. *Let $\mathbf{d} \in \mathbb{Z}^n_{\geq 0}$, $F \in \mathsf{K}[X]^{m\times n}$, and $P \in \mathsf{K}[X]^{m\times m}$. Then P is a basis of $\mathscr{A}_{\mathbf{d}}(F)$ if and only if there exists $Q \in \mathsf{K}[X]^{m\times n}$ such that $[P \;\; Q]$ is a kernel basis for $[F^\mathsf{T} \;\; -X^{\mathbf{d}}]^\mathsf{T}$. If this is the case, then we have $Q = PFX^{-\mathbf{d}}$ and there exist $V \in \mathsf{K}[X]^{m\times m}$ and $W \in \mathsf{K}[X]^{m\times n}$ such that $PV + QW = I_m$.*

From this lemma, we have shifted our problem of being a basis of $\mathscr{A}_{\mathbf{d}}(F)$ to the one of $[P \;\; Q]$ being a kernel basis. The following theorem proven in [Cpub-GN18] shows that only the knowledge of $C = Q(0) \in \mathsf{K}^{m\times n}$ is in fact necessary to ensure P is a basis, and therefore C could serve as certificate.

Theorem 4.3.4. *Let $\mathbf{d} \in \mathbb{Z}^n_{\geq 0}$, $F \in \mathsf{K}[X]^{m\times n}$, and $s \in \mathbb{Z}^m$. A matrix P is an s-minimal approximant basis of $\mathscr{A}_{\mathbf{d}}(F)$ if and only if the following properties hold:*

(i) *P is s-reduced;*

(ii) $\det(P)$ *is a non zero monomial in* $\mathsf{K}[X]$*;*

(iii) *the rows of P are in $\mathscr{A}_{\mathbf{d}}(F)$, that is, $PF = 0 \bmod X^{\mathbf{d}}$;*

(iv) $\begin{bmatrix} P(0) & C \end{bmatrix} \in \mathsf{K}^{m\times(m+n)}$ *has full rank, where C is the coefficient of degree 0 of $PFX^{-\mathbf{d}}$.*

Using this theorem and assuming the matrix C is given, we provide in [Cpub-GN18] a Monte-Carlo algorithm that verifies that a given polynomial matrix P is a s-minimal approximant basis of $\mathscr{A}_{\mathbf{d}}(F)$ at a cost of $O(\mathrm{Size}(P) + \mathrm{Size}(F) + m^{\omega-1}(m+n))$ operations in K. The algorithm is always correct when returning false, while the probability of error of a true answer is less than $1/2$ when K contains at least $2(D+1)$ elements with $D = \sum_{i=1}^{n} d_i$.

Let $\Delta = \sum_{i=1}^{m} r_i$ where $\mathrm{rdeg}(X^{-s}P) = [r_1, \ldots, r_m]$, the algorithm consists of verifying each item as follow:

1. check (i) by computing the rank of the matrix T whose entry i,j is the coefficient of degree r_i of the entry i,j of P. If the matrix is not full rank return `false`;
2. check first part of (iv) by computing the rank of the matrix $[P(0) \quad C]$. If it is not full rank return `false`;
3. To ensure (ii), we use the fact that $\det(P) = X^{\Delta}$ [Kai80, Section 6.3.2]. Therefore, we pick a random $\alpha \in S \subset \mathsf{K}$ and check that $\det(P(\alpha)) = \det(P(1))\alpha^{\Delta}$. If it is not the case return `false`;
4. Finally, we check (iii) and the last part of (iv) by verifying probabilistically that $PF = C \bmod X^{\mathbf{t}}$ where $\mathbf{t} = (d_1+1, \ldots, d_m+1)$. If it is not the case return `false`; otherwise return `true`;

The complexity is easily derived since all operations are either dense linear algebra over K or evaluation of the polynomial matrices P and F. When the dense linear algebra part over K is not dominant our algorithm is optimal. Note that this is mainly the case of interest. In order to derive the probability of error, one need to take the maximum of the error probabilities in step 3 and in step 4 that is no more than $D/\#S$.

Computing the certificate: Such a verification protocol implies that the matrix certificate $C \in \mathsf{K}^{m\times m}$, that is the term of degree 0 of $PFX^{-\mathbf{d}}$, is computed together with the polynomial matrix P by the prover. The verifier receives this two matrices and apply our algorithm to verify that P is correct with good probability. The cost of computing C must be negligible compared to the computation of P. We remind that fastest known algorithms to compute P have a complexity of $T(m,D) = \tilde{O}(m^{\omega-1}D)$ operations in K.

For example, suppose that the dimensions and the orders are balanced: $m = n$ and $\mathbf{d} = (D/m, \ldots, D/m)$. The matrix C is then the coefficient of degree D/m of the polynomial matrix product PF. Thus, C can be computed by D/m multiplications of matrices in $\mathsf{K}^{m\times m}$, which gives a cost of $O(m^{\omega-1}D)$ that is in $o(T(m,D))$ as expected.

The main difficulty to obtain similar cost for the general case is that the degree of both P and F can be unbalanced. While the column degree of F is bounded by $\mathbf{d}$ by definition, no such restrictions exist in general on either the row degree or the column degree of the minimal approximant basis P. Nevertheless, all fast algorithms with complexity $\tilde{O}(m^{\omega-1}D)$ share the property that the size of the approximant basis P is in $O(mD)$. In particular, when the shift satisfies $|s - \min(s)| \in O(D)$ the matrix P is such that $|\mathrm{rdeg}(P)| \in O(D)$ [VB92, Thm 4.1]. For larger shift, the use of shifted Popov form for the basis ensures that $|\mathrm{cdeg}(P)| \in O(D)$ [Jea+16]. In such cases, we provide the following theorem.

Theorem 4.3.5. *Assuming $m \in O(D)$ and either $|\mathrm{rdeg}(P)| \in O(D)$ or $|\mathrm{cdeg}(P)| \in O(D)$, where $D = |\mathbf{d}|$, one can compute the matrix $C = X^{-\mathbf{d}}PF \bmod X$ with $O(m^{\omega-1}D)$ operations in K if $n \le m$ and $O(m^{\omega-1}D\log(n/m))$ operations in K otherwise.*

The main idea behind our approach is to bound the number of non zero entries of given degree in both P and F. Let us denote $\delta = \max(\mathbf{d})$. Hence, for every possible degrees $k < \delta$ the number of rows of P with degree $\ge k$ is no more than $\gamma D/k$ when $|\mathrm{rdeg}(P)| \le \gamma D$. A similar remark applies to F, the number of columns with degree $\ge k$ is no more than D/k. Let us rewrite the matrix C as $C = X^{-\delta}PH \bmod X$ where the j-th column of H corresponds to the j-th column of F multiplied by $X^{\delta - d_j}$.

Assuming $P = \sum_{k=0}^{\delta} P_k X^k$ and $F = \sum_{k=0}^{\delta} F_k X^k$, we then have

$$C = \sum_{k=0}^{\delta} P_k H_{\delta-k} \tag{4.2}$$

We can therefore provide the following lemma on the number of nonzero rows of P_k and the nonzero columns of $H_{\delta-k}$.

Lemma 4.3.6. *Let us define the sets $\mathscr{R}_k = \{i \in \{1,\dots,m\} \mid \mathrm{rdeg}(P_{i,*})\} \ge k\}$ and $\mathscr{D}_k = \{j \in \{1,\dots,n\} \mid d_j \ge k\}$. For a given integer $0 \le k \le \delta$, if $i \notin \mathscr{R}_k$ the i-th row of P_k is zero; if $j \notin \mathscr{D}_k$ the j-th column of F_{d_j-k} is zero. In particular, P_k has at most $\#\mathscr{R}_k \le \gamma D/k$ nonzero rows and $H_{\delta-k}$ has at most $\#\mathscr{D}_k \le D/k$ non zero columns.*

Following this lemma, our algorithm is almost straightforward. Indeed, the idea is to evaluate Equation (4.2) using only the relevant rows and columns of each summand : the rows of P having their indices in $\mathscr{R}_k$ and the columns of H having their indices in $\mathscr{D}_{\delta-k}$. The complexity estimate follows by using the given bound on the size of $\mathscr{R}_k$ and $\mathscr{D}_k$. Note that our description corresponds to the case of $|\mathrm{rdeg}(P)| \in O(D)$. The case $|\mathrm{cdeg}(P)| \in O(D)$ implies similar techniques where both the column indices of P and the row indices of F are chosen in the set of non-zero columns of P_k.

Note. It seems unlikely that efficient computation of the certificate can be extended to the more general case where only the assumption on $\mathrm{Size}(F) \in O(mD)$ is made. Indeed, our technique heavily relies on the degree structure of F to remove unnecessary calculation and then reduce the complexity to the average row/column degree. To our knowledge, only one fast algorithm for minimal bases approximant from [ZL12] does not provide any degree property on F. This appears when the shift is weakly unbalanced: $|\min(s) - s| \in O(D)$. As the more general result of [Jea+16] using shifted Popov form encompasses the latter case, this is not a problem. Nevertheless, some new techniques avoiding the use of degree structure to compute the certificate would be really interesting.

Conclusion and future work

5

5.1 Conclusion

This manuscript covers some of the most representative works we have achieved during the past fifteen years on providing efficient algorithms and implementations in exact linear algebra. Beside improving the complexity of some problems with fine tuned algorithms, we have also provided their efficient implementations in libraries that are internationally renowned. We think our work has been helping to improve the benefits of using effective computer algebra for solving computational mathematics problems.

Among our results, we have shown in section 4 that reductions to matrix multiplication is essential to obtain the best possible complexities while offering practical benefits. In particular, our design of delayed modular arithmetic on top on numerical matrix multiplication yields one the most efficient solution for dense linear algebra over half-word size prime fields to date. Extending this approach with our new RNS conversions of Section 3.3 allows to further broaden these performances to the case of larger prime fields. For linear algebra over univariate polynomials we have shown in Section 4.3 that our approach with `PM-Basis` algorithm or its iterative variant `iPM-Basis` has played an important role for fast algorithms in theory and in practice. In particular, the implementation of block Wiedemann approach can rely on such tool to get as much performance as possible, and some of the intermediate calculation can be checked using our optimal certificate of Section 4.3.3. Finally, for more basic arithmetic operations on integers or polynomials, we have seen that novel faster algorithms are still reachable but it would be hard to make them relevant in practice. In Section 2.3, we further study the multiplication of polynomials with a new eye, aiming to provide algorithms that are both efficient in time and memory. In particular, we exhibit a generic framework that provides self-reductions that are time preserving while decreasing memory requirement. While our approach is still theoretical, we believe such approach would help to design faster code in computer algebra.

5.2 Future work

We detail in the next sections the axis or research we want to continue or to explore in the next few years.

5.2.1 Toward better space complexity

In Section 2.3, we show that one can turn any algorithm for univariate polynomial product (full, middle and short) into an algorithm that is in-place, meaning that no extra memory from the input/output are necessary. This approach is orthogonal to the ones that have been previously done in the community and we hope this new approach will be followed to strengthen the memory efficiency of algorithms. Following this result many challenging tasks remain to be solved to go further in this direction.

Ad hoc optimization for specific algorithms: while we employ a general framework to provide in-place algorithms, the extra constant arising in the time complexity might not be satisfactory compared to the one obtained specifically for a given algorithm. In particular, as seen in Table 2.2, we remark that dedicated in-place algorithms can exploit some features that make these constants not too far from the original algorithms using more memory. It is then reasonable to continue to provide such dedicated in-place algorithms. For instance the Toom-3 method seems to be the natural candidate. Indeed, from the description provided in Section 2.3 with linear algebra, it appears feasible to find a factorization of the Toom-3 matrices (i.e. Δ and Γ) such that they can be applied without any extra memory. Here the major difficulty will be to minimize the number of involved extra operations. Probably, designing a partial additive variant, certainly half-additive, should help.

Another interesting approach would be to use our general reduction scheme and optimize it for a given algorithm or a family of algorithms. For instance, when computing the out-of-place unbalanced multiplications in algorithm 1, one can save some time by evaluating the smaller operands only once (it is done many times in our complexity estimate). Similar remark applies for the interpolation of the coefficients that appear at the same position in the output space. For all algorithms using evaluation/interpolation a few redundant operations could be saved. Following the idea on unbalanced multiplication given in [BZ10, Section 1.3.5], we should be able to further improve the complexity of our reduction scheme.

Another idea would be to change the space complexity model. As mentioned in the definition of our space complexity models, using an input space that has a read/write access is reflecting nowadays architecture. Therefore, using this memory for storing intermediate calculation might help. Of course, the model must ensure that original inputs are reverted back by the end of the computation. We have some preliminary results that provides similar space complexity as in [Roc09] for the Karatsuba middle product in this model. It slightly improves the time complexity of our generic approach but at an expense of a non-constant extra memory space i.e. logarithmic in the polynomial size. Finally, the extra logarithmic factor for quasi-linear time in-place middle product should be further studied in order to either remove it or prove it to be a lower bound.

Practicability of the in-place reductions: from Section 2.3 we've seen that reductions preserving memory exist between product problems and also that any out-of-place algorithms can be turned in-place. In order to provide efficient in-place implementations it is therefore needed to study in more detail which path from the reductions graph, summarized in Figure 2.4, should be taken. This has to be done for each problem and for each possible targeted complexity. Of course, this study has to be consolidated with extensive experimentations on various architectures in order to emphasize the most efficient choices in practice.

Re-use in-place products for other problems: polynomial (or integer) multiplication is a central operation in computer algebra as many other problems reduce to it in order to achieve the best possible asymptotic complexity. It is then meaningful to see whether such reductions can be turned in-place by re-using our in-place products. Of course, this implies to adapt existing algorithms for a better memory management. For instance, the division problem seems a good candidate as it has been demonstrated in [HQZ04] that using middle product can reduce the time complexity. In fact, the latter also reduces the extra memory requirement. Another fundamental problem that should be studied next is multi-point evaluation/interpolation [GG13; BS05]. Such operations are central in computer algebra but the use of the notorious subproduct tree makes the fast classical algorithms fall into a rather different class of problems. Indeed, classical approaches precompute the whole subproduct tree that is larger than the output space [GG13, Section 10.1]; this clearly makes a description of an in-place variant a relatively more complex task. There, new algorithm designs must be invented to better take care of the memory. This could be achieved by interleaving the computation of some nodes of the subproduct tree whenever needed, while not storing them all. This should raise similar approaches that have been already exhibited for online computations of polynomial products [Leb12].

Finally, our ultimate goal would be to design an in-place fast gcd and its generalized matrix version: e.g. minimal approximant basis. Being able to provide such an algorithm would definitively be an asset for improving memory footprint of block Wiedemann approach. Our idea is to exploit the particularity of the latter problem that is the input operand (the matrix power series $\sum_{i>0} UA^iVX^i$) can be replaced by a smaller one as the algorithm goes on. We think that using this feature could help to design a better in-place polynomial middle product and then improving the general space complexity for the minimal approximant basis.

More generally, the holy grail in this direction would be to be design completely in-place algorithms: meaning inputs are overwritten by the result, without having any output space. Note this has been demonstrated feasible in linear algebra for matrix LU factorization [JPS13] but we haven't seen yet any such results for polynomial or integer multiplication, even using a quadratic time complexity.

The integer case: our generic approach for in-place algorithms has been designed essentially for polynomials. A natural question is to see if it can be applied to the integer case that is more difficult due to carry propagation. Note that only one result reduces the space complexity of integer multiplication in the Karatsuba case [Che16]. The latter approach is only an adaptation from the polynomial case given in [Roc09]. A very interesting result would be to find in-place variant for the Schönage-Strassen integer product algorithm [SS71] that is often used in practice [GKZ07].

Harder problems: for a long time, we have in mind to design fast in-place modular multiplication algorithm as it is important for low memory devices or low power calculations that today only employ an in-place quadratic approach [Ish+17]. Of course, our research did not succeed yet and our experience is that the problem is really hard. One can think of this problem through linear algebra meaning that the modular multiplication is a linear application with a dense matrix that embeds a small structure : a sum of two distinct linear applications. Therefore, the problem reduces to almost in-place dense matrix-vector product, which of course seems very hard to be turned in-place using subquadratic approaches.

5.2.2 Certification

Relation between direct and transposed problems: Section 2.2 shows that short product and middle product have an optimal probabilistic verification procedure, similar to the one used for classic polynomial product. Our approach is based on using linear map and the transposition principle [BLS03] together with *Freivalds*' technique [Fre79]. In fact, our middle product verification procedure is equivalent to the multiplication of the polynomial $F = f_0 + f_1X + \cdots + f_{n-1}X^{n-1}$ by the polynomial $G = \sum_{i=0}^{n-1}(\alpha X)^i$ for a random α. As we demonstrated in [Jpub-Gio18] and section 2.2 the latter operation can be done in $O(n)$ operations. Therefore, a natural question is to see if any other problems being described through linear maps can be optimally verified by finding a sufficiently large family of inputs that makes the problem solvable in optimal time. In our case, the multiplication of a polynomial with another one having its coefficients in geometric progression is optimal and so for the transposed problem: the middle product. Maybe a first target should be the Euclidean division which is the transposed operation of the extension of linear recurrence [BLS03]. A consequence of an optimal verification procedure for Euclidean division would raise the more interesting problem of verifying modular multiplication. In the latter case, the difficulty would be to combine efficiently the two approaches to verify the composition of the two operations: multiplication and reduction. Of course, the goal is to have no knowledge of the quotient, otherwise the problem is easily handled with the verification of two short products.

Extend certificate to other minimal bases of $\mathsf{K}[X]$-submodules of $\mathsf{K}[X]^m$: Section 4.3.3 provides an almost optimal verification procedure for minimal bases of $\mathsf{K}[X]$-submodules of $\mathsf{K}[X]^m$ described by relations on polynomials modulo some powers of X. Our verification strongly uses this property. In particular, this raised the problem of verifying truncated polynomial matrix multiplication (see Section 2.2.2 and [Jpub-Gio18]) and we strongly use the fact that the determinant of the basis is a power of X (see Section 4.3.3). In [VB92; BL00], vector M-Padé and matrix rational interpolant problems are also described through $\mathsf{K}[X]$-submodules of $\mathsf{K}[X]^m$ and they indeed resemble our minimal approximant basis problem. In [Jea+17; Nei16a] this resemblance is used to provide the best asymptotic complexity for the minimal interpolant basis problem, re-using techniques similar to [BL94; Cpub-GJV03]. It seems therefore natural to extend our verification procedure to these problems. In the specific case of interpolant, the submodule is described by polynomial relations modulo some $(X-\alpha_i)^{\delta_i}$ for a set of points $(\alpha_1, \dots, \alpha_n)$ and a set of integers $\{\delta_1, \dots, \delta_n\}$. In that case, verifying the relations can be done efficiently by evaluating at the given points α_i. For the minimal basis verification our approach using the determinant of the basis no longer works, indeed we do not have an a priori knowledge of its roots and their multiplicities. Nevertheless, this information can be carried out into a new certificate as they are computed by current fast algorithms [Nei16a]. In a more general setting, the polynomial relations in the module is described by a multiplication matrix: in triangular form [BL00] or in block Jordan form [Jea+17]. In those cases, the roots of the basis determinant are no longer explicitly computed by fast algorithms. Hence, further investigations are needed to provide a relevant certificate.

Non-interactive certificates in linear algebra: interactive proof-of-work has been extensively used recently to prove optimal certificates in linear algebra [DKT15; DK14; Dum+16b]. While these results have a great importance from a theoretical point of view, their implementation might be rather complicated, even if they can be turned non-interactive using some cryptographic hash functions. However, using such certificates to automatically verify software seems inappropriate. We hope that providing dedicated non-interactive certificates would be a real advantage for their wide usage in existing linear algebra software. Following idea of [Luc+18] and our approach for minimal approximant bases may help to get non-interactive certificates for linear algebra over polynomial rings.

5.2.3 Efficient software in exact linear algebra

Exact linear algebra reached an algorithmic maturity where most of the problems have been reduced to matrix multiplication, even in the case of rings embedding expression swell. See Figure 1.1 and [Per14, Figure 1] for a summary of these reductions. These complexity results are fundamental as twenty years back they seemed out of reach. As discussed in Chapter 4 some of these reductions are already implemented in the LinBox library and they already established their superiority on the more classical approaches. For the next few years, our effort still needs to be continued to provide more of these reductions and to harness their efficiency.

Polynomial matrices: the use of polynomial matrix multiplication, minimal approximant basis, and high order lifting technique are the main components for fast dense linear algebra with polynomials. Our effort in LinBox library has been mostly focused on the efficiency of polynomial matrix multiplication and minimal approximant basis. The use of vectorized version of FFT algorithms and BLAS matrix multiplication has been predominant in the efficiency of our code for half word-size prime fields. The underlying structure used by these two approaches are however orthogonal and some conversions are still necessary. These conversions are both memory and time consuming and it is a major priority to avoid them. In fact, polynomial coefficients must be stored contiguously for better efficiency in the FFT and so for the matrix coefficients in the use of BLAS matrix multiplication. To avoid storage conversions, we need to design some hybrid formats that enable contiguous storage on both sides (matrix and polynomial), probably using strides as in BLAS. In case of minimal approximant basis, this structure must also embed some properties to facilitate memory operations such as row/column permutations that are extensively used in `M-Basis` algorithm. As mentioned previously, the structure must also be compliant with memory efficiency as it will be a key point for challenging computation. Apart from being completely in-place, algorithmic trade-off should be found between time and memory efficiency.

Similar trade-of should be found between parallel and sequential algorithm in order to provide the most efficient parallel execution. While it is almost straightforward to distribute matrix row (or column) operations as it is classically done in the block Wiedemann case, it seems less obvious how to do this on the polynomial structure. Indeed, minimal approximant basis of vectors of dimension two is related to gcd. The latter operation is known to be in the NC class, meaning that it has a poly-logarithmic parallel complexity using a polynomial number of processors. While for gcd, the resultant operation is used to achieve this result, the algorithm remains only of theoretic interest as sequential fast gcd is more efficient in practice [Kal10]. Surprisingly, no similar results have been done for the matrix Padé problem that should clearly get more benefit into practice. Following this direction might lead to some improvement for parallel computation in polynomial linear algebra.

Finally, high order lifting technique has not received as much attention as approximant basis, probably because less reductions are implied. However, providing an efficient implementation remains a challenging task while being crucial for row reduction and Smith normal form. This is one major step for polynomial linear algebra in LinBox and more generally in any computer algebra software. This will join a similar effort that have been just started in [HNS19] and applied to the bivariate resultant problem.

Word-size matrix multiplication: : In our FFLAS-FFPACK library, most efforts have been done to treat the case of matrices that either have rather small entries, i.e. less than half a wordsize, or very large entries, i.e. more than a hundred of bits (see Section 4.2). The medium size range around a word-size integer appears to be quite useful in the community while we have not yet provided optimized code for such a range of values. Indeed, our rule of thumb is to used half-word size code to extend to higher precision and then reach the best ratio of performance per bit [Jpub-Dol+18]. However, in that specific range this rule is no more relevant. In that case, it urges to provide efficient matrix multiplication that either relies on exact BLAS like kernel with extended precision, as done for FFT in [HLQ16], or to use Kronecker segmentation to exhibit polynomial with smaller coefficient and perform their product with subquadratic scheme. The main objective of the latter approach would be to find thresholds that minimize the number of matrix multiplication with smaller coefficients.

Block Wiedemann: Our main motivation has been the design of efficient code for block Wiedemann approach. As of today, LinBox provides essentially all core components to achieve this for some medium size challenging computation[1] and essentially for few byte prime fields. Our effort needs to be continued to improve the range of efficiency and to ease its usage. The main difference with the CADO-NFS software is that our block Wiedemann implementation is rather generic and it is entirely exhibited to the high-end users. This specificity makes it difficult to provide high-level API while keeping fine tuning of the code. Some crucial engineer work has to be done in that sense, and we are confident that would be done with the support from Cyril Bouvier that just been freshly hired as a research engineer at LIRMM. His experience with the CADO-NFS project is a real asset for the success of that work.

In [Kle+10] a variant of block Wiedemann approach is presented where the column of the series $S(X) = \sum_{i>0} UA^iVX^i$ are not truncated to the same order. In particular, an adaptation of the matrix Berlekamp-Massey is described to that specific case. This approach is mostly motivated for computations that are distributed on nodes that offer different computing power. Instead of using dynamic load balancing during the algorithm, the idea in [Kle+10] is to compute as many coefficients of the series as in original block Wiedemann but unbalancing the degree on the columns. This problem typically resembles to some approximant basis problem. Nevertheless, in that case minimal approximant bases are related to linear combinations of the columns of S which makes generalized approximant basis modulo $X^{\mathbf{d}}$ with $\mathbf{d} \in \mathbb{N}_{>1}^{n\times 1}$ not directly

[1]not using distributed computing

suited. It seems that the problem is related to classical approximant basis but with a specific shift that compensates the unbalanced column degrees. A proper description and testing of this approach within LinBox would be interesting. A more difficult problem to consider would be when the truncation order is not known in advance, as in the case of online computation. Even if the latter context does not match any applications in cryptography, the theoretical question remains interesting by itself and it could offer more flexibility for distributed parallel computation of the rank of very large sparse matrices [Cpub-Dum+07].

Faster polynomial arithmetic for ring-LWE: Efficient polynomial arithmetic is a major tool in the development of efficient computer algebra software, but this is also the case in lattice-based cryptography. Several major advances in fully homomorphic cryptography (FHE) has emerged recently following the work of Gentry's PhD [Gen09]. In particular, the Learning With Error problem regained a lot of attentions [Reg05] to define fully homomorphic protocols. It has been shown in [LPR10] that using cyclotomic rings helps to speed-up some lattice-based public-key cryptosystems, and especially for the ones using LWE problems. Several practicable post-quantum cryptography protocols are today relying on such polynomial rings, see the recent report from the NIST Post-Quantum Cryptography Standardization Process [Ala+19]. The arithmetic of such cyclotomic rings is then crucial for the efficiency of practical applications. Classical approaches have mainly focused their attention on the power-of-two cyclotomic rings as they ease the calculation while keeping a good level of security. In [LPR13] a general framework to deal with arbitrary cyclotomic rings is presented. This lets some hope to enhance a bit further the security of ring-LWE while still providing efficient operations in the ring. In a recent collaboration with our former PhD student Bastien Vialla, we are interested by the question of fast modular polynomial multiplication in arbitrary cyclotomic rings. Our goal is here a more prospective work to provide efficient basic routines for FHE and more generally to lattice based cryptography.

5.2.4 Sparse polynomial

During the past years, we have concentrated our attention on dense polynomial as we mainly focused on the univariate case. However, when dealing with multivariate polynomials it is more appropriate to use a sparse representation as the number of non zero coefficients remains small compare to the possible monomial supports. A classic way of dealing with these multivariate sparse polynomials is to reduce operations to the univariate case using Kronecker substitution [GG13]. Therefore, the univariate case received a lot of attention and a series of works improved the complexity of sparse polynomial multiplication. A really nice recent survey on these results is available in [Roc18]. In particular, one can distinguish two phases that are: finding the monomial support of the products, and then finding the coefficients. The first phase is the most difficult part and it ends up to be solved with sparse interpolation [Bla79; BT88], while the second phase is using evaluation/interpolation on special sets of points. When the support of the product is known in advance, [HL13] provides a quasi-linear time algorithm in both the settings of algebraic and bit complexity. Similar result holds when the support is not known but it is assumed that no cancellation occurred when doing calculation on the coefficients [AR15], i.e. the support can be computed without knowing the coefficients. In such a situation, while the computation is almost optimal (only some extra logarithmic factors), we do not know yet how to verify such product in less operations, as done for the dense case. With the new insight given in Section 2.2 for verifying truncated products of dense polynomials, it seems feasible to provide faster verification. Indeed, the probability that the product must hold modulo $X^p - 1$ for a random integer p seems to be sufficiently reasonable. In particular, this probability is related to the divisibility by cyclotomic polynomials of the polynomial representing erroneous values in the product. The study of this divisibility property will give a lower bound on the value of p and the complexity should follow.

In a more general way, we collaborate with Bruno Grenet to provide efficient implementations of sparse polynomial arithmetic and to develop new ideas to tackle more difficult problems such as the interpolation or the factorization of sparse polynomials. Our experience on dense polynomial arithmetic should lead us to a fruitful collaboration. Some of the axis that we want to explore are: sparse polynomial division using similar techniques used in [AR15]; switch from the monomial basis to other ones, e.g. Chebyshev as in Section 2.4, or using bounded depth-circuit larger than two [FS15]. An ideal goal would be to provide a library similar to NTL for the sparse case, continuing some efforts already done by the community within the FLINT or the Mathemagix library [GR16; HL14]. This axis of research led to a PhD grant application at the University of Montpellier starting in fall 2019. With Bruno Grenet, we are currently supervising a master student working on probabilistic verification of sparse polynomial product.

Publication list

Refereed journals

[Jpub-Dol+18] J. Doliskani, P. Giorgi, R. Lebreton, and É. Schost. "Simultaneous conversions with the Residue Number System using linear algebra". In: *ACM Transactions on Mathematical Software* 44.3 (2018), 27:1–27:21. DOI: 10.1145/3145573 (cit. on pp. 7, 44, 50, 52, 55, 56, 59, 62–64, 84).

[Jpub-Gio18] P. Giorgi. "A probabilistic algorithm for verifying polynomial middle product in linear time". In: *Information Processing Letters* 139 (2018), pp. 30–34. DOI: 10.1016/j.ipl.2018.06.014 (cit. on pp. 9, 19, 23, 82).

[Jpub-Gio12] P. Giorgi. "On Polynomial Multiplication in Chebyshev Basis". In: *IEEE Transactions on Computers* 61.6 (2012), pp. 780–789. DOI: 10.1145/1391989.1391992 (cit. on pp. 15, 38–40).

[Jpub-DGP08] J.-G. Dumas, P. Giorgi, and C. Pernet. "Dense Linear Algebra over Finite Fields: the FFLAS and FFPACK package". In: *ACM Transactions on Mathematical Software* 35 (2008), 19:1–19:42. DOI: 10.1145/1391989.1391992 (cit. on pp. 6, 7, 10, 51, 52, 59, 61, 65).

Refereed conference proceedings

[Cpub-GGR19] P. Giorgi, B. Grenet, and D. S. Roche. "Generic reductions for in-place polynomial multiplication". In: *Proceedings of the 2019 international symposium on symbolic and algebraic computation*. ISSAC'19. To appear at ISSAC'19. 2019 (cit. on pp. 10, 24, 30, 31, 33, 35, 36).

[Cpub-GN18] P. Giorgi and V. Neiger. "Certification of minimal approximant bases". In: *Proceedings of the 2018 international symposium on symbolic and algebraic computation*. ISSAC'18. ACM, 2018, pp. 67–174. DOI: 10.1145/3208976.3208991 (cit. on pp. 9, 19, 21, 22, 60, 74, 75).

[Cpub-Bre+16] A. Breust, C. Chabot, J.-G. Dumas, L. Fousse, and P. Giorgi. "Recursive double-size fixed precision arithmetic". In: *ICMS: International Congress on Mathematical Software*. Vol. 9725. Lecture Notes in Computer Science. 2016, pp. 223–231. DOI: 10.1007/978-3-319-42432-3_28 (cit. on p. 11).

[Cpub-Boy+14] B. Boyer, J.-G. Dumas, P. Giorgi, C. Pernet, and B.-D. Saunders. “Elements of Design for Containers and Solutions in the LinBox Library”. In: *ICMS: International Congress on Mathematical Software*. Vol. 8592. Lecture Notes in Computer Science. 2014, pp. 654–662. DOI: 10.1007/978-3-662-44199-2_98.

[Cpub-GL14a] P. Giorgi and R. Lebreton. “Online Order Basis and its impact on Block Wiedemann Algorithm”. In: *Proceedings of the 2014 international symposium on symbolic and algebraic computation*. ISSAC’14. ACM, 2014, pp. 202–209. DOI: 10.1145/2608628.2608647 (cit. on pp. 6, 8, 60, 69, 71, 72).

[Cpub-GV14] P. Giorgi and B. Vialla. “Generating Optimized Sparse Matrix Vector Product over Finite Fields”. In: *ICMS: International Congress on Mathematical Software*. Vol. 8592. Lecture Notes in Computer Science. 2014, pp. 685–690. DOI: 10.1007/978-3-662-44199-2_102 (cit. on p. 14).

[Cpub-GII13] P. Giorgi, L. Imbert, and T. Izard. “Parallel Modular Multiplication on Multi-Core Processors”. In: *Proceedings of the 21th IEEE Symposium on Computer Arithmetic, ARITH21*. IEEE Computer Society, 2013. DOI: 10.1109/ARITH.2013.20 (cit. on pp. 13, 44, 45, 47).

[Cpub-BDG10] B. Boyer, J.-G. Dumas, and P. Giorgi. “Exact Sparse Matrix-Vector Multiplication on GPU’s and Multicore Architectures”. In: *Proceedings of PASCO’10: Parallel Symbolic Computation*. ACM, 2010, pp. 80–88. DOI: 10.1145/1837210.1837224 (cit. on pp. 13, 14).

[Cpub-GII09] P. Giorgi, L. Imbert, and T. Izard. “Optimizing elliptic curve scalar multiplication for small scalars”. In: *Mathematics for Signal and Information Processing*. Vol. 7444. Proceedings of SPIE. SPIE Ed., 2009, 74440N. DOI: 10.1117/12.827689 (cit. on p. 14).

[Cpub-GIT09] P. Giorgi, T. Izard, and A. Tisserand. “Comparison of Modular Arithmetic Algorithms on GPUs”. In: *Proceedings of the International Conference on Parallel Computing ParCo*. Vol. 19. IOS Press, 2009, pp. 315–322. DOI: 10.3233/978-1-60750-530-3-315 (cit. on pp. 12, 13, 44).

[Cpub-BDG08] S. Boldo, M. Daumas, and P. Giorgi. “Formal proof for delayed finite field arithmetic using floating point operators”. In: *Proceedings of the 8th Conference on Real Numbers and Computers*. Santiago de Compostela, Spain, July 2008.

[Cpub-Dum+07] J.-G. Dumas, P. Elbaz-Vincent, P. Giorgi, and A. Urbánska. “Parallel computation of the rank of large sparse matrices from algebraic K-theory”. In: *Proceedings of the 2007 international workshop on parallel symbolic computation*. PASCO ’07. ACM, 2007, pp. 43–52. DOI: 10.1145/1278177.1278186 (cit. on pp. 8, 13, 70, 85).

[Cpub-Ebe+07] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. “Faster inversion and other black box matrix computations using efficient block projections”. In: *Proceedings of the 2007 international symposium on symbolic and algebraic computation*. ISSAC ’07. ACM, 2007, pp. 143–150. DOI: 10.1145/1277548.1277569.

[Cpub-GNP07] P. Giorgi, C. Nègre, and T. Plantard. “Subquadratic Binary Field Multiplier in Double Polynomial System”. In: *Proceedings of the International Conference on Security and Cryptography, SECRYPT'07*. 2007, pp. 229–236.

[Cpub-Ebe+06] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. “Solving sparse rational linear systems”. In: *Proceedings of the 2006 international symposium on symbolic and algebraic computation*. ISSAC '06. ACM, 2006, pp. 63–70. DOI: 10.1145/1145768.1145785.

[Cpub-DGP04] J.-G. Dumas, P. Giorgi, and C. Pernet. “FFPACK: finite field linear algebra package”. In: *Proceedings of the 2004 international symposium on symbolic and algebraic computation*. ISSAC '04. ACM, 2004, pp. 119–126. DOI: 10.1145/1005285.1005304 (cit. on p. 59).

[Cpub-GJV03] P. Giorgi, C.-P. Jeannerod, and G. Villard. “On the complexity of polynomial matrix computations”. In: *Proceedings of the 2003 international symposium on symbolic and algebraic computation*. ISSAC '03. ACM, 2003, pp. 135–142. DOI: 10.1145/860854.860889 (cit. on pp. 5–7, 60, 68, 69, 82).

[Cpub-Dum+02] J.-G. Dumas, T. Gautier, M. Giesbrecht, et al. “Linbox: A Generic Library For Exact Linear Algebra”. In: *Proceedings of the 2002 International Congress of Mathematical Software*. World Scientific, 2002.

Others

[Ppub-GL14b] P. Giorgi and R. Lebreton. *Relaxing Order Basis Computation*. Poster at ISSAC'13. Abstract published in ACM Communications in Computer Algebra, 47.3/4 (2014), p. 100–101. 2014. DOI: 10.1145/2576802.2576813.

[Ppub-Gio14] P. Giorgi. *Toward High Performance Matrix Multiplication for Exact Computation*. talk given at SIAM Conference on Parallel Processing for Scientific Computing. http://www.lirmm.fr/~giorgi/seminaire-ljk-14.pdf. 2014 (cit. on pp. 7, 61).

[Ppub-GOS05] P. Giorgi, Z. Olesh, and A. Storjohann. *Implementation of a Las Vegas Integer Matrix Determinant Algorithm*. Poster East Coast Computer Algebra Day. 2005.

[Ppub-BGP03] M. Brassel, P. Giorgi, and C. Pernet. *LUdivine: a Symbolic block LU factorisation for matrices over finite fields using BLAS*. Poster at East Coast Computer Algebra Day. 2003.

References

[Adl94] L. M. Adleman. “The Function Field Sieve”. In: *Proceedings of the First International Symposium on Algorithmic Number Theory*. ANTS-I. London, UK, UK: Springer-Verlag, 1994, pp. 108–121 (cit. on p. 1).

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974 (cit. on p. 43).

[Ala+19] G. Alagic, J. Alperin-Sheriff, D. Apon, et al. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST Pubs. https://doi.org/10.6028/NIST.IR.8240. 2019 (cit. on p. 85).

[Ale01] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001 (cit. on p. 11).

[And+99] E. Anderson, Z. Bai, C. Bischof, et al. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999 (cit. on p. 62).

[AR15] A. Arnold and D. S. Roche. “Output-Sensitive Algorithms for Sumset and Sparse Polynomial Multiplication”. In: *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '15. Bath, United Kingdom: ACM, 2015, pp. 29–36. DOI: 10.1145/2755996.2756653 (cit. on pp. 85, 86).

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. 1st. Cambridge University Press, 2009 (cit. on p. 4).

[Bar+14] R. Barbulescu, C. Bouvier, J. Detrey, et al. “Discrete Logarithm in GF(2809) with FFS”. In: *Public-Key Cryptography – PKC 2014*. Ed. by H. Krawczyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 221–238 (cit. on pp. 13, 24).

[Bar86] P. Barrett. “Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor”. In: *Advances in Cryptology, CRYPTO'86*. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 311–326 (cit. on pp. 13, 43, 44).

[BT97] G. Baszenski and M. Tasche. “Fast polynomial multiplication and convolution related to the discrete cosine transform”. In: *Linear Algebra and its Application* 252.1-3 (1997), pp. 1–25 (cit. on pp. 15, 37, 38, 40, 41).

[BT04] Z. Battles and L. Trefethen. “An extension of MATLAB to continuous fractions and operators”. In: *SIAM J. Sci. Comp* (2004) (cit. on p. 37).

[BS83] W. Baur and V. Strassen. “The Complexity of Partial Derivatives”. In: *Theoretical Computer Science* 22 (1983), pp. 317–330. DOI: 10.1016/0304-3975(83)90110-X (cit. on p. 5).

[BL94] B. Beckermann and G. Labahn. “A Uniform Approach for the Fast Computation of Matrix-Type Padé Approximants”. In: *SIAM J. Matrix Anal. Appl.* 15.3 (1994), pp. 804–823 (cit. on pp. 5, 68, 69, 82).

[BL00] B. Beckermann and G. Labahn. “Fraction-Free Computation of Matrix Rational Interpolants and Matrix GCDs”. In: *SIAM J. Matrix Anal. Appl.* 22.1 (2000), pp. 114–144 (cit. on p. 82).

[BLV99] B. Beckermann, G. Labahn, and G. Villard. “Shifted Normal Forms of Polynomial Matrices”. In: *ISSAC’99*. Vancouver, British Columbia, Canada: ACM, 1999, pp. 189–196 (cit. on pp. 68, 74).

[BLV06] B. Beckermann, G. Labahn, and G. Villard. “Normal forms for general polynomial matrices”. In: *J. Symbolic Comput.* 41.6 (2006), pp. 708–737 (cit. on pp. 6, 68).

[BT88] M. Ben-Or and P. Tiwari. “A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation”. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC ’88. Chicago, Illinois, USA: ACM, 1988, pp. 301–309. DOI: 10.1145/62212.62241 (cit. on p. 85).

[Ben12] A. Benoit. “Fast Semi-numerical Algorithms for Chebyshev Series”. Theses. Ecole Polytechnique X, July 2012 (cit. on p. 38).

[Ber95] D. J. Bernstein. *Multidigit Modular Multiplication With The Explicit Chinese Remainder Theorem*. Tech. rep. 1995 (cit. on p. 66).

[BL12] J. Berthomieu and R. Lebreton. “Relaxed P-adic Hensel Lifting for Algebraic Systems”. In: *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC ’12. Grenoble, France: ACM, 2012, pp. 59–66. DOI: 10.1145/2442829.2442842 (cit. on p. 71).

[Bla79] R. E. Blahut. “Transform Techniques for Error Control Codes”. In: *IBM Journal of Research and Development* 23.3 (May 1979), pp. 299–315. DOI: 10.1147/rd.233.0299 (cit. on p. 85).

[Blu70] L. Bluestein. “A linear filtering approach to the computation of discrete Fourier transform”. In: *IEEE Transactions on Audio and Electroacoustics* 18.4 (Dec. 1970), pp. 451–455. DOI: 10.1109/TAU.1970.1162132 (cit. on p. 63).

[BZ07] M. Bodrato and A. Zanoni. “Integer and Polynomial Multiplication: Towards Optimal Toom-Cook Matrices”. In: *ISSAC’07*. Ed. by C. W. Brown. Waterloo, Ontario, Canada: ACM press, July 2007, pp. 17–24. DOI: http://doi.acm.org/10.1145/1277548.1277552 (cit. on pp. 27, 28).

[BM74] A. Borodin and R. Moenck. “Fast modular transforms”. In: *Journal of Computer and System Sciences* 8.3 (1974), pp. 366–386 (cit. on pp. 49, 50, 62–64, 69).

[BLS03] A. Bostan, G. Lecerf, and É. Schost. “Tellegen’s Principle into Practice”. In: *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’03. Philadelphia, PA, USA: ACM, 2003, pp. 37–44. DOI: 10.1145/860854.860870 (cit. on pp. 18, 19, 82).

[BS05] A. Bostan and É. Schost. “Polynomial evaluation and interpolation on special sets of points”. In: *J. Complexity* 21.4 (2005), pp. 420–446 (cit. on pp. 62, 63, 81).

[Bos+17] A. Bostan, F. Chyzak, M. Giusti, et al. *Algorithmes Efficaces en Calcul Formel*. 1.0. Aug. 2017 (cit. on p. 5).

[Bos+12] A. Bostan, B. Salvy, M. F. I. Chowdhury, É. Schost, and R. Lebreton. “Power Series Solutions of Singular (Q)-differential Equations”. In: *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC ’12. Grenoble, France: ACM, 2012, pp. 107–114. DOI: 10.1145/2442829.2442848 (cit. on p. 71).

[BSS08] A. Bostan, B. Salvy, and É. Schost. “Power series composition and change of basis”. In: *Proceedings of the 2008 International Symposium on Symbolic and Algebraic Computation*. ACM, 2008, pp. 269–276 (cit. on pp. 15, 37, 38).

[BSS10] A. Bostan, B. Salvy, and É. Schost. “Fast Conversion Algorithms for Orthogonal Polynomials”. In: *Linear Algebra and its Applications* 432.1 (Jan. 2010), pp. 249–258 (cit. on pp. 15, 37, 38).

[Boy01] J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. New York: Dover N.Y., 2001 (cit. on p. 37).

[BD16] B. Boyer and J.-G. Dumas. “Matrix Multiplication Over Word-Size Modular Rings Using Approximate Formulas”. In: *ACM Trans. Math. Softw.* 42.3 (2016), 20:1–20:12. DOI: 10.1145/2829947 (cit. on pp. 59, 62).

[Boy+09] B. Boyer, J.-G. Dumas, C. Pernet, and W. Zhou. “Memory Efficient Scheduling of Strassen-Winograd’s Matrix Multiplication Algorithm”. In: *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’09. Seoul, Republic of Korea: ACM, 2009, pp. 55–62. DOI: 10.1145/1576702.1576713 (cit. on pp. 10, 11, 30, 62).

[BZ10] R. Brent and P. Zimmermann. *Modern Computer Arithmetic*. version 0.5.3,http://www.loria.fr/ zimmerma/mca/mca-cup-0.5.3.pdf. Aug. 2010 (cit. on pp. 3, 5, 13, 17, 24–28, 43, 44, 49, 80).

[BJ10] N. Brisebarre and M. Joldeş. “Chebyshev Interpolation Polynomial-based Tools for Rigorous Computing”. In: *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’10. Munich, Germany: ACM, 2010, pp. 147–154. DOI: 10.1145/1837934.1837966 (cit. on p. 37).

[BU11] D. Buchfuhrer and C. Umans. “The complexity of Boolean formula minimization”. In: *Journal of Computer and System Sciences* 77.1 (2011), pp. 142–153. DOI: https://doi.org/10.1016/j.jcss.2010.06.011 (cit. on p. 14).

[BH74] J. R. Bunch and J. E. Hopcroft. “Triangular Factorization and Inversion by Fast Matrix Multiplication”. In: *Mathematics of Computation* 28.125 (1974), pp. 231–236 (cit. on p. 61).

[CK91] D. G. Cantor and E. Kaltofen. “On Fast Multiplication of Polynomials over Arbitrary Algebras”. In: *Acta Informatica* 28.7 (1991), pp. 693–701 (cit. on pp. 5, 17).

[CH90] S. C. Chan and K. L. Ho. “Direct methods for computing discrete sinusoidal transforms”. In: *Radar and Signal Processing, IEE Proceedings F* 137.6 (1990), pp. 433–442 (cit. on p. 37).

[Che16] Y. Cheng. “Space-Efficient Karatsuba Multiplication for Multi-Precision Integers”. In: *CoRR* abs/1605.06760 (2016) (cit. on pp. 24, 81).

[Coo66] S. A. Cook. “On the minimum computation time of functions”. MA thesis. Harvard University, May 1966, pp. 51–77 (cit. on pp. 17, 26, 29, 43).

[CT65] J. Cooley and J. Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19.90 (1965), pp. 297–301 (cit. on pp. 10, 17, 28, 29).

[Cop94] D. Coppersmith. “Solving homogeneous linear equations over GF[2] via block Wiedemann algorithm”. In: *Mathematics of Computation* 62.205 (Jan. 1994), pp. 333–350 (cit. on pp. 1, 2, 5, 8, 70).

[DL78] R. A. Demillo and R. J. Lipton. “A probabilistic remark on algebraic program testing”. In: *Information Processing Letters* 7.4 (1978), pp. 193–195. DOI: http://dx.doi.org/10.1016/0020-0190(78)90067-4 (cit. on pp. 20, 21).

[DH76] W. Diffie and M. E. Hellman. “New directions in cryptography”. In: *IEEE Transactions on Information Theory* IT-22.6 (Nov. 1976), pp. 644–654 (cit. on p. 1).

[DIM05] V. Dimitrov, L. Imbert, and P. K. Mishra. “Efficient and Secure Elliptic Curve Point Multiplication using Double-Base Chains”. In: *Advances in Cryptology, ASIACRYPT’05*. Vol. 3788. Lecture Notes in Computer Science. Springer, 2005, pp. 59–78 (cit. on p. 14).

[DK14] J.-G. Dumas and E. Kaltofen. “Essentially Optimal Interactive Certificates in Linear Algebra”. In: *ISSAC’14*. ACM, 2014, pp. 146–153 (cit. on pp. 9, 73, 83).

[DGP02] J. G. Dumas, T. Gautier, and C. Pernet. “Finite Field Linear Algebra Subroutines”. In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’02. Lille, France: ACM, 2002, pp. 63–74. DOI: 10.1145/780506.780515 (cit. on pp. 7, 51, 60).

[Dum18] J.-G. Dumas. “Proof-of-work certificates that can be efficiently computed in the cloud”. In: *The 20th International Workshop on Computer Algebra in Scientific Computing*. Ed. by V. P. Gerdt, W. Koepf, W. M. Seiler, and E. V. Vorozhtsov. Lille, France, Sept. 2018, pp. 1–17. DOI: 10.1007/978-3-319-99639-4_1 (cit. on p. 74).

[Dum+16a] J.-G. Dumas, T. Gautier, C. Pernet, J.-L. Roch, and Z. Sultan. “Recursion based parallelization of exact dense linear algebra routines for Gaussian elimination”. In: *Parallel Computing* 57 (2016), pp. 235–249. DOI: https://doi.org/10.1016/j.parco.2015.10.003 (cit. on p. 62).

[Dum+14] J.-G. Dumas, T. Gautier, C. Pernet, and Z. Sultan. “Parallel Computation of Echelon Forms”. In: *Euro-Par 2014 Parallel Processing*. Ed. by F. Silva, I. Dutra, and V. Santos Costa. Cham: Springer International Publishing, 2014, pp. 499–510 (cit. on p. 62).

[DGR10] J.-G. Dumas, T. Gautier, and J.-L. Roch. “Generic Design of Chinese Remaindering Schemes”. In: *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*. PASCO ’10. Grenoble, France: ACM, 2010, pp. 26–34. DOI: 10.1145/1837210.1837218 (cit. on p. 70).

[DKT15] J.-G. Dumas, E. Kaltofen, and E. Thomé. “Interactive certificate for the verification of Wiedemann’s Krylov sequence: application to the certification of the determinant, the minimal and the characteristic polynomials of sparse matrices”. working paper or preprint. July 2015 (cit. on pp. 10, 83).

[Dum+16b] J.-G. Dumas, E. Kaltofen, E. Thomé, and G. Villard. “Linear Time Interactive Certificates for the Minimal Polynomial and the Determinant of a Sparse Matrix”. In: *International Symposium on Symbolic and Algebraic Computation*. ISSAC’2016, Proceedings of the 2016 ACM International Symposium on Symbolic and Algebraic Computation. Waterloo, Canada: ACM, July 2016, pp. 199–206. DOI: 10.1145/2930889.2930908 (cit. on pp. 9, 10, 73, 83).

[DPS13] J.-G. Dumas, C. Pernet, and Z. Sultan. “Simultaneous Computation of the Row and Column Rank Profiles”. In: *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*. ISSAC ’13. Boston, Maine, USA: ACM, 2013, pp. 181–188. DOI: 10.1145/2465506.2465517 (cit. on pp. 65, 67).

[DPS15] J.-G. Dumas, C. Pernet, and Z. Sultan. “Computing the Rank Profile Matrix”. In: *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC ’15. Bath, United Kingdom: ACM, 2015, pp. 149–156. DOI: 10.1145/2755996.2756682 (cit. on pp. 61, 62).

[EK97] W. Eberly and E. Kaltofen. “On Randomized Lanczos Algorithms”. In: *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’97. Kihei, Maui, Hawaii, USA: ACM, 1997, pp. 176–183. DOI: 10.1145/258726.258776 (cit. on p. 8).

[El 85] T. El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Proceedings of CRYPTO 84 on Advances in Cryptology*. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18 (cit. on p. 1).

[Far02] J. Farkas. "Theorie der einfachen Ungleichungen." In: *J. Reine Angew. Math.* 124 (1902), pp. 1–27 (cit. on p. 9).

[FS74] M. J. Fischer and L. J. Stockmeyer. "Fast on-line integer multiplication". In: *Journal of Computer and System Science* 9 (1974), pp. 317–331 (cit. on p. 71).

[FS15] M. A. Forbes and A. Shpilka. "Complexity Theory Column 88: Challenges in Polynomial Factorization1". In: *SIGACT News* 46.4 (Dec. 2015), pp. 32–49. DOI: 10.1145/2852040.2852051 (cit. on p. 86).

[Fre79] R. Freivalds. "Fast probabilistic algorithms". In: *Mathematical Foundations of Computer Science*. Vol. 74. Springer Berlin Heidelberg, 1979, pp. 57–69 (cit. on pp. 9, 21, 82).

[FJ05] M. Frigo and S. Johnson. "The Design and Implementation of FFTW3". In: *Proceedings of the IEEE* 93.2 (2005). Special issue on "Program Generation, Optimization, and Platform Adaptation", pp. 216–231 (cit. on p. 40).

[Für07] M. Fürer. "Faster Integer Multiplication". In: *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*. STOC '07. New York, NY, USA: ACM, 2007, pp. 57–66. DOI: 10.1145/1250790.1250800 (cit. on pp. 17, 43).

[GG13] J. v. z. Gathen and J. Gerhard. *Modern Computer Algebra (third edition)*. Cambridge University Press, 2013 (cit. on pp. 3–5, 7, 10, 17, 25, 29, 49, 50, 55, 81, 85).

[GKZ07] P. Gaudry, A. Kruppa, and P. Zimmermann. "A Gmp-based Implementation of SchöNhage-strassen's Large Integer Multiplication Algorithm". In: *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*. ISSAC '07. Waterloo, Ontario, Canada: ACM, 2007, pp. 167–174. DOI: 10.1145/1277548.1277572 (cit. on pp. 43, 81).

[GT07] P. Gaudry and E. Thomé. "The mpFq library and implementing curve-based key exchanges". In: *SPEED: Software Performance Enhancement for Encryption and Decryption*. ECRYPT Network of Excellence in Cryptology. Amsterdam, Netherlands, June 2007, pp. 49–64 (cit. on pp. 12, 13, 44).

[GS66] W. M. Gentleman and G. Sande. "Fast Fourier Transforms: For Fun and Profit". In: *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*. AFIPS '66 (Fall). San Francisco, California: ACM, 1966, pp. 563–578. DOI: 10.1145/1464291.1464352 (cit. on p. 17).

[Gen09] C. Gentry. "A fully homomorphic encryption scheme". crypto.stanford.edu/craig. PhD thesis. Stanford University, 2009 (cit. on p. 85).

[GKR08] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. “Delegating Computation: Interactive Proofs for Muggles”. In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC ’08. Victoria, British Columbia, Canada: ACM, 2008, pp. 113–122. DOI: 10.1145/1374376.1374396 (cit. on pp. 9, 73).

[GO89] G. H. Golub and D. P. O’Leary. “Some History of the Conjugate Gradient and Lanczos Methods”. In: *SIAM Rev.* 31.1 (Mar. 1989), pp. 50–102. DOI: 10.1137/1031003 (cit. on p. 1).

[GG08] K. Goto and R. van de Geijn. “High-performance implementation of the level-3 BLAS”. In: *ACM Transactions on Mathematical Software* 35.1 (2008), p. 4 (cit. on pp. 7, 51, 61).

[Gt16] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 6.1.2. http://gmplib.org/. 2016 (cit. on pp. 11, 25, 27, 40, 44).

[GR16] A. W. Groves and D. S. Roche. “Sparse Polynomials in FLINT”. In: *ACM Commun. Comput. Algebra* 50.3 (Nov. 2016), pp. 105–108. DOI: 10.1145/3015306.3015314 (cit. on p. 86).

[Gup+12] S. Gupta, S. Sarkar, A. Storjohann, and J. Valeriote. “Triangular x-basis decompositions and derandomization of linear algebra algorithms over $K[x]$”. In: *J. Symbolic Comput.* 47.4 (2012), pp. 422–453 (cit. on p. 5).

[GS11] S. Gupta and A. Storjohann. “Computing Hermite Forms of Polynomial Matrices”. In: *ISSAC’11*. San Jose, California, USA: ACM, 2011, pp. 155–162 (cit. on p. 6).

[Hag98] T. Hagerup. “Sorting and Searching on the Word RAM”. In: *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*. STACS ’98. Springer-Verlag, 1998, pp. 366–398 (cit. on p. 3).

[HZ04] G. Hanrot and P. Zimmermann. “A long note on Mulders’ short product”. In: *Journal of Symbolic Computation* 37.3 (2004), pp. 391–401. DOI: 10.1016/j.jsc.2003.03.001 (cit. on p. 18).

[HQZ04] G. Hanrot, M. Quercia, and P. Zimmermann. “The Middle Product Algorithm I”. In: *Applicable Algebra in Engineering, Communication and Computing* 14.6 (Mar. 2004), pp. 415–438. DOI: 10.1007/s00200-003-0144-2 (cit. on pp. 10, 18, 23, 81).

[Har10] W. B. Hart. “Fast library for number theory: an introduction”. In: *Mathematical Software–ICMS 2010*. http://www.flintlib.org/. Springer, 2010, pp. 88–91 (cit. on pp. 52, 63).

[Har14] D. Harvey. “Faster arithmetic for number-theoretic transforms”. In: *Journal of Symbolic Computation* 60 (2014), pp. 113–119. DOI: https://doi.org/10.1016/j.jsc.2013.09.002 (cit. on p. 60).

[HH18] D. Harvey and J. van der Hoeven. “On the complexity of integer matrix multiplication”. In: *Journal of Symbolic Computation* 89 (2018), pp. 1–8. DOI: https://doi.org/10.1016/j.jsc.2017.11.001 (cit. on pp. 62, 63).

[HH19a] D. Harvey and J. van der Hoeven. “Integer multiplication in time O(n log n)”. working paper or preprint. Mar. 2019 (cit. on pp. 5, 17, 43).

[HH19b] D. Harvey and J. van der Hoeven. “Polynomial multiplication over finite fields in time O(n log n)”. working paper or preprint. Mar. 2019 (cit. on pp. 5, 17, 43).

[HHL16] D. Harvey, J. van der Hoeven, and G. Lecerf. “Even faster integer multiplication”. In: *Journal of Complexity* 36 (2016), pp. 1–30. DOI: `https://doi.org/10.1016/j.jco.2016.03.001` (cit. on p. 17).

[HHL17] D. Harvey, J. van der Hoeven, and G. Lecerf. “Faster Polynomial Multiplication over Finite Fields”. In: *J. ACM* 63.6 (Jan. 2017), 52:1–52:23. DOI: `10.1145/3005344` (cit. on pp. 17, 43).

[HR10] D. Harvey and D. S. Roche. “An in-place truncated fourier transform and applications to polynomial multiplication”. In: *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*. ACM, 2010, pp. 325–329 (cit. on pp. 10, 24, 29).

[Hen66] F. C. Hennie. “On-Line Turing Machine Computations”. In: *Electronic Computers, IEEE Transactions on* EC-15.1 (1966), pp. 35–44. DOI: `10.1109/PGEC.1966.264374` (cit. on p. 71).

[Hoe02] J. van der Hoeven. “Relax, but don’t be too lazy”. In: *J. Symbolic. Comput.* 34.6 (2002), pp. 479–542. DOI: `10.1006/jsco.2002.0562` (cit. on p. 71).

[Hoe03] J. van der Hoeven. “Relaxed multiplication using the middle product”. In: *ISSAC’03*. New York: ACM, 2003, pp. 143–147. DOI: `10.1145/860854.860890` (cit. on p. 71).

[Hoe04] J. van der Hoeven. “The truncated Fourier transform and applications”. In: *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*. ISSAC ’04. Santander, Spain: ACM, 2004, pp. 290–296. DOI: `10.1145/1005285.1005327` (cit. on p. 29).

[Hoe07] J. van der Hoeven. “New algorithms for relaxed multiplication”. In: *J. Symbolic Comput.* 42.8 (2007), pp. 792–802. DOI: `10.1016/j.jsc.2007.04.004` (cit. on p. 71).

[Hoe08] J. van der Hoeven. *Making fast multiplication of polynomials numerically stable*. Tech. rep. 2008-02. Orsay, France: Université Paris-Sud, 2008 (cit. on pp. 40, 41).

[Hoe+12] J. van der Hoeven, G. Lecerf, B. Mourrain, et al. “Mathemagix: the quest of modularity and efficiency for symbolic and certified numeric computation?” In: *ACM Communications in Computer Algebra* 45.3/4 (2012). `http://www.mathemagix.org/`, pp. 186–188 (cit. on pp. 52, 63).

[Hoe97] J. van der Hoeven. “Lazy Multiplication of Formal Power Series”. In: *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’97. Kihei, Maui, Hawaii, USA: ACM, 1997, pp. 17–20. DOI: `10.1145/258726.258738` (cit. on p. 71).

[Hoe14] J. van der Hoeven. “Faster Relaxed Multiplication”. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC ’14. Kobe, Japan: ACM, 2014, pp. 405–412. DOI: 10.1145/2608628.2608657 (cit. on pp. 71, 72).

[Hoe17] J. van der Hoeven. “Fast Chinese Remaindering in Practice”. In: *Mathematical Aspects of Computer and Information Sciences*. Ed. by J. Blömer, I. S. Kotsireas, T. Kutsia, and D. E. Simos. Cham: Springer International Publishing, 2017, pp. 95–106 (cit. on p. 50).

[HL13] J. van der Hoeven and G. Lecerf. “On the bit-complexity of sparse polynomial and series multiplication”. In: *Journal of Symbolic Computation* 50 (2013), pp. 227–254. DOI: https://doi.org/10.1016/j.jsc.2012.06.004 (cit. on p. 85).

[HL14] J. van der Hoeven and G. Lecerf. “Sparse polynomial interpolation in practice”. working paper or preprint. Apr. 2014 (cit. on p. 86).

[HLQ16] J. van der Hoeven, G. Lecerf, and G. Quintin. “Modular SIMD Arithmetic in Mathemagix”. In: *ACM Trans. Math. Softw.* 43.1 (Aug. 2016), 5:1–5:37. DOI: 10.1145/2876503 (cit. on pp. 55, 61, 64, 84).

[HNS19] S. G. Hyun, V. Neiger, and É. Schost. “Implementations of efficient univariate polynomial matrix algorithms and application to bivariate resultants”. working paper or preprint. Jan. 2019 (cit. on p. 84).

[IMH82] O. H. Ibarra, S. Moran, and R. Hui. “A generalization of the fast LUP matrix decomposition algorithm and applications”. In: *Journal of Algorithms* 3.1 (1982), pp. 45–56 (cit. on pp. 6, 61, 67).

[Int07] M. Intel. *Intel math kernel library*. 2007 (cit. on pp. 7, 61).

[Ish+17] M. Ishii, J. Detrey, P. Gaudry, A. Inomata, and K. Fujikawa. “Fast Modular Arithmetic on the Kalray MPPA-256 Processor for an Energy-Efficient Implementation of ECM”. In: *IEEE Transactions on Computers* 66.12 (Dec. 2017), pp. 2019–2030. DOI: 10.1109/TC.2017.2704082 (cit. on p. 81).

[Iza11] T. Izard. “Opérateurs arithmétiques parallèles pour la cryptographie asymétrique”. disponible à http://www.biu-montpellier.fr/florabium/jsp/nnt.jsp?nnt=2011MON20184. PhD thesis. Université Montpellier 2, Dec. 2011 (cit. on p. 47).

[Jea+16] C.-P. Jeannerod, V. Neiger, É. Schost, and G. Villard. “Fast computation of minimal interpolation bases in Popov form for arbitrary shifts”. In: *ISSAC’16*. Waterloo, ON, Canada: ACM, 2016, pp. 295–302 (cit. on pp. 68, 76, 77).

[Jea+17] C.-P. Jeannerod, V. Neiger, É. Schost, and G. Villard. “Computing minimal interpolation bases”. In: *J. Symbolic Comput.* 83 (2017), pp. 272–314 (cit. on pp. 22, 82).

[JV05] C.-P. Jeannerod and G. Villard. “Essentially optimal computation of the inverse of generic polynomial matrices”. In: *J. Complexity* 21.1 (2005), pp. 72–86 (cit. on p. 6).

[JNV19] C.-P. Jeannerod, V. Neiger, and G. Villard. “Fast computation of approximant bases in canonical form”. working paper or preprint. Apr. 2019 (cit. on p. 68).

[JPS13] C.-P. Jeannerod, C. Pernet, and A. Storjohann. “Rank-profile revealing Gaussian elimination and the CUP matrix decomposition”. In: *J. Symbolic Comput.* 56 (2013), pp. 46–68 (cit. on pp. 67, 81).

[Jel15] H. Jeljeli. “Accélérateurs logiciels et matériels pour l’algèbre linéaire creuse sur les corps finis”. https://tel.archives-ouvertes.fr/tel-01751696v2/document. PhD thesis. Université de Lorraine, 2015 (cit. on p. 14).

[KT08] M. E. Kaihara and N. Takagi. “Bipartire Modular Multiplication Method”. In: *IEEE Transactions on Computers* 57.2 (2008), pp. 157–164 (cit. on pp. 13, 45, 47).

[Kai80] T. Kailath. *Linear Systems*. Prentice-Hall, 1980 (cit. on p. 75).

[Kal94] E. Kaltofen. “Asymptotically Fast Solution of Toeplitz-like Singular Linear Systems”. In: *ISSAC’94*. ACM, 1994, pp. 297–304 (cit. on pp. 2, 5).

[KL03] E. Kaltofen and W.-S. Lee. “Early Termination in Sparse Interpolation Algorithms”. In: *J. Symbolic. Comput.* 36.3-4 (2003), pp. 365–400 (cit. on pp. 8, 71).

[Kal+12] E. Kaltofen, B. Li, Z. Yang, and L. Zhi. “Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients”. In: *J. Symbolic Comput.* 47.1 (2012), pp. 1–15 (cit. on pp. 9, 73).

[KL99] E. Kaltofen and A. Lobo. “Distributed Matrix-Free Solution of Large Sparse Linear Systems over Finite Fields”. In: *Algorithmica* 24.3 (July 1999), pp. 331–348. DOI: 10.1007/PL00008266 (cit. on p. 8).

[KNS11] E. Kaltofen, M. Nehring, and B. D. Saunders. “Quadratic-time Certificates in Linear Algebra”. In: *ISSAC’11*. ACM, 2011, pp. 171–176 (cit. on pp. 9, 73).

[KS91] E. Kaltofen and D. Saunders. “On Wiedemann’s method of solving sparse linear systems”. In: *AAECC-9*. Vol. 539. LNCS. Springer, 1991, pp. 29–38 (cit. on p. 8).

[KV05] E. Kaltofen and G. Villard. “On the complexity of computing determinants”. In: *Comput. Complexity* 13.3 (2005), pp. 91–130 (cit. on p. 70).

[Kal95] E. Kaltofen. “Analysis of Coppersmith’s block Wiedemann algorithm for the parallel solution of sparse linear systems”. In: *Mathematics of Computation* 64.210 (Apr. 1995), pp. 777–806 (cit. on pp. 2, 5).

[Kal10] E. L. Kaltofen. “Fifteen Years After DSC and WLSS2 What Parallel Computations I Do Today: Invited Lecture at PASCO 2010”. In: *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*. PASCO ’10. Grenoble, France: ACM, 2010, pp. 10–17. DOI: 10.1145/1837210.1837213 (cit. on p. 84).

[KO63] A. Karatsuba and Y. Ofman. “Multiplication of Multidigit Numbers on Automata”. In: *Soviet Physics-Doklady* 7 (1963), pp. 595–596 (cit. on pp. 5, 17, 25, 29, 43).

[KS93] T. Kimbrel and R. K. Sinha. “A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2(n)+O(1)$ random bits”. In: *Information Processing Letters* 45.2 (1993), pp. 107–110 (cit. on pp. 9, 20).

[Kle+17] T. Kleinjung, C. Diem, A. K. Lenstra, C. Priplata, and C. Stahlke. “Computation of a 768-Bit Prime Field Discrete Logarithm”. In: *Eurocrypt 2017*. Springer International Publishing, 2017, pp. 185–201 (cit. on p. 10).

[Kle+10] T. Kleinjung, K. Aoki, J. Franke, et al. “Factorization of a 768-Bit RSA Modulus”. In: *Advances in Cryptology – CRYPTO 2010*. 2010, pp. 333–350 (cit. on p. 84).

[Kle+12] T. Kleinjung, J. W. Bos, A. K. Lenstra, et al. “A heterogeneous computing environment to solve the 768-bit RSA challenge”. In: *Cluster Computing* 15.1 (2012), pp. 53–68. DOI: 10.1007/s10586-010-0149-0 (cit. on pp. 2, 10, 24, 73).

[Knu70] D. E. Knuth. “The analysis of algorithms”. In: *Congrès int. Math., Nice, France*. Vol. 3. 1970, pp. 269–274 (cit. on pp. 5, 43, 69).

[Knu97] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Third. Reading, MA: Addison-Wesley, 1997 (cit. on p. 14).

[KAK96] Ç. K. Koç, T. Acar, and B. S. Kaliski Jr. “Analyzing and Comparing Montgomery Multiplication Algorithms”. In: *IEEE Micro* 16.3 (June 1996), pp. 26–33. DOI: 10.1109/40.502403 (cit. on p. 44).

[LNZ17] G. Labahn, V. Neiger, and W. Zhou. “Fast, deterministic computation of the Hermite normal form and determinant of a polynomial matrix”. In: *J. Complexity* (in press) (2017) (cit. on pp. 5, 6).

[Law+79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. “Basic Linear Algebra Subprograms for Fortran Usage”. In: *ACM Trans. Math. Softw.* 5.3 (Sept. 1979), pp. 308–323. DOI: 10.1145/355841.355847 (cit. on p. 7).

[Le 14] F. Le Gall. “Powers of Tensors and Fast Matrix Multiplication”. In: *ISSAC’14*. Kobe, Japan: ACM, 2014, pp. 296–303 (cit. on pp. 5, 59).

[Leb12] R. Lebreton. “Contribution to relaxed algorithms and polynomial system solving”. PhD thesis. École Polytechnique, 2012 (cit. on pp. 8, 71, 81).

[Leb15] R. Lebreton. “Relaxed Hensel lifting of triangular sets”. In: *Journal of Symbolic Computation* 68 (May 2015), pp. 230–258. DOI: 10.1016/j.jsc.2014.09.012 (cit. on p. 71).

[LS16] R. Lebreton and É. Schost. “A simple and fast online power series multiplication and its analysis”. In: *Journal of Symbolic Computation* 72 (2016), pp. 231–251. DOI: https://doi.org/10.1016/j.jsc.2015.03.001 (cit. on p. 71).

[LH93] A. K. Lenstra and J. Hendrik W. Lenstra, eds. *The development of the number field sieve*. Vol. 1554. Lecture Notes in Mathematics. Berlin: Springer-Verlag, 1993 (cit. on p. 1).

[LMT93] R. Lidl, G. Mullen, and G. Turnwald. *Dickson polynomials*. Pitman monographs and surveys in pure and applied mathematics. Longman Scientific & Technical, 1993 (cit. on p. 38).

[Lin+16] A. Lincoln, V. Vassilevska Williams, J. R. Wang, and R. R. Williams. "Deterministic Time-Space Trade-Offs for k-SUM". In: *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Ed. by I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi. Vol. 55. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 58:1–58:14. DOI: 10.4230/LIPIcs.ICALP.2016.58 (cit. on p. 36).

[Luc+18] D. Lucas, V. Neiger, C. Pernet, D. S. Roche, and J. Rosenkilde. "Interactive Certificates for Polynomial Matrices with Sub-Linear Communication". In: *CoRR* abs/1807.01272 (2018) (cit. on pp. 73, 83).

[LPR10] V. Lyubashevsky, C. Peikert, and O. Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by H. Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–23 (cit. on p. 85).

[LPR13] V. Lyubashevsky, C. Peikert, and O. Regev. "A Toolkit for Ring-LWE Cryptography". In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by T. Johansson and P. Q. Nguyen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 35–54 (cit. on p. 85).

[MH02] J. C. Mason and D. C. Handscomb. *Chebyshev polynomials*. Boca Raton, FL: Chapman and Hall/CRC, 2002 (cit. on p. 37).

[Mas69] J. L. Massey. "Shift-register synthesis and BCH decoding". In: *IEEE Trans. Inf. Theory* 15.1 (Jan. 1969), pp. 122–127 (cit. on p. 3).

[MD07] P. K. Mishra and V. Dimitrov. "Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication Using Multibase Number Representation". In: *Information Security*. Ed. by J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 390–406 (cit. on p. 14).

[MB72] R. Moenck and A. Borodin. "Fast modular transforms via division". In: *13th Annual Symposium on Switching and Automata Theory (swat 1972)*. Oct. 1972, pp. 90–96. DOI: 10.1109/SWAT.1972.5 (cit. on p. 7).

[Mon85] P. L. Montgomery. "Modular Multiplication Without Trial Division". In: *Mathematics of Computation* 44.170 (Apr. 1985), pp. 519–521 (cit. on pp. 12, 13, 43, 44).

[MS90] P. L. Montgomery and R. D. Silverman. "An FFT Extension to the P - 1 Factoring Algorithm". In: *Mathematics of Computation - Math. Comput.* 54 (Apr. 1990), pp. 839–854. DOI: 10.1090/S0025-5718-1990-1011444-3 (cit. on p. 66).

[Mul00] T. Mulders. “On Short Multiplications and Divisions”. In: *Applicable Algebra in Engineering, Communication and Computing* 11.1 (2000), pp. 69–88. DOI: 10.1007/s002000000037 (cit. on pp. 17, 18).

[Neg16] C. Negre. “Multiplication in Finite Fields and Elliptic Curves”. Habilitation à diriger des recherches. Université de Montpellier, June 2016 (cit. on p. 44).

[Nei16a] V. Neiger. “Bases of relations in one or several variables: fast algorithms and applications”. PhD thesis. École Normale Supérieure de Lyon, Nov. 2016 (cit. on pp. 22, 68, 82).

[Nei16b] V. Neiger. “Fast Computation of Shifted Popov Forms of Polynomial Matrices via Systems of Modular Polynomial Equations”. In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC ’16. Waterloo, ON, Canada: ACM, 2016, pp. 365–372. DOI: 10.1145/2930889.2930936 (cit. on p. 6).

[NRS18] V. Neiger, J. Rosenkilde, and G. Solomatov. “Computing Popov and Hermite Forms of Rectangular Polynomial Matrices”. In: *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*. ISSAC ’18. New York, NY, USA: ACM, 2018, pp. 295–302. DOI: 10.1145/3208976.3208988 (cit. on p. 6).

[Per14] C. Pernet. “High Performance and Reliable Algebraic Computing”. Habilitation à diriger des recherches. Université Joseph Fourier, Grenoble 1, Nov. 2014 (cit. on pp. 5, 61, 62, 83).

[PS07] C. Pernet and A. Storjohann. “Faster Algorithms for the Characteristic Polynomial”. In: *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’07. Waterloo, Ontario, Canada: ACM, 2007, pp. 307–314. DOI: 10.1145/1277548.1277590 (cit. on p. 61).

[Pol71] J. M. Pollard. “The fast Fourier transform in a finite field”. In: *Mathematics of Computation* 25.114 (1971), pp. 365–374. DOI: 10.1090/S0025-5718-1971-0301966-0 (cit. on pp. 55, 63).

[PST98] D. Potts, G. Steidl, and M. Tasche. “Fast algorithms for discrete polynomial transforms”. In: *Mathematics of Computation* 67 (Oct. 1998), pp. 1577–1590 (cit. on p. 37).

[Reg05] O. Regev. “On Lattices, Learning with Errors, Random Linear Codes, and Cryptography”. In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. STOC ’05. Baltimore, MD, USA: ACM, 2005, pp. 84–93. DOI: 10.1145/1060590.1060603 (cit. on p. 85).

[RSA78] R. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public Key Cryptosystems”. In: *Communications of the ACM* 21.2 (Feb. 1978), pp. 120–126 (cit. on pp. 1, 43).

[Roc09] D. S. Roche. “Space- and Time-efficient Polynomial Multiplication”. In: *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’09. Seoul, Republic of Korea: ACM, 2009, pp. 295–302. DOI: 10.1145/1576702.1576743 (cit. on pp. 10, 24, 25, 29, 80, 81).

[Roc18] D. S. Roche. “What Can (and Can’T) We Do with Sparse Polynomials?” In: *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*. ISSAC ’18. New York, NY, USA: ACM, 2018, pp. 25–30. DOI: 10.1145/3208976.3209027 (cit. on p. 85).

[Sak+11] K. Sakiyama, M. Knezevic, J. Fan, B. Preneel, and I. Verbauwhede. “Tripartite modular multiplication”. In: *Integration, the VLSI Journal* (2011). In Press, Corrected Proof. DOI: 10.1016/j.vlsi.2011.03.008 (cit. on pp. 45, 47).

[SSV04] B. D. Saunders, A. Storjohann, and G. Villard. “Matrix Rank Certification”. In: *The Electronic Journal of Linear Algebra* 11 (Feb. 2004), pp. 16–23. DOI: 10.13001/1081-3810.1118 (cit. on p. 8).

[Sch71] A. Schönhage. “Schnelle Berechnung von Kettenbruchentwicklungen”. In: *Acta Inf.* 1 (June 1971), pp. 139–144. DOI: 10.1007/BF00289520 (cit. on pp. 5, 43, 69).

[Sch77] A. Schönhage. “Fast multiplication of polynomials over fields of characteristic 2”. In: *Acta Inform.* 7.4 (1977), pp. 395–398 (cit. on p. 5).

[SS71] A. Schönhage and V. Strassen. “Schnelle Multiplikation grosser Zahlen”. In: *Computing* 7 (1971), pp. 281–292 (cit. on pp. 5, 17, 43, 81).

[Sch80] J. T. Schwartz. “Fast Probabilistic Algorithms for Verification of Polynomial Identities”. In: *J. ACM* 27.4 (Oct. 1980), pp. 701–717. DOI: 10.1145/322217.322225 (cit. on pp. 20, 21).

[SK89] A. P. Shenoy and R. Kumaresan. “Fast base extension using a redundant modulus in RNS”. In: *IEEE Transactions on Computers* 38.2 (Feb. 1989), pp. 292–297. DOI: 10.1109/12.16508 (cit. on p. 66).

[SP04] N. P. Smart and D. Page. “Parallel Cryptographic Arithmetic Using a Redundant Montgomery Representation”. In: *IEEE Transactions on Computers* 53 (Nov. 2004), pp. 1474–1482. DOI: 10.1109/TC.2004.100 (cit. on p. 47).

[Sou99] C. Soulé. “Perfects forms and the Vandiver conjecture”. In: *J. reine angew. Math.* 517 (1999), pp. 209–221 (cit. on p. 8).

[Sto00] A. Storjohann. “Algorithms for Matrix Canonical Forms”. PhD thesis. Swiss Federal Institute of Technology – ETH, 2000 (cit. on p. 67).

[Sto03] A. Storjohann. “High-order lifting and integrality certification”. In: *J. Symbolic Comput.* 36.3-4 (2003), pp. 613–648 (cit. on pp. 5, 6, 59).

[Sto06] A. Storjohann. “Notes on computing minimal approximant bases”. In: *Challenges in Symbolic Computation Software*. Dagstuhl Seminar Proceedings. 2006 (cit. on p. 68).

[SV05] A. Storjohann and G. Villard. “Computing the Rank and a Small Nullspace Basis of a Polynomial Matrix”. In: *ISSAC’05*. Beijing, China: ACM, 2005, pp. 309–316 (cit. on p. 6).

[Sto05] A. Storjohann. “The Shifted Number System for Fast Linear Algebra on Integer Matrices”. In: *J. Complex.* 21.4 (Aug. 2005), pp. 609–650. DOI: 10.1016/j.jco.2005.04.002 (cit. on p. 5).

[Str69] V. Strassen. “Gaussian elimination is not optimal”. English. In: *Numerische Mathematik* 13.4 (1969), pp. 354–356. DOI: 10.1007/BF02165411 (cit. on pp. 5, 7, 51, 59–61).

[ST67] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and Its Applications to Computer Technology*. New York: McGraw-Hill Book Company, 1967 (cit. on p. 65).

[Tha13] J. Thaler. “Time-Optimal Interactive Proofs for Circuit Evaluation”. In: *Advances in Cryptology – CRYPTO 2013*. Ed. by R. Canetti and J. A. Garay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 71–89 (cit. on p. 9).

[Tho02a] E. Thomé. “Subquadratic Computation of Vector Generating Polynomials and Improvement of the Block Wiedemann Algorithm”. In: *J. Symbolic Comput.* 33.5 (2002), pp. 757–775 (cit. on pp. 2, 5).

[Tho02b] E. Thomé. “Karatsuba multiplication with temporary space of size $\leq$ n”. online. 2002 (cit. on pp. 10, 24, 25, 29).

[Too63] A. L. Toom. “The complexity of a scheme of functional elements realizing the multiplication of integers”. In: *Soviet Math* 3 (1963), pp. 714–716 (cit. on pp. 17, 26, 29, 43).

[Tra15] C. Tran. “Symbolic computing with the basis of Chebyshev’s monic polynomials”. Theses. Université Pierre et Marie Curie - Paris VI, Oct. 2015 (cit. on p. 38).

[Tur06] W. J. Turner. “A Block Wiedemann Rank Algorithm”. In: *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’06. Genoa, Italy: ACM, 2006, pp. 332–339. DOI: 10.1145/1145768.1145822 (cit. on pp. 8, 70).

[VB92] M. Van Barel and A. Bultheel. “A general module theoretic framework for vector M-Padé and matrix rational interpolation”. In: *Numer. Algorithms* 3 (1992), pp. 451–462 (cit. on pp. 68, 74, 76, 82).

[Vas18] V. Vassilevska Williams. “On some fine-grained questions in algorithms and complexity”. In: *Proceedings ICM*. Vol. 3. 2018, pp. 3431–3472 (cit. on p. 36).

[Vil97a] G. Villard. “Further Analysis of Coppersmith’s Block Wiedemann Algorithm for the Solution of Sparse Linear Systems (Extended Abstract)”. In: *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’97. Kihei, Maui, Hawaii, USA: ACM, 1997, pp. 32–39. DOI: 10.1145/258726.258742 (cit. on p. 2).

[Vil97b] G. Villard. *A study of Coppersmith’s block Wiedemann algorithm using matrix polynomials*. Tech. rep. 975–IM. LMC/IMAG, Apr. 1997 (cit. on pp. 1, 2, 5).

[Vil03] G. Villard. “Algorithmique en algèbre linéaire exacte”. Habilitation à diriger des recherches. Université Claude Bernard Lyon 1, 2003 (cit. on p. 5).

[VDY05] R. Vuduc, J. W. Demmel, and K. A. Yelick. “OSKI: A library of automatically tuned sparse matrix kernels”. In: *Journal of Physics: Conference Series* 16 (Jan. 2005), pp. 521–530. DOI: 10.1088/1742-6596/16/1/071 (cit. on p. 14).

[WPD01] R. C. Whaley, A. Petitet, and J. J. Dongarra. “Automated empirical optimizations of software and the ATLAS project”. In: *Parallel Computing* 27.1–2 (Jan. 2001). http://www.netlib.org/utk/people/JackDongarra/PAPERS/atlas_pub.pdf, pp. 3–35 (cit. on p. 61).

[Wie86] D. H. Wiedemann. “Solving Sparse Linear Equations Over Finite Fields”. In: *IEEE Transactions on Information Theory* 32.1 (Jan. 1986), pp. 54–62 (cit. on p. 1).

[Win71] S. Winograd. “On multiplication of 2×2 matrices”. In: *Linear Algebra and its Applications* 4.4 (1971), pp. 381–388. DOI: https://doi.org/10.1016/0024-3795(71)90009-7 (cit. on p. 60).

[Zho12] W. Zhou. “Fast Order Basis and Kernel Basis Computation and Related Problems”. PhD thesis. University of Waterloo, 2012 (cit. on pp. 5, 22, 69).

[ZL12] W. Zhou and G. Labahn. “Efficient Algorithms for Order Basis Computation”. In: *J. Symbolic Comput.* 47.7 (2012), pp. 793–819 (cit. on pp. 68, 76).

[ZLS12] W. Zhou, G. Labahn, and A. Storjohann. “Computing Minimal Nullspace Bases”. In: *ISSAC’12*. Grenoble, France: ACM, 2012, pp. 366–373 (cit. on pp. 6, 22).

[ZLS15] W. Zhou, G. Labahn, and A. Storjohann. “A deterministic algorithm for inverting a polynomial matrix”. In: *J. Complexity* 31.2 (2015), pp. 162–173 (cit. on pp. 5, 6).

[Zip79] R. Zippel. “Probabilistic Algorithms for Sparse Polynomials”. In: *Symbolic and Algebraic Computation. In: Lecture Notes in Comput. Sci., vol. 72*. Springer-Verlag, 1979, pp. 216–226 (cit. on pp. 20, 21).