Optimization Techniques for Small Matrix Multiplication

Charles-Éric Drevet
Ancien élève, École polytechnique, Palaiseau, France
drevet@m4x.org
Md. Nazrul Islam, Éric Schost
Department of Computer Science and ORCCA
The University of Western Ontario, London, ON
mislam63@uwo.ca, eschost@uwo.ca

May 14, 2010

Abstract

The complexity of matrix multiplication has attracted a lot of attention in the last forty years. In this paper, instead of considering asymptotic aspects of this problem, we are interested in reducing the cost of multiplication for matrices of small size, say up to 30. Following previous work in a similar vein by Probert & Fischer, Smith, and Mezzarobba, we base our approach on previous algorithms for small matrices, due to Strassen, Winograd, Pan, Laderman, ... and show how to exploit these standard algorithms in an improved way. We illustrate the use of our results by generating multiplication code over various rings, such as integers, polynomials, differential operators or linear recurrence operators.

Keywords: matrix multiplication, small matrix, complexity.

1 Introduction

Understanding the complexity of matrix multiplication remains an outstanding problem. Work of Strassen [32, 33, 34], Pan [22, 23, 25, 24], Schönhage [28], among many others, culminated with Coppersmith and Winograd's algorithm [11] of cost $O(n^{2.37})$ for multiplication in size $(n \times n \times n)$ – in what follows, a product *in size* $(m \times n \times p)$ means the product of a matrix of size $(m \times n)$ by a matrix of size $(n \times p)$. However, with a few exceptions, almost all algorithms of cost lower than $O(n^3)$ are impractical for reasonably sized matrices. As a result, Strassen's original algorithm, its variant by Winograd [39] and, less widely known,

Pan's trilinear aggregating techniques [25, 18, 16], remain essentially the only non-trivial algorithms implemented.

In many situations (such as the multiplication of large matrices with machine float entries), performance improvements result more from optimizing data access than reducing operation count [38]. However, there are situations, usually involving small matrices with large entries, where multiplication in the base ring remains the bottleneck: this is the case for multiprecision integers, high-degree polynomials, etc. Such questions arise for instance in the following contexts:

- *Padé-Hermite approximation.* Given power series $(f_1, \ldots, f_n)$, and degree bounds $(d_1, \ldots, d_n)$, a Padé-Hermite approximant of $(f_1, \ldots, f_n)$ is a vector of polynomials $(a_1, \ldots, a_n)$, with $\deg(a_i) < d_i$, such that $a_1 f_1 + \cdots + a_n f_n$ has a large valuation (usually about $d_1 + \cdots + d_n$). Beckermann and Labahn's divide-and-conquer algorithm for Padé-Hermite approximation [2] involves polynomial matrix multiplication in size $(n \times n \times n)$. This generalizes the products in size $(2 \times 2 \times 2)$ used in the half-gcd algorithm or the Padé approximant algorithm of [8]; often, n is small (say, a few dozens).

- *Holonomic function evaluation.* A function $f(x)$ is holonomic if it satisfies a linear differential equation with polynomial coefficients, or equivalently if the coefficients f_n of its power series expansion satisfy a linear recurrence with polynomial coefficients. The value of f at a given rational point can be approximated at high precision using this recurrence, using binary splitting techniques [10]. This involves matrix multiplication, in a size which is the order of the recurrence, and with multiprecision integer entries.

- *Lifting techniques for triangular sets.* Triangular representations are a versatile data structure used to solve systems of polynomial equations. A useful tool for this data structure is the lifting algorithm of [29], which enables one to start from a representation known modulo a prime p and deduce "better" representations with coefficients known modulo powers of p (and eventually over $\mathbb{Q}$). One of the main operations in this algorithm is the product of matrices whose entries are multivariate polynomials. Their size is equal to the number of variables in the system we want to solve (so it usually ranges up to 10 or 15), but the entries' degrees can reach hundreds or thousands.

For such situations, two main directions can be considered: using modular (Chinese Remaindering) methods, when possible, or reducing the number of base ring multiplications. This paper follows the second route. Our goal is to tabulate the best possible number of multiplications for small matrix products.

Previous work. Most results on matrix multiplication cited before were concerned with asymptotic estimates. As it turns out, the resulting algorithms are usually more costly than the naive one in size less than billions: techniques such as ε-algorithms induce extra polylogarithmic factors, that do not affect the exponent but are harmful in realistic sizes.

However, there also exist a handful of simple techniques, that may not be competitive asymptotically, but that are useful in small sizes. Table 1 reviews algorithms dedicated to

matrices of fixed small size; the second column indicates whether the algorithm assumes commutativity of the base ring or not. We indicate only the "number of multiplications"; see below for a precise definition of our model of computation.

In size $(2 \times 2 \times 2)$, Winograd's algorithm differs from Strassen's only by the additions it performs: we mention both as, surprisingly, they do not yield the same results when applied recursively for odd-sized matrices. In the table, we only give the best known results to date, but we should also mention previous work by Schachtel [27] for the $(5 \times 5 \times 5)$ case, with 103 multiplications, as well as the families of algorithms by Johnson and McLoughlin for the $(3 \times 3 \times 3)$ case, with 23 multiplications.

dimension	commutative	author	base ring multiplications
$(2 \times 2 \times 2)$	no	Strassen [32], Winograd [39]	7
$(3 \times 3 \times 3)$	no	Laderman [17]	23
$(5 \times 5 \times 5)$	no	Makarov [20]	100
$(3 \times 3 \times 3)$	yes	Makarov [19]	22

Table 1: Upper bounds on the number of multiplications, fixed small size

Table 2 reviews some families of algorithms. After the naive algorithm and Sykora's algorithm, we mention dimensions of the form $(a \times 2 \times c)$ for Hopcroft and Kerr's algorithm; duality techniques (Hopcroft and Musinski [13]) show that the number of multiplications is the same for $(2 \times a \times c)$ and $(a \times c \times 2)$.

The fourth block in Table 2 refers to techniques based on the simultaneous computation of two products of sizes $(a \times b \times c)$ and $(b \times c \times a)$, due to Pan; these techniques can be used to perform single products as well. Along the same lines, *trilinear aggregating* techniques (fifth block) enable one to perform three products at once, and can be extended to do single products. Pan's original approach, and some variants were put to practice by Laderman, Pan and Sha [18] and Kaporin [16].

Finally, we mention families of algorithms that require commutativity of the base ring. In [40], Winograd introduced an algorithm that allows one to reduce the number of multiplications for $(a \times b \times c)$ products almost by half. Waksman [37] subsequently improved it, to give the result in the last entry of our table.

Building tables. All the algorithms that do not rely on commutativity of the entries can be used recursively. For sizes that are not pure powers of the size of the base case, one usually uses techniques such as *peeling* (removing rows / columns) or *padding* (adding zero rows / columns): this does not affect the exponent in the asymptotic scale, but caution must be taken when applying these techniques in small sizes.

For instance, if the base ring is non-commutative, matrices of size (6×6) can be multiplied using $23 \times 7 = 161$ base ring multiplications by combining Strassen's and Laderman's algorithms, and this is the best method to date. To multiply matrices of size (7×7), however, it is less obvious what approach should be employed. Seeing the variety of available methods, the question we consider is thus how to combine them in an optimal way: in many

dimension	commutative	author	base ring multiplications
$(a \times b \times c)$	no	naive	abc
$(a \times a \times a)$	no	Sykora [35]	$a^3 - (a-1)^2$
$(a \times 2 \times c)$	no	Hopcroft-Kerr [14]	$(3ac + \max(a,c))/2$
$(a \times b \times c)$ $+(b \times c \times a)$	no	Pan [25]	$abc + ab + bc + ac$
$(a \times a \times a)$, a even	no	Pan [22]	$(a^3 + 4.5a^2 - 3a)/2$
$(a \times b \times c)$ $+(b \times c \times a)$ $+(c \times a \times b)$, a, b, c even	no	Pan [25]	$abc + 2(ab + ac + bc) + 4(a + b + c) + 15$
$(a \times a \times a)$, a even	no	Pan [25, 23]	$\min \begin{cases} (a^3 + 12a^2 + 17a)/3, \\ (a^3 + 11.25a^2 + 32a + 27)/3 \end{cases}$
$(a \times b \times c)$, b even	yes	Waksman [37]	$b(ac + a + c - 1)/2$
$(a \times b \times c)$, b odd	yes	Waksman [37]	$(b-1)(ac + a + c - 1)/2 + ac$

Table 2: Upper bounds on the number of multiplications

cases, the answers are not obvious. To add to our motivation, we note that little is known as to the actual complexity of small matrix multiplications. Except for size $(2 \times 2 \times 2)$ and $(2 \times 3 \times 2)$, none of the best known lower bounds [3, 4] matches the upper ones.

We are not the first to be interested in the small cases of matrix multiplication. Probert and Fischer [26] already tabulated upper bounds for square dimensions up to $(40 \times 40 \times 40)$. Pan [25, Sect. 31] gave such a table as well, for some square dimensions up to $(52 \times 52 \times 52)$; Smith [31] produced a similar table, for all rectangular dimensions up to $(28 \times 28 \times 28)$, which, most likely, he obtained using a computer search. Motivated by applications to holonomic function evaluation, Mezzarobba [21] tabulated the commutative case for square sizes up to $(28 \times 28 \times 28)$.

Our contribution. The references we examined show that even for small sizes, the complexity of matrix multiplication remains mysterious. Our goal in this paper is thus to revisit the case of small matrices, and tabulate improved number of multiplications: due to the large amount of prior work, obtaining an improvement of even a few dozens is never immediate.

We will do this mainly by finding better combinations of former techniques. We focus on the recursive approach, when the target dimensions are not multiples of the dimensions of the base case. The key contribution of this paper is in Section 2: for such situations, we show how to avoid useless multiplications, by taking into account the "sparseness" of the algorithm we wish to apply recursively. Besides, we also present a slight improvement of

trilinear aggregating techniques, that achieves a lower operation count than former versions in the sizes we consider.

Since these optimizations can be performed in an automated way, we used a computer search for square sizes up to (30×30). We obtained lower multiplication counts, for the non-commutative and commutative cases, for many of these sizes; the complete data is available at `http://www.csd.uwo.ca/~mislam63/`. As a proof-of-concept, we developed a code generator that automatically creates implementations of the algorithms we find as we complete the search; however, not much attention was paid to optimize the code thus produced.

Our work should not be confused with another way of using computers to find matrix multiplication algorithms. Indeed, using (usually numerical) optimization techniques, it is also possible to look for an algorithm for a fixed product size, that uses a prescribed number of base field multiplications, by solving polynomial equations: this idea originated in [7], and has since then been reused in [15] or [31]. We do not pursue this approach here.

Notation, computational model. Informally, we wish to count only "essential" multiplications (that do not involve constants) in our algorithms. To formalize this, we use the following standard computational model (see e.g. [9, Ch. 14]). Below, all indices in sums or sequences start at 1.

We consider matrices with entries in a ring R; in general, we do not assume that R is commutative, so the algorithms we consider are *bilinear* algorithms. Given integers a, b, c, a bilinear algorithm for matrix multiplication in size $(a \times b \times c)$ consists of three sequences $(U_\ell)_{\ell \leq L}$, $(V_\ell)_{\ell \leq L}$ and $(W_\ell)_{\ell \leq L}$, with

$$U_\ell = (u_{\ell,i,j})_{i \leq a, j \leq b}, \quad V_\ell = (v_{\ell,i,j})_{i \leq b, j \leq c}, \quad W_\ell = (w_{\ell,i,j})_{i \leq a, j \leq c},$$

such that the following holds. Let $\mathbf{M} = (\mathbf{m}_{i,j})_{i \leq a, j \leq b}$ and $\mathbf{N} = (\mathbf{n}_{i,j})_{i \leq b, j \leq c}$ be matrices whose entries are indeterminates $\mathbf{m}_{i,j}$ and $\mathbf{n}_{i,j}$ over R; for $\ell \leq L$, define

$$\alpha_\ell = \sum_{i \leq a, j \leq b} u_{\ell,i,j} \mathbf{m}_{i,j}, \quad \beta_\ell = \sum_{i \leq b, j \leq c} v_{\ell,i,j} \mathbf{n}_{i,j}, \quad \gamma_\ell = \alpha_\ell \beta_\ell$$

and define finally $p_{i,j} = \sum_{\ell \leq L} w_{\ell,i,j} \gamma_\ell$ for $i \leq a$ and $j \leq c$. Then, we ask that

$$p_{i,j} = \sum_{k \leq b} \mathbf{m}_{i,k} \mathbf{n}_{k,j}$$

for all i, j. This scheme can then be applied to compute *any* matrix product $P = MN$ in size $(a \times b \times c)$: after performing L linear combinations of the entries of M and the entries of N, and multiplying them pairwise, we obtain the entries of the product P by a last series

of linear combinations. For example, Strassen's algorithm can be represented as

$$
\begin{aligned}
(U_\ell)_{\ell\le 7} &= \left(\begin{bmatrix}0&0\\0&1\end{bmatrix}, \begin{bmatrix}1&0\\0&0\end{bmatrix}, \begin{bmatrix}0&0\\1&1\end{bmatrix}, \begin{bmatrix}1&1\\0&0\end{bmatrix}, \begin{bmatrix}-1&0\\1&0\end{bmatrix}, \begin{bmatrix}0&1\\0&-1\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix}\right),\\
(V_\ell)_{\ell\le 7} &= \left(\begin{bmatrix}-1&0\\1&0\end{bmatrix}, \begin{bmatrix}0&1\\0&-1\end{bmatrix}, \begin{bmatrix}1&0\\0&0\end{bmatrix}, \begin{bmatrix}0&0\\0&1\end{bmatrix}, \begin{bmatrix}1&1\\0&0\end{bmatrix}, \begin{bmatrix}0&0\\1&1\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix}\right),\\
(W_\ell)_{\ell\le 7} &= \left(\begin{bmatrix}1&0\\1&0\end{bmatrix}, \begin{bmatrix}0&1\\0&1\end{bmatrix}, \begin{bmatrix}0&0\\1&-1\end{bmatrix}, \begin{bmatrix}-1&1\\0&0\end{bmatrix}, \begin{bmatrix}0&0\\0&1\end{bmatrix}, \begin{bmatrix}1&0\\0&0\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix}\right).
\end{aligned}
$$

From the first three matrices, we get the information that we should compute $\gamma_1 = m_{2,2}(n_{2,1} - n_{1,1})$ and that γ_1 will be used in the result matrix for entries $p_{1,1}$ and $p_{2,1}$, both times with coefficient 1. Note that this model does not specify how to perform the linear combinations (e.g., what common subexpressions can be shared).

If we assume that R is commutative, we can allow algorithms that exploit commutativity: in this case, U_ℓ and V_ℓ have the form

$$U_\ell = \big((u_{\ell,i,j})_{i\le a, j\le b}, (u'_{\ell,i,j})_{i\le b, j\le c}\big), \quad V_\ell = \big((v_{\ell,i,j})_{i\le a, j\le b}, (v'_{\ell,i,j})_{i\le b, j\le c}\big),$$

and α_ℓ and β_ℓ are now defined by

$$\alpha_\ell = \sum_{i\le a, j\le b} u_{\ell,i,j}\mathbf{m}_{i,j} + \sum_{i\le a, j\le b} u'_{\ell,i,j}\mathbf{n}_{i,j}, \quad \beta_\ell = \sum_{i\le a, j\le b} v_{\ell,i,j}\mathbf{m}_{i,j} + \sum_{i\le a, j\le b} v'_{\ell,i,j}\mathbf{n}_{i,j}.$$

The rest of the definition is unchanged; such algorithms are called *quadratic* algorithms.

Of course, many algorithms can be described in a higher-level manner, without explicitly giving the coefficient sequences $(U_\ell)_{\ell\le L}$, $(V_\ell)_{\ell\le L}$ and $(W_\ell)_{\ell\le L}$: this is for instance the case for the trilinear aggregating techniques we will mention later on. However, we will rely on the actual data of $(U_\ell)_{\ell\le L}$, $(V_\ell)_{\ell\le L}$ and $(W_\ell)_{\ell\le L}$ to perform our searches.

Organization of the paper. Section 2 contains our main technical contribution: we show how to avoid useless multiplication when applying recursively a multiplication algorithm such as Strassen's or Winograd's. Section 3 explains how we used this result, among other techniques, to complete a computer search for improved multiplication algorithms for small square matrices; Section 4 gives a few experimental results obtained using code that was automatically generated. Finally, we give in appendix a slightly improved version of trilinear aggregating techniques.

Acknowledgements. We thank Marc Mezzarobba for useful discussions. We acknowledge the financial support of NSERC, MITACS and the Canada Research Chairs program.

2 Improved padding techniques

It is well-known that we can use multiplication algorithms in a recursive way through block matrix multiplication: this leads to divide-and-conquer techniques which are at the basis of all asymptotically fast matrix multiplication algorithms. In what follows, to highlight the operations we wish to perform, we will call *pattern* the base case algorithm that we wish to apply recursively; the *size* $(a \times b \times c)$ of the pattern is the size of the base case. Thus, we will consider patterns called `Strassen`, `Winograd`, `Laderman`, etc, that correspond to the algorithms in Table 1 which do not rely on commutativity of the base ring; they have respective sizes $(2 \times 2 \times 2)$, $(2 \times 2 \times 2)$, $(3 \times 3 \times 3)$, etc.

Such recursive techniques require adaptations when the target size is not a multiple of the pattern size: typically, one *pads* the input matrices using extra rows or columns of zeros, or one *peels* it from extra rows or columns. This is harmless as far as asymptotic estimates are concerned, since it only induces a constant factor overhead, but this constant factor is harmful for smaller size matrices.

In this section, we show how to control the cost incurred by padding techniques, by taking into account the sparsity of the pattern. We start by reviewing the formulas for the *exact* case, that is, when the target size is a multiple of the pattern size, so no padding is necessary. Then, we study the non-exact case on an example, using Strassen's pattern to multiply square matrices of size 3. The final subsection discusses the non-exact case in general.

2.1 The exact case

We start with the easiest situation: suppose we want to apply a pattern $(U_\ell)_{\ell \le L}$, $(V_\ell)_{\ell \le L}$, $(W_\ell)_{\ell \le L}$ of size $(a \times b \times c)$ to compute a product $P = MN$ of size $(m \times n \times p)$, and assume that we are in the *exact* case, that is, a divides m, b divides n and c divides p.

Here, no padding is necessary, and the subdivisions of M, N and P are straightforward: we subdivide M into blocks $M_{i,j}$ of size $(m/a \times n/b)$, N into blocks $N_{i,j}$ of size $(n/b \times p/c)$ and P into blocks $P_{i,j}$ of size $(m/a \times p/c)$:

$$M = \begin{bmatrix} M_{1,1} & \cdots & M_{1,b} \\ \vdots & & \vdots \\ M_{a,1} & \cdots & M_{a,b} \end{bmatrix}, \quad N = \begin{bmatrix} N_{1,1} & \cdots & N_{1,c} \\ \vdots & & \vdots \\ N_{b,1} & \cdots & N_{b,c} \end{bmatrix}, \quad P = \begin{bmatrix} P_{1,1} & \cdots & P_{1,c} \\ \vdots & & \vdots \\ P_{a,1} & \cdots & P_{a,c} \end{bmatrix}.$$

Then, the formulas used to obtain P are straightforward as well: for $\ell \le L$, we compute

$$\alpha_\ell = \sum_{i \le a, j \le b} u_{\ell,i,j} M_{i,j}, \quad \beta_\ell = \sum_{i \le a, j \le b} v_{\ell,i,j} N_{i,j}, \quad \gamma_\ell = \alpha_\ell \beta_\ell$$

and we obtain $P_{i,j} = \sum_{\ell \le L} w_{\ell,i,j} \gamma_\ell$. These simple formulas will be useful later on.

2.2 The non-exact case: a worked example

Most of our attention will be devoted to the *non-exact* case, where the dimensions of the product we want to compute are not multiples of the dimensions of the pattern. We start here with an example: we describe how to multiply two square matrices M, N of size (3×3), using Strassen's pattern.

The padding strategy consists in adding an extra row and column of zeros, to make the matrices (4×4). If we forget that the matrices hold many zeros, we obtain a cost of 49 multiplications. Explicitly, after padding the input matrices M, N, we obtain

$$\tilde{M} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \tilde{N} = \begin{bmatrix} n_{1,1} & n_{1,2} & n_{1,3} & 0 \\ n_{2,1} & n_{2,2} & n_{2,3} & 0 \\ n_{3,1} & n_{3,2} & n_{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

whose block decomposition is

$$\tilde{M}_{1,1} = \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix}, \quad \tilde{M}_{1,2} = \begin{bmatrix} m_{1,3} & 0 \\ m_{2,3} & 0 \end{bmatrix}, \quad \tilde{M}_{2,1} = \begin{bmatrix} m_{3,1} & m_{3,2} \\ 0 & 0 \end{bmatrix}, \quad \tilde{M}_{2,2} = \begin{bmatrix} m_{3,3} & 0 \\ 0 & 0 \end{bmatrix},$$

and

$$\tilde{N}_{1,1} = \begin{bmatrix} n_{1,1} & n_{1,2} \\ n_{2,1} & n_{2,2} \end{bmatrix}, \quad \tilde{N}_{1,2} = \begin{bmatrix} n_{1,3} & 0 \\ n_{2,3} & 0 \end{bmatrix}, \quad \tilde{N}_{2,1} = \begin{bmatrix} n_{3,1} & n_{3,2} \\ 0 & 0 \end{bmatrix}, \quad \tilde{N}_{2,2} = \begin{bmatrix} n_{3,3} & 0 \\ 0 & 0 \end{bmatrix}.$$

Recall that the product terms in the block version of Strassen's algorithm are, in this case

- $\gamma_1 = \tilde{M}_{2,2}(\tilde{N}_{2,1} - \tilde{N}_{1,1})$
- $\gamma_2 = \tilde{M}_{1,1}(\tilde{N}_{1,2} - \tilde{N}_{2,2})$
- $\gamma_3 = (\tilde{M}_{2,1} + \tilde{M}_{2,2})\tilde{N}_{1,1}$
- $\gamma_4 = (\tilde{M}_{1,1} + \tilde{M}_{1,2})\tilde{N}_{2,2}$
- $\gamma_5 = (\tilde{M}_{2,1} - \tilde{M}_{1,1})(\tilde{N}_{1,1} + \tilde{N}_{1,2})$
- $\gamma_6 = (\tilde{M}_{1,2} - \tilde{M}_{2,2})(\tilde{N}_{2,1} + \tilde{N}_{2,2})$
- $\gamma_7 = (\tilde{M}_{1,1} + \tilde{M}_{2,2})(\tilde{N}_{1,1} + \tilde{N}_{2,2})$.

The result matrix $\tilde{P} = \tilde{M}\tilde{N}$ has the form

$$\tilde{P} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & 0 \\ p_{2,1} & p_{2,2} & p_{2,3} & 0 \\ p_{3,1} & p_{3,2} & p_{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix};$$

its block decomposition is

$$\tilde{P}_{1,1} = \begin{bmatrix} p_{1,1} & p_{1,2} \\ p_{2,1} & p_{2,2} \end{bmatrix}, \quad \tilde{P}_{1,2} = \begin{bmatrix} p_{1,3} & 0 \\ p_{2,3} & 0 \end{bmatrix}, \quad \tilde{P}_{2,1} = \begin{bmatrix} p_{3,1} & p_{3,2} \\ 0 & 0 \end{bmatrix}, \quad \tilde{P}_{2,2} = \begin{bmatrix} p_{3,3} & 0 \\ 0 & 0 \end{bmatrix},$$

whose entries are given by

- $\tilde{P}_{1,1} = \gamma_1 + \gamma_6 + \gamma_7 - \gamma_4$
- $\tilde{P}_{1,2} = \gamma_2 + \gamma_4$
- $\tilde{P}_{2,1} = \gamma_1 + \gamma_3$
- $\tilde{P}_{2,2} = \gamma_2 - \gamma_3 + \gamma_5 + \gamma_7$.

Naively, each γ_i can be computed using 7 multiplications in size $(2 \times 2 \times 2)$, each of them done using Strassen's algorithm recursively. However, some may visibly be done in smaller size than $(2 \times 2 \times 2)$:

- Some improvements are obvious, as some γ_i are seen to contain zero rows or columns. Consider for instance $\gamma_1 = \tilde{M}_{2,2}(\tilde{N}_{2,1} - \tilde{N}_{1,1})$. Since $\tilde{M}_{2,2}$ has one row full of zeros, γ_1 does as well; thus, we can reduce the cost of computing γ_1 from 7 to 4 multiplications. Noticing the extra column of zeros in $\tilde{M}_{2,2}$ reduces the cost further to 2 multiplications.

- Less obviously, even if some γ_i has no zero row or column, we may not need all of it for the end result. Consider for instance $\gamma_5 = (\tilde{M}_{2,1} - \tilde{M}_{1,1})(\tilde{N}_{1,1} + \tilde{N}_{1,2})$. There is no zero row or column in any of the terms in this product. However, γ_5 is used only to compute $\tilde{P}_{2,2} = \gamma_2 - \gamma_3 + \gamma_5 + \gamma_7$, and we know that $\tilde{P}_{2,2}$ has only one non-zero term. Thus, we only need one term in γ_5: this reduces the cost from 7 to 2 multiplications.

Doing all these optimizations, we obtain an algorithm using 25 products instead our initial naive estimate 49 to perform multiplications in size $(3 \times 3 \times 3)$: this is not as good as Laderman's algorithm, but better than the naive algorithm.

We conclude this subsection by a discussion of some relevant previous work. In [12], D'Alberto and Nicolau discuss strategies for implementing Strassen's algorithm for odd-sized matrices, and explain how to reduce the sizes of some recursive calls. In particular, they already exhibit the reduction of the cost of computing γ_1 showed in our first example; however, they do not obtain the same cost as us for γ_5.

2.3 The non-exact case in general

The ideas underlying the former example are generalized in this subsection: we show how to exploit sparseness properties of a pattern to avoid useless computations.

First, we need a new notation, as we will have to resize matrices by increasing or reducing their dimensions. Given a matrix Z of size $(g \times h)$ and integers α, β, the matrix $Z' = \mathsf{R}(Z, \alpha, \beta)$ is the matrix of size $(\alpha \times \beta)$ defined as follows:

- If $\alpha \leq g$ then delete the rows in Z of indices greater than α; if $\alpha > g$ then pad $\alpha - g$ zero rows at the bottom of Z, to make the row dimension equal to α.
- If $\beta \leq h$ then delete the columns in Z of indices greater than β; if $\beta > h$ then pad $\beta - h$ rightmost zero columns to make the column dimension equal to β.

Given a problem size $(m \times n \times p)$ and a pattern $(U_\ell)_{\ell \leq L}$, $(V_\ell)_{\ell \leq L}$, $(W_\ell)_{\ell \leq L}$ of size $(a \times b \times c)$, we require compositions (that is, ordered partitions) $(m_1, \ldots, m_a)$ of m, $(n_1, \ldots, n_b)$ of n and $(p_1, \ldots, p_c)$ of p. If we are to compute a product $P = MN$ of size $(m \times n \times p)$, we use these compositions to determine the sizes of the submatrices in the matrices M, N and P, writing

$$M = \begin{bmatrix} M_{1,1} & \cdots & M_{1,b} \\ \vdots & & \vdots \\ M_{a,1} & \cdots & M_{a,b} \end{bmatrix}, \quad N = \begin{bmatrix} N_{1,1} & \cdots & N_{1,c} \\ \vdots & & \vdots \\ N_{b,1} & \cdots & N_{b,c} \end{bmatrix}, \quad P = \begin{bmatrix} P_{1,1} & \cdots & P_{1,c} \\ \vdots & & \vdots \\ P_{a,1} & \cdots & P_{a,c} \end{bmatrix},$$

where $M_{i,j}$ has size $(m_i \times n_j)$, $N_{i,j}$ has size $(n_i \times p_j)$ and $P_{i,j}$ has size $(m_i \times p_j)$.

Our goal here is to obtain integers $(\mu_\ell)_{\ell \leq L}$, $(\nu_\ell)_{\ell \leq L}$ and $(\pi_\ell)_{\ell \leq L}$, such that μ_ℓ, ν_ℓ and π_ℓ indicate in what size we perform the ℓth linear combination and the ℓth recursive product. Assuming these integers are known, our algorithm simply follows the one in the exact case, up to the management of the submatrices' sizes. For $\ell \leq L$, we compute

- $\alpha_\ell = \sum_{i \leq a, j \leq b} u_{\ell,i,j} \tilde{M}_{\ell,i,j}$, with $\tilde{M}_{\ell,i,j} = \mathsf{R}(M_{i,j}, \mu_\ell, \nu_\ell)$; note that α_ℓ has size $(\mu_\ell \times \nu_\ell)$
- $\beta_\ell = \sum_{i \leq b, j \leq c} v_{\ell,i,j} \tilde{N}_{\ell,i,j}$, with $\tilde{N}_{\ell,i,j} = \mathsf{R}(N_{i,j}, \nu_\ell, \pi_\ell)$; note that β_ℓ has size $(\nu_\ell \times \pi_\ell)$
- $\gamma_\ell = \alpha_\ell \beta_\ell$; note that γ_ℓ has size $(\mu_\ell \times \pi_\ell)$

Finally, for $i \leq a$ and $j \leq c$, we compute $\tilde{P}_{i,j} = \sum_{\ell \leq L} w_{\ell,i,j} \mathsf{R}(\gamma_\ell, m_i, p_j)$ and we set

$$\tilde{P} = \begin{bmatrix} \tilde{P}_{1,1} & \cdots & \tilde{P}_{1,c} \\ \vdots & & \vdots \\ \tilde{P}_{a,1} & \cdots & \tilde{P}_{a,c} \end{bmatrix}.$$

There is no guarantee that this algorithm produces the correct result in all cases: if $\mu_\ell, \nu_\ell, \pi_\ell$ are too small, the products γ_ℓ will not contain enough information. We now give a condition sufficient to ensure validity, while maintaining $\mu_\ell, \nu_\ell, \pi_\ell$ as small as possible. We need the following quantities, for $\ell \leq L$:

$$\begin{aligned}
R_{1,\ell} &= \max\{m_i \text{ for } i \leq a, j \leq b \text{ such that } u_{\ell,i,j} \neq 0\} && (1)\\
C_{1,\ell} &= \max\{n_j \text{ for } i \leq a, j \leq b \text{ such that } u_{\ell,i,j} \neq 0\} && (2)\\
R_{2,\ell} &= \max\{n_i \text{ for } i \leq b, j \leq c \text{ such that } v_{\ell,i,j} \neq 0\} && (3)\\
C_{2,\ell} &= \max\{p_j \text{ for } i \leq b, j \leq c \text{ such that } v_{\ell,i,j} \neq 0\} && (4)\\
R_{3,\ell} &= \max\{m_i \text{ for } i \leq a, j \leq c \text{ such that } w_{\ell,i,j} \neq 0\} && (5)\\
C_{3,\ell} &= \max\{p_j \text{ for } i \leq a, j \leq c \text{ such that } w_{\ell,i,j} \neq 0\}. && (6)
\end{aligned}$$

Remark that these quantities depend on the *sparseness* of the pattern: the more zeros in $(U_\ell)_{\ell\le L}$, $(V_\ell)_{\ell\le L}$, $(W_\ell)_{\ell\le L}$, the smaller they are. Remark also that given $(U_\ell)_{\ell\le L}$, $(V_\ell)_{\ell\le L}$, $(W_\ell)_{\ell\le L}$ and the compositions of m, n, p, these quantities can be easily computed: this will be instrumental to automatize the search process in the next section.

Proposition 1. *Suppose that the following conditions hold for all $\ell \le L$:*

- $\mu_\ell \ge \min(R_{1,\ell}, R_{3,\ell})$,
- $\nu_\ell \ge \min(C_{1,\ell}, R_{2,\ell})$,
- $\pi_\ell \ge \min(C_{2,\ell}, C_{3,\ell})$,

Then $\tilde{P} = MN$.

Proof. Let us first rewrite the definition of $\tilde{P}_{i,j}$ explicitly as

$$\tilde{P}_{i,j} \;=\; \sum_{\ell\le L} \sum_{i'\le a,\ j'\le b} \sum_{i''\le b,\ j''\le c} \tilde{T}_{\ell,i,j,i',j',i'',j''}$$

$$\text{with } \tilde{T}_{\ell,i,j,i',j',i'',j''} \;=\; u_{\ell,i',j'} v_{\ell,i'',j''} w_{\ell,i,j} \mathsf{R}(\tilde{M}_{\ell,i',j'} \tilde{N}_{\ell,i'',j''}, m_i, p_j).$$

Adding the obvious conditions $u_{\ell,i',j'} \neq 0$, $v_{\ell,i'',j''} \neq 0$ and $w_{\ell,i,j} \neq 0$ in the definitions of α_ℓ, β_ℓ and γ_ℓ gives

$$\tilde{P}_{i,j} \;=\; \sum_{\ell\le L,\ w_{\ell,i,j}\neq 0} \sum_{i'\le a,\ j'\le b,\ u_{\ell,i',j'}\neq 0} \sum_{i''\le b,\ j''\le c,\ v_{\ell,i'',j''}\neq 0} \tilde{T}_{\ell,i,j,i',j',i'',j''}. \tag{7}$$

On the other hand, let us also describe the classical way to multiply M, N by padding zeros. Let

$$m' = \max(m_1, \dots, m_a), \quad n' = \max(n_1, \dots, n_b), \quad p' = \max(p_1, \dots, p_c).$$

We pad columns and rows of zeros to the right-bottom of all submatrices $M_{i,j}$ and $N_{i,j}$. This gives blocks $M'_{i,j}$ and $N'_{i,j}$ of sizes $(m' \times n')$ and $(n' \times p')$ respectively; formally, we have $M'_{i,j} = \mathsf{R}(M_{i,j}, m', n')$ and $N'_{i,j} = \mathsf{R}(N_{i,j}, n', p')$. For these larger matrices, we are in the exact case of Subsection 2.1, so we can apply the recursive algorithm. For $\ell \le L$, we define

$$\alpha'_\ell = \sum_{i\le a, j\le b} u_{\ell,i,j} M'_{i,j}, \quad \beta'_\ell = \sum_{i\le b, j\le c} v_{\ell,i,j} N'_{i,j}, \quad \gamma'_\ell = \alpha'_\ell \beta'_\ell$$

and $P'_{i,j} = \sum_{\ell\le L} w_{\ell,i,j} \gamma'_\ell$. Thus, the blocks in the result $P = MN$ are given by $P_{i,j} = \mathsf{R}(P'_{i,j}, m_i, p_j)$. To make the connection with the previous formulas, we rewrite $P_{i,j}$ as

$$\begin{aligned} P_{i,j} &= \mathsf{R}(P'_{i,j}, m_i, p_j) \\ &= \mathsf{R}(\sum_{\ell\le L} w_{\ell,i,j} \gamma'_\ell, m_i, p_j) \\ &= \sum_{\ell\le L} w_{\ell,i,j} \mathsf{R}(\gamma'_\ell, m_i, p_j). \end{aligned}$$

As before, we add the conditions $u_{\ell,i',j'} \neq 0$, $v_{\ell,i'',j''} \neq 0$ and $w_{\ell,i,j} \neq 0$; this gives, after expansion

$$P_{i,j} \;=\; \sum_{\ell \le L,\ w_{\ell,i,j}\neq 0} \;\; \sum_{i' \le a,\ j' \le b,\ u_{\ell,i',j'}\neq 0} \;\; \sum_{i'' \le b,\ j'' \le c,\ v_{\ell,i'',j''}\neq 0} T_{\ell,i,j,i',j',i'',j''} \tag{8}$$

$$\text{with } \; T_{\ell,i,j,i',j',i'',j''} \;=\; u_{\ell,i',j'} v_{\ell,i'',j''} w_{\ell,i,j} \mathsf{R}(M'_{i',j'} N'_{i'',j''}, m_i, p_j).$$

We have to prove that under the assumptions of the proposition, the output $\widetilde{P}_{i,j}$ obtained in our algorithm agrees with the correct result $P_{i,j}$ we just obtained. Since the summation indices are the same in eqs. (7) and (8), it is enough to prove that for all terms that appear, we have

$$\mathsf{R}(\tilde{M}_{\ell,i',j'} \tilde{N}_{\ell,i'',j''}, m_i, p_j) \;=\; \mathsf{R}(M'_{i',j'} N'_{i'',j''}, m_i, p_j). \tag{9}$$

We first consider the right-hand side of (9). The definitions of the matrices $M'_{i',j'}$ and $N'_{i'',j''}$ give $\mathsf{R}(M'_{i',j'} N'_{i'',j''}, m_i, p_j) = \mathsf{R}(\mathsf{R}(M_{i',j'}, m', n')\mathsf{R}(N_{i'',j''}, n', p'), m_i, p_j)$. Let $n_{j',i''} = \min(n_{j'}, n_{i''})$. Since we have $m' \geq m_i$, $n' \geq n_{j',i''}$ and $p' \geq p_j$, by Lemma 3 (stated and proved below), we deduce

$$\mathsf{R}(M'_{i',j'} N'_{i'',j''}, m_i, p_j) = \mathsf{R}(M_{i',j'}, m_i, n_{j',i''})\mathsf{R}(N_{i'',j''}, n_{j',i''}, p_j). \tag{10}$$

As to the the left-hand side of (9), the definitions of the matrices $\tilde{M}_{\ell,i',j'}$ and $\tilde{N}_{\ell,i'',j''}$ now give $\mathsf{R}(\tilde{M}_{\ell,i',j'} \tilde{N}_{\ell,i'',j''}, m_i, p_j) = \mathsf{R}(\mathsf{R}(M_{i',j'}, \mu_\ell, \nu_\ell)\mathsf{R}(N_{i'',j''}, \nu_\ell, \pi_\ell), m_i, p_j)$. Observe now that by assumption, we have $\mu_\ell \geq \min(m_i, m_{i'})$, $\nu_\ell \geq n_{j',i''} = \min(n_{j'}, n_{i''})$ and $\pi_\ell \geq \min(p_j, p_{j''})$. Thus, Lemma 3 now implies

$$\mathsf{R}(\tilde{M}_{\ell,i',j'} \tilde{N}_{\ell,i'',j''}, m_i, p_j) = \mathsf{R}(M_{i',j'}, m_i, n_{j',i''})\mathsf{R}(N_{i'',j''}, n_{j',i''}, p_j). \tag{11}$$

Combining (10) and (11) finishes the proof. □

We continue with a series of lemmas, that leads to Lemma 3 used in the former proof.

Lemma 1. *Let A and B be matrices of respective sizes $(r \times s)$ and $(s \times t)$ and let r', t' be integers. Then $\mathsf{R}(AB, r', t') = \mathsf{R}(A, r', s)\,\mathsf{R}(B, s, t')$.*

Proof. This lemma simply says that one can add / remove rows to A or columns to B either before or after computing the product AB, and get the same result in both cases. □

Lemma 2. *Let A and B be matrices of respective sizes $(r \times s)$ and $(s' \times t)$ and let $r', t', \rho, \sigma, \tau$ be integers, with $\rho \geq \min(r, r')$ and $\tau \geq \min(t, t')$. Then, the following holds:*

- $\mathsf{R}(\mathsf{R}(A, \rho, \sigma), r', \sigma) = \mathsf{R}(A, r', \sigma)$
- $\mathsf{R}(\mathsf{R}(B, \sigma, \tau), \sigma, t') = \mathsf{R}(B, \sigma, t')$.

Proof. We prove only the first item; the second one is the exact analogue for column operations. If $r' \geq r$, we deduce that $\rho \geq r$, so resizing A in row-size ρ introduces new rows of zeros; after resizing in row-size r', we thus obtain exactly $\mathsf{R}(A, r', \sigma)$. If $r \geq r'$, we deduce that $\rho \geq r'$; independently of whether $\rho \geq r$ or not, $\mathsf{R}(\mathsf{R}(A, \rho, \sigma), r', \sigma)$ is thus obtained by resizing A is row-size r', by removing $r - r'$ rows. □

Lemma 3. *Let A and B be matrices of respective sizes $(r \times s)$ and $(s' \times t)$ and let $r', t', \rho, \sigma, \tau$ be integers, with $\rho \geq \min(r, r')$, $\sigma \geq \min(s, s')$ and $\tau \geq \min(t, t')$. Then, letting $s'' = \min(s, s')$, the following equality holds:*

$$\mathsf{R}(\mathsf{R}(A, \rho, \sigma)\, \mathsf{R}(B, \sigma, \tau), r', t') = \mathsf{R}(A, r', s'')\, \mathsf{R}(B, s'', t').$$

Proof. Lemma 1 shows that the left-hand side equals $\mathsf{R}(\mathsf{R}(A, \rho, \sigma), r', \sigma)\, \mathsf{R}(\mathsf{R}(B, \sigma, \tau), \sigma, t')$. Applying Lemma 2 shows that it is equal to

$$\mathsf{R}(A, r', \sigma)\, \mathsf{R}(B, \sigma, t').$$

It remains to prove that this product equals $\mathsf{R}(A, r', s'')\, \mathsf{R}(B, s'', t')$, with $s'' = \min(s, s')$. Suppose indeed that $s \leq s'$ (the argument is the same if $s \geq s'$, using B instead of A). Then, we have by assumption $s'' = s$ and $\sigma \geq s$, so $\mathsf{R}(A, r', \sigma)$ is obtained by adding some zero columns to A, and changing the number of rows to r'. The zero columns do not participate in the product, which thus equals $\mathsf{R}(A, r', s)\, \mathsf{R}(B, s, t')$, as requested. □

To conclude this section, we revisit the example of Subsection 2.2. There, we had $m = 3, n = 3, p = 3$ and we used the compositions $(m_1 = 2, m_2 = 1)$, $(n_1 = 2, n_2 = 1)$ and $(p_1 = 2, p_2 = 1)$. Taking $\ell = 5$ in Strassen's pattern, we get

$$R_{1,5} = 2, \quad C_{1,5} = 2, \quad R_{2,5} = 2, \quad C_{2,5} = 2, \quad R_{3,5} = 1, \quad C_{3,5} = 1,$$

so that we can take $\mu_5 = 1, \nu_5 = 2, \pi_5 = 1$, and the 5th product γ_5 can be computed in size $(1 \times 2 \times 1)$. We have thus recovered the result obtained in Subsection 2.2.

3 Filling the tables

We will now describe an automated search for new upper bounds for square sizes from 2 to 30. We build recursively a 3-dimensional table T, indexed by integers (m, n, p), where $T[m, n, p]$ gives an upper bound on the number of multiplications for a product of size $(m \times n \times p)$. While our main target is square matrix multiplication, we have to compute information for rectangular matrix products along the way.

Our search is based on the application of the former section's construction to a fixed list of patterns, based on those in Table 1; other ingredients are used as well. We describe first our list of patterns, then the process we used to build the table T; some possible further optimizations are described next, and we conclude this section with a discussion of our results.

Initial list of patterns. First, we discuss the patterns we used. Some are straightforward: `Strassen`, `Winograd`, `Laderman`, `Hopcroft323`, `Hopcroft332`, `Hopcroft233`, `Makarov` just follow the algorithms mentioned in Tables 1 and 2. We also use three other patterns, called `mul211`, `mul121` and `mul112`:

- `mul211` has length 2 and size $(2 \times 1 \times 1)$; it describes the product of a (2×1) matrix by a (1×1) matrix:

$$(U_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right), \quad (V_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 1 \end{bmatrix} \right), \quad (W_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right)$$

- `mul121` has length 2 and size $(1 \times 2 \times 1)$; it describes the product of a (1×2) matrix by a (2×1) matrix.

$$(U_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \end{bmatrix} \right), \quad (V_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right), \quad (W_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 1 \end{bmatrix} \right)$$

- `mul112` has length 2 and size $(1 \times 1 \times 2)$; it describes the product of a (1×1) matrix by a (1×2) matrix.

$$(U_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 1 \end{bmatrix} \right), \quad (V_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \end{bmatrix} \right), \quad (W_\ell)_{\ell \leq 2} = \left(\begin{bmatrix} 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \end{bmatrix} \right).$$

Using these three patterns allows us to split a product $(m \times n \times p)$ along respectively the first, second or third dimension. For instance, after fixing (m, n, p) and choosing the pattern `mul211`, we have to split m as $m = m_1 + m_2$ (n and p are not split); then, in this case, Proposition 1 simply returns $(\mu_1, \nu_1, \ell_1) = (m_1, n, p)$ and $(\mu_2, \nu_2, \ell_2) = (m_2, n, p)$. Taking (on the above example) a composition of the form $m_1 = m - 1, m_2 = 1$ thus enables us to automatically incorporate classical *peeling* strategies into our search.

Using symmetries to create new patterns. A further series of patterns was obtained using symmetries. Indeed, if one starts from a pattern $(U_\ell)_{\ell \leq L}$, $(V_\ell)_{\ell \leq L}$, $(W_\ell)_{\ell \leq L}$ of size $(a \times b \times c)$ and chooses invertible matrices X, Y, Z, of respective sizes $(a \times a)$, $(b \times b)$ and $(c \times c)$, one can define, for $\ell \leq L$:

$$U'_\ell = X U_\ell Y^{-1}, \quad V'_\ell = Y V_\ell W^{-1}, \quad W'_\ell = W U_\ell X^{-1}. \tag{12}$$

Then it is known that $(U'_\ell)_{\ell \leq L}$, $(V'_\ell)_{\ell \leq L}$, $(W'_\ell)_{\ell \leq L}$ is a valid pattern as well. In the following paragraphs, we denote this new pattern $S_{X,Y,Z}(U, V, W)$.

We started from the list of patterns `Strassen`, `Winograd`, `Laderman`, `Hopcroft233`, `Hopcroft323`, `Hopcroft332` and `Makarov`; there was no point in applying such transforms to `mul211`, `mul121` and `mul112`. For each pattern $(U_\ell)_{\ell \leq L}$, $(V_\ell)_{\ell \leq L}$, $(W_\ell)_{\ell \leq L}$ in this list, we created a sequence of matrices (X_i, Y_i, Z_i) as follows: for each i, two of (X_i, Y_i, Z_i) were set to the identity, and the last one was a permutation matrix (all possible permutations were used). Then, we created all transforms $S_{X_i,Y_i,Z_i}(U, V, W)$, and added them to our pool.

Recursive completion of the table. The table T is built as follows. Let (m, n, p) be the given target size. If one of m, n, p is equal to 1, $T[m, n, p]$ receives the product mnp. Else, to obtain the entry $T[m, n, p]$, we apply the following process.

We loop over all patterns in the list obtained previously. For a given pattern $(U_\ell)_{\ell \leq L}$, $(V_\ell)_{\ell \leq L}$, $(W_\ell)_{\ell \leq L}$ of size $(a \times b \times c)$, we do the following. We start by determining compositions $\mathbf{m} = (m_1, \ldots, m_a)$ of m, $\mathbf{n} = (n_1, \ldots, n_b)$ of n and $\mathbf{p} = (p_1, \ldots, p_c)$ of p (details follow). For each such composition, we determine integers $(\mu_\ell)_{\ell \leq L}$, $(\nu_\ell)_{\ell \leq L}$ and $(\pi_\ell)_{\ell \leq L}$ using Proposition 1: these integers tell us what are the sizes of the products to do recursively.

Then, we determine the cost associated to $((U_\ell), (V_\ell), (W_\ell), \mathbf{m}, \mathbf{n}, \mathbf{p})$. A first estimate is obviously the sum of $T[\mu_\ell, \nu_\ell, \pi_\ell]$, for $\ell \leq L$. However, better may be done by matching products and using Pan's simultaneous products techniques [25] (this idea was suggested in [31]). For any pair $\ell < \ell' \leq L$, Pan's technique is applicable if either $(\mu_\ell, \nu_\ell, \pi_\ell) = (\nu_{\ell'}, \pi_{\ell'}, \mu_{\ell'})$ or $(\mu_{\ell'}, \nu_{\ell'}, \pi_{\ell'}) = (\nu_\ell, \pi_\ell, \mu_\ell)$. In this case, we can compute both products using $\mu_\ell\nu_\ell\pi_\ell + \mu_\ell\nu_\ell + \mu_\ell\pi_\ell + \nu_\ell\pi_\ell$ base ring multiplications. If this is lower than the sum $T[\mu_\ell, \nu_\ell, \pi_\ell] + T[\mu_{\ell'}, \nu_{\ell'}, \pi_{\ell'}]$, we tag the pair $\{\ell, \ell'\}$. Then, we need to determine what pairs of products should be matched and computed using Pan's technique, and what products should be looked up in the table T. Ideally, a matching algorithm would tell us the optimal choice; however, in our situations, the number of tagged products was so small that an exhaustive search was always fast enough.

At this point, we have obtained an estimate for $T[m, n, p]$ using the pattern $(U_\ell)_{\ell \leq L}$, $(V_\ell)_{\ell \leq L}$, $(W_\ell)_{\ell \leq L}$ and the composition $\mathbf{m}, \mathbf{n}, \mathbf{p}$. We loop over all patterns and all compositions and keep the minimum.

Other cost estimates are considered. For all sizes (m, n, p), if one of m, n, p is equal to 2, we take into account the cost of Hopcroft and Kerr's algorithm [14]. For square matrices, we also determine the cost provided by *trilinear aggregating techniques*. We use a slightly better cost estimate than previously published results:

- if n is even, one can compute $(n \times n \times n)$ matrix products using $(n^3 + 12n^2 + 11n)/3$ multiplications
- if n is odd, one can compute $(n \times n \times n)$ matrix products using $(n^3 + 15n^2 + 14n - 6)/3$ multiplications.

Recall that Table 2 (in the introduction) reported a cost that is the minimum of $(n^3 + 12n^2 + 17n)/3$ and $(n^3 + 11.25n^2 + 32n + 27)/3$, for n even. Our result improves on the former for all n, and on the latter for n in our range of interest. The proof of our result follows closely previous work by Pan [25, 23] and Laderman, Pan and Sha [18], and differs considerably from the rest of this article: formulas are given that allow to almost divide by 3 the number of multiplications, without using any recursive algorithm. For these reasons, the proof is given in the appendix of this paper (see Proposition 2).

Finally, if the base ring is commutative, we also take into account Makarov's commutative algorithm for the $(3 \times 3 \times 3)$ case, as well as Waksman's algorithm in general.

Composition strategies For the patterns `mul211`, `mul121` and `mul112`, we use all possible compositions for (m, n, p). For larger patterns, two strategies were used. The first one is a brute-force approach, where all compositions were tried: this is used for integers up to 6. The second approach is more balanced: given an integer $m \geq 7$ to divide into a parts, we take $\mathbf{m} = (m_0, \ldots, m_0, m_1, \ldots, m_1)$ and all its permutations, with $m_0 = m \text{ div } a$ and $m_1 = m_0 + 1$, where m_0 is repeated $(m \bmod a)$ times.

Further optimizations. Further attempts were made to obtain better results; while they were not successful, we list them here for completeness.

Symmetries. Instead of a single application of the symmetry transformations, we also generated further transforms of the form

$$S_{X_{i_1},Y_{i_1},Z_{i_1}}(U,V,W), \quad S_{X_{i_2},Y_{i_2},Z_{i_2}}(S_{X_{i_1},Y_{i_1},Z_{i_1}}(U,V,W)), \quad \ldots$$

We applied up to three transforms, and used other matrices than just permutations (namely, upper or lower triangular matrices with entries in $\{0, \pm 1, \pm 2\}$).

Patterns. We used patterns from Johnson and McLoughlin's list [15] for the $(3 \times 3 \times 3)$ case; these patterns involve free parameters, which we set to ± 1.

Compositions. To partition an integer m into a parts, we also tried taking $\mathbf{m} = (m_0, \ldots, m_0, m_1)$, with $m_0 = m \text{ div } a$ and $m_1 = m - (a-1)m_0$, as well as all its permutations.

Simultaneous pairs of products. We relaxed the conditions of applicability of Pan's algorithm for two simultaneous products: padding with a zero row or column if necessary, we allowed up to one unit difference between the products' sizes (and accounted for it in the cost estimate), to try to generate more matches.

Simultaneous triples of products. Finally, it would have been possible to use Pan's trilinear aggregating techniques for *three* simultaneous products; however, the sizes we considered were too small for it to be fruitful.

Results. The results of our search are given for square multiplication problems sizes up to $(30 \times 30 \times 30)$ in Tables 3 and 4, for respectively non-commutative and commutative base rings. The tables give the number of multiplications we obtained, the technique (either the pattern name, TA (for Trilinear Aggregating) or Waksman), and a comparison to previous work: new results are in bold face.

The tables do not give the details of what compositions were used, and what products were paired; also, we do not mention the various rectangular sizes that are needed as subproducts. All these details are given at the address `http://www.csd.uwo.ca/~mislam63/`, together with a toy code generator.

Before commenting on these results, we mention that using the "basic" search parameters described first, it takes about 1h to complete the searches for both the non-commutative and commutative cases on a recent PC, using a Magma [5] implementation. Adding all

extra options described in the last paragraph, the time rises to 5h. The search in the non-commutative case is faster, as we can exploit the fact that $T[m,n,p]$ is invariant under permutations of $\{m,n,p\}$.

Table 3. Let us first comment on the non-commutative case, in Table 3. For small powers of 2, as could be expected, no combination outperforms Strassen's algorithm. The algorithms of Laderman and Makarov are used only for the base cases (resp. 3 and 5); their relatively high exponents seem to make them useless for larger sizes. For sizes from 20 on, trilinear aggregating takes a clear lead (this was already the case in the tables of [26, 31]). In most other cases, our approach improves the previous results, by up to 10%.

An important observation is that even though Strassen's and Winograd's patterns both perform 7 multiplications, the difference in sparseness makes them nonequivalent for our purposes: the quantities $(\mu_\ell, \nu_\ell, \pi_\ell)$ obtained with these two patterns from Proposition 1 will in general not coincide. Both are useful: the results given here *require* Strassen's and Winograd's patterns, as well as a symmetry transform of the latter, called `Winograd2` in the table. This pattern is obtained by taking for X the (2×2) transposition matrix, and for Y, Z the (2×2) identity matrix in Eq. (12). Without putting all these three patterns in our list, we obtain inferior results (e.g., we would obtain 2116 in size 15 using only `Strassen` and `Winograd`).

Finally, we mention that the technique of pairing two simultaneous subproducts is found to be useful from size 13 on.

Table 4. Next, we comment on the commutative case, in Table 4. Our results are compared to Mezzarobba's [21], who gave a similar table. Mezzarobba's conclusion was that a combination of Strassen's and Waksman's algorithm was sufficient in almost all situations. In our table, it appears that our composition techniques allow to find significantly better estimates in many cases, using more complex compositions.

Besides, it is also worthwhile to mention that in many cases, commutativity does not help much. While the results in Table 4 are always better than, or equal to, those in Table 3, the gaps are not very large (up to about 10%).

Dimension	Muls	Algorithm	Probert-Fischer (1980)	Smith (2002)
2 2 2	7	`Strassen`	7	7
3 3 3	23	`Laderman`	23	23
4 4 4	49	`Strassen`	49	49
5 5 5	100	`Makarov`	103	100
6 6 6	161	`Strassen`	161	161
7 7 7	**258**	`Winograd`	276	273
8 8 8	343	`Strassen`	343	343
9 9 9	**522**	`mul121`	529	527
10 10 10	700	`Strassen`	710	700
11 11 11	**923**	`Strassen`	996	992
12 12 12	1125	`mul121`	1125	1125
13 13 13	**1450**	`Strassen`	1594	1580
14 14 14	**1728**	`Strassen`	1792	1743
15 15 15	**2108**	`Winograd2`	2369	2300
16 16 16	2401	`Strassen`	2401	2401
17 17 17	**2972**	`Strassen`	3218	3218
18 18 18	**3306**	TA	3375	3342
19 19 19	**4073**	`Strassen`	4402	4369
20 20 20	**4340**	TA	4870	4380
21 21 21	**5365**	`Strassen`	6131	5610
22 22 22	**5566**	TA	6380	5610
23 23 23	**6806**	TA	7875	7048
24 24 24	**7000**	TA	7875	7048
25 25 25	**8448**	TA	9676	8710
26 26 26	**8658**	TA	9880	8710
27 27 27	**10330**	TA	11984	10612
28 28 28	**10556**	TA	11984	10612
29 29 29	**12468**	TA	14360	not listed
30 30 30	**12710**	TA	14360	not listed

Table 3: Upper bounds on the number of multiplications, non-commutative case

Dimension	Muls	Algorithm	Mezzarobba (2007)
2 2 2	7	`Strassen`	7
3 3 3	**22**	`Makarov333`	23
4 4 4	46	Waksman	46
5 5 5	93	Waksman	93
6 6 6	141	Waksman	141
7 7 7	235	Waksman	235
8 8 8	316	Waksman	316
9 9 9	**472**	`mul121`	473
10 10 10	595	Waksman	595
11 11 11	**825**	`mul121`	831
12 12 12	987	`Strassen`	987
13 13 13	**1318**	`mul121`	1333
14 14 14	**1525**	`mul121`	1561
15 15 15	**1941**	`mul121`	2003
16 16 16	2212	`Strassen`	2212
17 17 17	**2762**	`Hopcroft332`	2865
18 18 18	**3060**	`Hopcroft332`	3231
19 19 19	**3757**	`mul121`	3943
20 20 20	**4158**	`Strassen`	4165
21 21 21	**4938**	`Strassen`	5261
22 22 22	**5440**	`mul121`	5610
23 23 23	**6382**	`Hopcroft332`	6843
24 24 24	**6900**	`Hopcroft332`	6909
25 25 25	**8083**	`mul121`	8710
26 26 26	**8658**	TA	8710
27 27 27	**9994**	`mul121`	10612
28 28 28	**10556**	TA	10612
29 29 29	**12109**	`mul121`	not listed
30 30 30	**12710**	TA	not listed

Table 4: Upper bounds on the number of multiplications, commutative case

4 Experiments

Along with the computer search, we developed a proof-of-concept code generator that produces multiplication functions for various kinds of entries. Using predefined functions for base ring arithmetic, the code generator produces a series of functions `mul111`, ..., `mul303030` and auxiliary functions for the required rectangular cases; these functions contain as hard-coded information what compositions are done, what products are done recursively, and how.

4.1 Setup

Our experiments used entries of the following kinds: multiprecision integers, univariate polynomials over $\mathbb{F}_p$, differential operator with coefficients in $\mathbb{F}_p[x]$ and linear recurrence operators with coefficients in $\mathbb{F}_p[n]$. The last two types are examples of non-commutative rings over which one is interested in linear algebra algorithms.

The target implementation language was either C (for integers) or C++ (for other base rings). For the integer case, we used the GMP library [1] to provide base ring arithmetic; for other base rings, we used the NTL C++ package [30] to provide polynomial arithmetic in $\mathbb{F}_p[x]$. To handle the non-commutative cases, extra work was needed. Recall that a differential operator has the form $a = \sum_{i=0}^{d} a_i \partial^i$, where the a_i are in $\mathbb{F}_p[x]$, and ∂ represents the differentiation operator $\partial/\partial x$. The multiplication of such operators is given by

$$\sum_{i=0}^{d} a_i \partial^i \;\times\; \sum_{j=0}^{e} b_j \partial^j \;=\; \sum_{i=0}^{d} a_i \sum_{j=0}^{e} \sum_{k=0}^{i} \binom{i}{k} b_j^{(k)} \partial^{i+j-k},$$

where $b_j^{(k)}$ is the kth derivative of b_j. In a similar manner, a linear recurrence has the form $a = \sum_{i=0}^{d} a_i E^i$, where a_i are in $\mathbb{F}_p[n]$, and E stands for the *shift operator*. The multiplication formula becomes

$$\sum_{i=0}^{d} a_i E^i \;\times\; \sum_{j=0}^{e} b_j E^j \;=\; \sum_{i=0}^{d} a_i \sum_{j=0}^{e} b_j(n+i) E^{i+j}.$$

No quasi-linear time algorithm is known for multiplication of differential operators or recurrences: for operators of order n with coefficients of degree n, the formulas above induce $O(n^2)$ multiplications of polynomials of degree n. Better can be done in the case of differential operators [36, 6], but we do not use these techniques here.

4.2 Results

We describe here timings obtained by our implementation. The purpose of these experiments is to understand what practical gain can be expected from results such as ours compared to a naive implementation. Remark that for this goal, comparing to existing systems offers little insight, since we would not be able to separate the gain, or loss, induced by the matrix algorithms, to that induced by the differences in base ring arithmetic.

We also stress the fact that we do not attempt to achieve optimal performance. Indeed, for entries such as above, optimizing the implementation would require implementing other strategies, typically Chinese Remaindering, combining them to our approach, and performing many optimizations, such as saving additions or memory: this is out of the scope of this paper. On the contrary, our point of view here is to describe what is achievable by taking the base ring arithmetic, as well as memory allocation, as given (this is typically the case when developing in a high-level environment).

In the graphs below, we give a slightly more complete information than the one in the tables of the previous section. We give timings for families of algorithms (trilinear aggregating, and Waksman's algorithm, when applicable) for *all* sizes; the corresponding curves in the graphs are called TA and Waksman, respectively. Besides, to show what can be done using our table look-up techniques, we give timings for these techniques for all sizes, even for the few dimensions between 20 and 30 where trilinear aggregating was the best solution in terms of number of multiplications.

Commutative entries The following graphs give timings for integers and polynomials. All timings are averaged over 150 runs, and were obtained on an Intel Core 2 Duo, CPU speed 2.4 GHz, with 3 GB of RAM.

In the integer case (Figure 1), we use integers of size 1000 bit. This figure illustrates that a low multiplication count does not imply a faster running time. Indeed, for 1000 bit integers, a GMP multiplication is only about 8 times slower than an addition. As a consequence, trilinear aggregating techniques (which are in theory quite competitive with the other solutions, and much better than the naive algorithm) are significantly slower in practice, due to the large amount of additions (and also memory allocation) they require. The jump at size 26 for our table look-up techniques is seemingly due to the fact that for this size, the recursion tree is deeper than for nearby sizes.

For polynomials of degree 100 over $\mathbb{F}_{9001}$ (Figure 2), the curve for our table approach is much smoother, and the results reflect much more closely the number of multiplications. This comes as no surprise: for such degrees, in NTL, a polynomial multiplication is about 65 times slower than an addition, so the number of multiplication becomes a key factor.

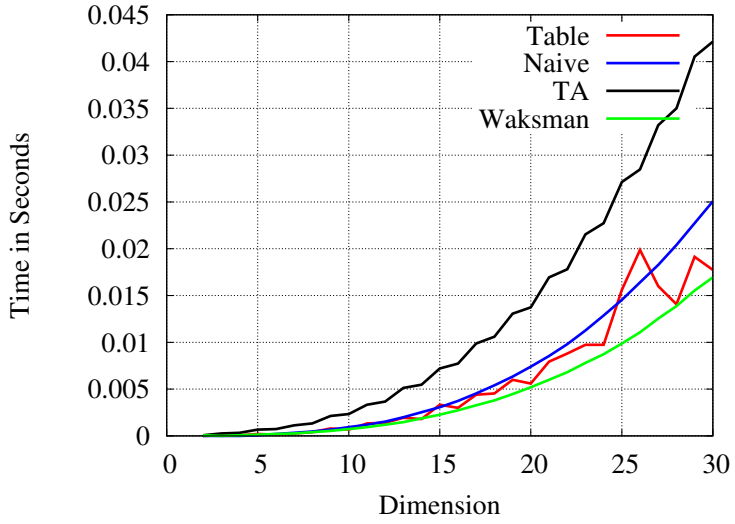

Figure 1: Timings for integer entries (length 1000 bits)

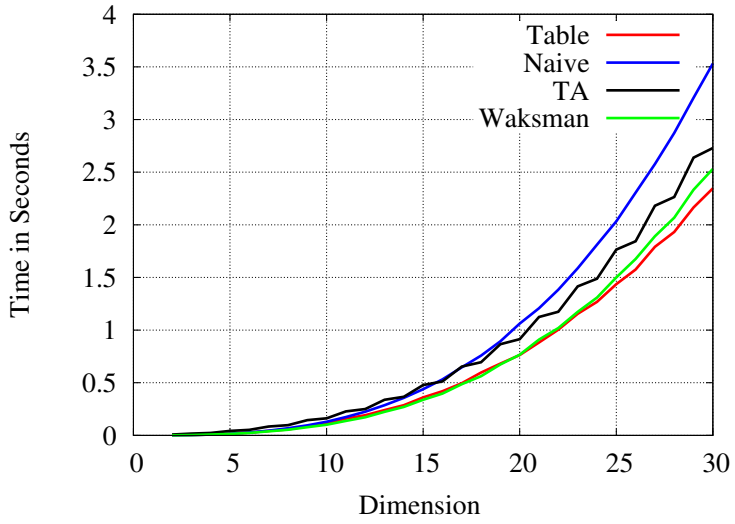

Figure 2: Timings for polynomial entries (degree 100)

Non-commutative entries In the next examples, the entries of the matrices are differential operators (resp. linear recurrences) of order 10, with polynomial coefficients of degree 10 over $\mathbb{F}_{9001}$; here, the timings are averaged over 10 runs. Remark that Waksman's algorithm cannot be employed here, since multiplication is not commutative.

For such cases, the cost of multiplication is much higher than that of other operations (more than a thousand times more than addition, for instance), so that saving multiplications pays off very quickly. Still, in the examples below, only 87% to 94% of the running time is spent on multiplication, so that the other operations and memory management are not completely negligible.

This partially explains why trilinear aggregating techniques do not perform quite as well as expected. For the degrees close to 30, these techniques are the best in terms of number of multiplications, by up to 15% compared to the table look-up approach, but as we can see, the graphs do not quite reflect this dominance.

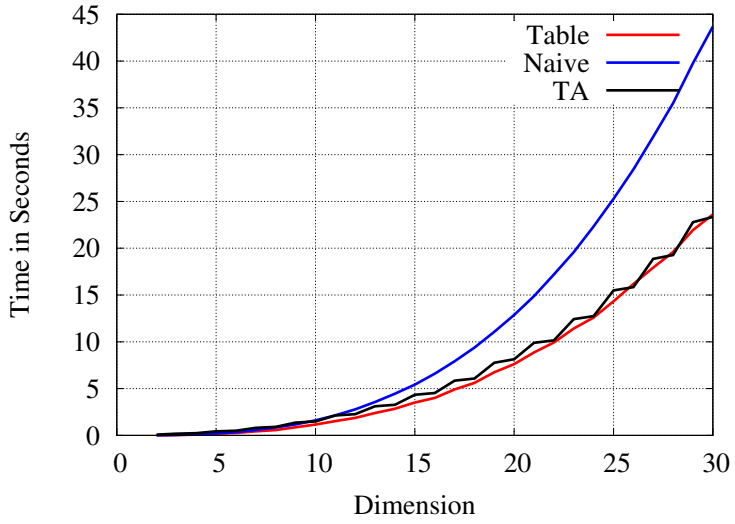

Figure 3: Timings for differential operator entries

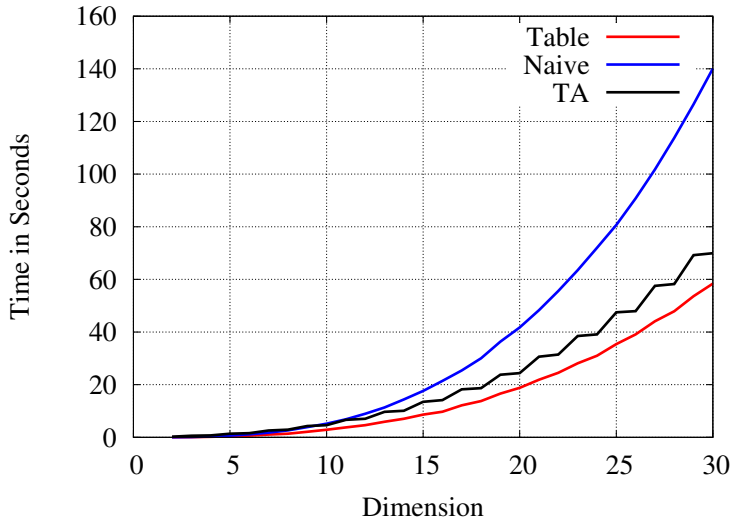

Figure 4: Timings for linear recurrence entries

5 Conclusion

The complexity of multiplying small matrices remains imperfectly understood. In this paper, our interest was first of all theoretical: we wanted to tabulate the best that can be done for a suitable range of small product sizes.

We showed how an appropriate combination of formerly known techniques results in improved upper bounds in many cases, and how the search for such combinations can be

automatized. As reported in Section 3, many optimizations were attempted to go further, but without success; it is possible that we have reached the limits of what this approach can offer without a significant new ingredient.

References

[1] GMP. `http://gmplib.org`.

[2] B. Beckermann and G. Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM J. Matrix Analysis and Applic.*, 15(3):804–823, 1994.

[3] M. Bläser. On the complexity of the multiplication of matrices of small formats. *Journal of Complexity*, 19(1):43–60, 2003.

[4] M. Bläser. Beyond the Alder-Strassen bound. *Theoretical Computer Science*, 331(1):3–21, 2005.

[5] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symb. Comp.*, 24(3-4):235–265, 1997.

[6] A. Bostan, F. Chyzak, and N. Le Roux. Products of ordinary differential operators by evaluation and interpolation. In *ISSAC'08*, pages 23–30. ACM, 2008.

[7] R. P. Brent. Algorithms for matrix multiplication. *Report TR-CS-70-157, DCS, Stanford*, page 52, 1970.

[8] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *Journal of Algorithms*, 1(3):259–295, 1980.

[9] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grund. der Math. Wiss.* Springer, 1997.

[10] D. V. Chudnovsky and G. V. Chudnovsky. Computer algebra in the service of mathematical physics and number theory. In *Computers in mathematics*, pages 109–232. Dekker, 1990.

[11] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[12] P. D'Alberto and A. Nicolau. Adaptive Strassen's matrix multiplication. In *ICS'07*, pages 284–292. ACM, 2007.

[13] J. Hopcroft and J. Musinski. Duality applied to the complexity of matrix multiplications and other bilinear forms. In *STOC'73*, pages 73–87. ACM, 1973.

[14] J. E. Hopcroft and L. R. Kerr. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM Journal on Applied Math.*, 20(1):30–36, 1971.

[15] R. Johnson and A. McLoughlin. Noncommutative bilinear algorithms for 3×3 matrix multiplication. *SIAM Journal on Computing*, 15(2):595–603, 1986.

[16] I. Kaporin. The aggregation and cancellation techniques as a practical tool for faster matrix multiplication. *Theor. Comput. Sci.*, 315(2-3):469–510, 2004.

[17] J. Laderman. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bull. Amer. Math. Soc.*, 82:126–128, 1976.

[18] J. Laderman, V. Pan, and X. H. Sha. On practical algorithms for accelerated matrix multiplication. *Linear Algebra Appl.*, 162/164:557–588, 1992.

[19] O. M. Makarov. An algorithm for multiplication of 3×3 matrices. *Zh. Vychisl. Mat. i Mat. Fiz.*, 26(2):293–294, 320, 1986.

[20] O. M. Makarov. A noncommutative algorithm for multiplying 5×5 matrices using one hundred multiplications. *U.S.S.R. Comput. Maths. Phys.*, 27:205–207, 1987.

[21] M. Mezzarobba. Génération automatique de procédures numériques pour les fonctions D-finies. Master's thesis, Master parisien de recherche en informatique, 2007.

[22] V. Pan. New fast algorithms for matrix operations. *SIAM J. Comput.*, 9(2):321–342, 1980.

[23] V. Pan. Trilinear aggregating with implicit canceling for a new acceleration of matrix multiplication. *Comput. Math. Appl.*, 8(1):23–34, 1982.

[24] V. Pan. How can we speed up matrix multiplication? *SIAM Rev.*, 26(3):393–415, 1984.

[25] V. Pan. *How to multiply matrices faster*, volume 179 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

[26] R. L. Probert and P. C. Fischer. Decomposition techniques for matrix multiplication problems. *Utilitas Mathematica*, 18:257–267, 1980.

[27] G. Schachtel. A non-commutative algorithm for multiplying 5×5 matrices using 103 multiplications. *Inf. Proc. Lett.*, 7:180–182, 1978.

[28] A. Schönhage. Partial and total matrix multiplication. *SIAM J. Comput.*, 10(3):434–455, 1981.

[29] É. Schost. Complexity results for triangular sets. *J. Symb. Comput.*, 36(3-4):555–594, 2003.

[30] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net`.

[31] W. D. Smith. Fast matrix multiplication formulae – report of the prospectors. `http://www.math.temple.edu/~wds/prospector.pdf`.

[32] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[33] V. Strassen. The asymptotic spectrum of tensors. *Journal für die reine und angewandte Mathematik*, 384:102–152, 1988.

[34] V. Strassen. Degeneration and complexity of bilinear maps: Some asymptotic spectra. *Journal für die reine und angewandte Mathematik*, 413:127–180, 1991.

[35] O. Sýkora. A fast non-commutative algorithm for matrix multiplication. In *Mathematical Foundations of Computer Science*, number 53 in LNCS, pages 504–512, 1977.

[36] J. van der Hoeven. FFT-like multiplication of linear differential operators. *J. Symb. Comput.*, 33(1):123–127, 2002.

[37] A. Waksman. On Winograd's algorithm for inner products. *IEEE Transactions On Computers*, C-19:360–361, 1970.

[38] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

[39] S. Winograd. On multiplication of 2×2 matrices. *Linear Algebra and its Applications*, 4:381–388, 1971.

[40] S. Winograd. On the algebraic complexity of inner product. *Linear Algebra and Appl.*, 4:377–379, 1971.

Appendix: Trilinear aggregating techniques

Trilinear aggregating is an idea due to Pan [25, 23], that enables one to perform three matrix products simultaneously, and can be adapted to perform a single product. In this appendix, we recall the main idea behind this algorithm and present a new version that improves some former ones (previous costs are reported in Table 2). Our improvements are summarized in the following proposition.

Proposition 2. *If n is even, one can compute $(n \times n \times n)$ matrix products using $(n^3+12n^2+11n)/3$ multiplications. If n is odd and greater than 3, one can compute $(n \times n \times n)$ matrix products using $(n^3 + 15n^2 + 14n - 6)/3$ multiplications.*

This appendix establishes these claims. Contrary to the rest of this paper, we will present here the algorithm under a compact form (essentially, a decomposition of the matrix multiplication tensor, that we choose to write simply under a polynomial form); besides, we will use without proof some key results that are in the literature (the main ideas are due

to Pan [25, 23], and Laderman, Pan and Sha [18]). We start with the even case, giving a review of the general approach to trilinear aggregating. Then, we give our modifications for the case of odd n (which is usually left out in previous work on this question).

As for the results in Section 3, an implementation in Magma that validates the claims in this appendix is available at the address `http://www.csd.uwo.ca/~mislam63/`

A.1 The even case

Preamble: polynomial notation. The following classical representation provides an alternative way to encode matrix multiplication algorithms. It amounts to write down a decomposition of the tensor of matrix multiplication; however, we choose to avoid the introduction of tensors, using simply commutative polynomials. This formalism will allow us to describe trilinear aggregating algorithms in a compact manner (although writing an actual implementation requires unfolding these formulas).

Consider matrices $A = [a_{i,j}]$ and $B = [b_{i,j}]$ of sizes $(m \times n)$ and $(n \times p)$, and let $\mathbf{C} = [\mathbf{c}_{i,j}]$ be an $(m \times p)$ matrix with indeterminate entries. Define the polynomial

$$\mathbf{T}(A, B, \mathbf{C}) = \sum_{1 \le i \le m,\, 1 \le j \le n,\, 1 \le k \le p} a_{i,j} b_{j,k} \mathbf{c}_{k,i},$$

and remark that for all i, k, the coefficient of $\mathbf{c}_{k,i}$ is the (i, k)-entry of the product $C = AB$; the fact that $\mathbf{c}_{k,i}$ appears in the sum instead of $\mathbf{c}_{i,k}$ is only meant to comply to the standard practice. Thus, computing $\mathbf{T}$ solves our matrix multiplication problem.

Reduction to zero-sum rows and columns. In this paragraph, we are to multiply two matrices A, B of size $(2n \times 2n)$. The following construction is inspired by [18] and follows [16]. We subdivide A, B into blocks of size $(n \times n)$ as

$$A = \begin{bmatrix} A^{1,1} & A^{1,2} \\ A^{2,1} & A^{2,2} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B^{1,1} & B^{1,2} \\ B^{2,1} & B^{2,2} \end{bmatrix};$$

here, block indices are given as superscripts, as we will need subscripts later on. Then, we define the matrices of respective sizes $((n+1) \times n)$ and $(n \times (n+1))$

$$L = \begin{bmatrix} I \\ -u \end{bmatrix}, \quad R = \begin{bmatrix} I - \frac{1}{n+1} u^t u & -\frac{1}{n+1} u^t \end{bmatrix},$$

where u is the row vector of length n with all entries equal to 1; if needed, we will write L_n and R_n to indicate the dependency in n.

We can then define $\tilde{A}^{i,j} = L\, A^{i,j}\, R$ and $\tilde{B}^{i,j} = L\, B^{i,j}\, L^t$; these are matrices of size $((n+1) \times (n+1))$. These matrices have zero-sum rows and columns, and this property underlies Lemma 4 below. Finally, we define the matrices $\tilde{A}$ and $\tilde{B}$ of size $((2n+2) \times (2n+2))$ by

$$\tilde{A} = \begin{bmatrix} \tilde{A}^{1,1} & \tilde{A}^{1,2} \\ \tilde{A}^{2,1} & \tilde{A}^{2,2} \end{bmatrix} \quad \text{and} \quad \tilde{B} = \begin{bmatrix} \tilde{B}^{1,1} & \tilde{B}^{1,2} \\ \tilde{B}^{2,1} & \tilde{B}^{2,2} \end{bmatrix},$$

and let $\tilde{C} = \tilde{A}\tilde{B}$. Knowing $\tilde{C}$, we can recover the result $C = AB$: we decompose $\tilde{C}$ into blocks of size $((n+1) \times (n+1))$ and obtain $C^{i,j}$ by discarding the last row and column of $\tilde{C}^{i,j}$.

Computing C. To simplify notation, let $m = n + 1$; for $i, j \in \{1, 2\}$, let then $\tilde{\mathbf{C}}^{i,j}$ be a matrix of indeterminates $\mathbf{c}_{a,b}^{i,j}$ of size $(m \times m)$. Following the description in the first paragraph, computing $\mathbf{T}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$ gives us $\tilde{C}$, and thus C.

However, we are not interested in all of $\mathbf{T}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$, since we are only interested in four blocks of size $(n \times n)$ in $\tilde{C}$. Hence, we can reduce the index sets used in our sums, using the following rule: any term of the form $\mathbf{c}_{a,b}^{i,j}$, with either $a = m$ or $b = m$, can be discarded. Thus, we let $\mathbf{t}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$ be obtained from $\mathbf{T}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$ by setting any such $\mathbf{c}_{a,b}^{i,j}$ to zero, and we note that computing $\mathbf{t}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$ is enough to compute C.

Decomposing the sum. Still following Pan's ideas, we introduce here a decomposition of the sum $\tilde{t}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$. In the following definition, $R, S, T, U, V, W, X, Y, Z$ denote matrices of size $(m \times m)$. Consider the sets

$$
\begin{aligned}
\mathcal{S}_1 &= \{(i,j,k),\ 1 \le i \le j < k \le m \text{ or } 1 \le k < j \le i \le m\},\\
\mathcal{S}_2 &= \{(i,j,k),\ 1 \le i,j,k \le m\} - \{(i,i,i),\ 1 \le i \le m\}\\
s_1 &= \{(i,j,k),\ (i,j,k) \in \mathcal{S}_1 \text{ and } (i,j,k) \text{ contains at most one index equal to } m\}\\
s_2 &= \{(i,j,k),\ (i,j,k) \in \mathcal{S}_2 \text{ and } (i,j,k) \text{ contains at most one index equal to } m\}
\end{aligned}
$$

and

$$
\begin{aligned}
\mathbf{s}_0(U,V,W) &= \sum_{1 \le i < m} 9u_{i,i}v_{i,i}w_{i,i}\\
\mathbf{s}_1(U,V,W) &= \sum_{(i,j,k) \in s_1} (u_{i,j} + u_{j,k} + u_{k,i})(v_{j,k} + v_{k,i} + v_{i,j})(w_{k,i} + w_{i,j} + w_{j,k})\\
\mathbf{s}_2(R,S,T,\ U,V,W,\ X,Y,Z) &= \sum_{(i,j,k) \in s_2} (r_{i,j} + u_{j,k} + x_{k,i})(s_{j,k} + v_{k,i} + y_{i,j})(t_{k,i} + w_{i,j} + z_{j,k})\\
\mathbf{u}_1(R,Y,T,W,Z) &= \sum_{1 \le i,j < m, i \ne j} r_{i,j}y_{i,j} \sum_{1 \le k \le m} w_{i,j} + t_{k,i} + z_{j,k}\\
\mathbf{u}_2(R,Y,T,W,Z) &= \sum_{1 \le i < m} r_{i,i}y_{i,i} \sum_{1 \le k \le m} w_{i,i} + t_{k,i} + z_{i,k}\\
\mathbf{u}_3(R,Y,T) &= \sum_{1 \le i < m} r_{i,m}y_{i,m} \sum_{1 \le k < m} t_{k,i}\\
\mathbf{u}_4(R,Y,Z) &= \sum_{1 \le j < m} r_{m,j}y_{m,j} \sum_{1 \le k < m} z_{j,k}\\
\mathbf{u}(R,Y,T,W,Z) &= \mathbf{u}_1(R,Y,T,W,Z) + \mathbf{u}_2(R,Y,T,W,Z)\\
&\quad + \mathbf{u}_3(R,Y,T) + \mathbf{u}_4(R,Y,Z).
\end{aligned}
$$

The following lemma shows how to compute the product $C = AB$ from such expressions. The proof follows e.g. [18] in a straightforward manner, up to a few modifications: previous work gave expressions for the full sum $\mathbf{T}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$ instead of $\mathbf{t}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}})$, and the decomposition of $\mathbf{u}$ used above is new.

Lemma 4. *The following equality holds:*

$$\begin{aligned}
\mathbf{t}(\tilde{A}, \tilde{B}, \tilde{\mathbf{C}}) &= \mathbf{s}_0(\tilde{A}^{1,1}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}) + \mathbf{s}_0(\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2}) \\
&+ \mathbf{s}_0(\tilde{A}^{1,2} - \tilde{A}^{1,1} + \tilde{A}^{2,1},\ \tilde{B}^{2,1} + \tilde{B}^{1,2} + \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1} - \tilde{\mathbf{C}}^{1,2} + \tilde{\mathbf{C}}^{2,1}) \\
&+ \mathbf{s}_0(\tilde{A}^{1,2} + \tilde{A}^{2,1} - \tilde{A}^{2,2},\ \tilde{B}^{2,2} + \tilde{B}^{1,2} + \tilde{B}^{2,1},\ \tilde{\mathbf{C}}^{1,2} + \tilde{\mathbf{C}}^{2,2} - \tilde{\mathbf{C}}^{2,1}) \\
&+ \mathbf{s}_1(\tilde{A}^{1,1}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}) + \mathbf{s}_1(\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2}) \\
&+ \mathbf{s}_2(\tilde{A}^{1,2},\ \tilde{B}^{2,1},\ \tilde{\mathbf{C}}^{1,1},\ -\tilde{A}^{1,1},\ \tilde{B}^{1,2}, -\tilde{\mathbf{C}}^{1,2},\ \tilde{A}^{2,1},\ \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{2,1}) \\
&+ \mathbf{s}_2(\tilde{A}^{1,2},\ \tilde{B}^{2,2},\ \tilde{\mathbf{C}}^{1,2},\ \tilde{A}^{2,1},\ \tilde{B}^{1,2},\ \tilde{\mathbf{C}}^{2,2},\ -\tilde{A}^{2,2},\ \tilde{B}^{2,1}, -\tilde{\mathbf{C}}^{2,1}) \\
&- \mathbf{u}(\ \tilde{A}^{1,1}, \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1},\ \tilde{\mathbf{C}}^{1,1},\ \tilde{\mathbf{C}}^{1,1}) - \mathbf{u}(\ \tilde{A}^{1,2}, \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2},\ \tilde{\mathbf{C}}^{2,1}) \\
&- \mathbf{u}(-\tilde{A}^{1,1}, \tilde{B}^{2,1},\ -\tilde{\mathbf{C}}^{1,2},\ \tilde{\mathbf{C}}^{2,1},\ \tilde{\mathbf{C}}^{1,1}) - \mathbf{u}(\ \tilde{A}^{2,1}, \tilde{B}^{1,2},\ \tilde{\mathbf{C}}^{2,1},\ \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2}) \\
&- \mathbf{u}(\ \tilde{A}^{1,2}, \tilde{B}^{2,1},\ \tilde{\mathbf{C}}^{1,2},\ \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1}) - \mathbf{u}(\ \tilde{A}^{2,1}, \tilde{B}^{2,2},\ \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1},\ \tilde{\mathbf{C}}^{1,2}) \\
&- \mathbf{u}(-\tilde{A}^{2,2}, \tilde{B}^{1,2},\ -\tilde{\mathbf{C}}^{2,1},\ \tilde{\mathbf{C}}^{1,2},\ \tilde{\mathbf{C}}^{2,2}) - \mathbf{u}(\ \tilde{A}^{2,2}, \tilde{B}^{2,2},\ \tilde{\mathbf{C}}^{2,2},\ \tilde{\mathbf{C}}^{2,2},\ \tilde{\mathbf{C}}^{2,2}).
\end{aligned}$$

Our improvements. We are going to simplify some terms of the form $\mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4$. Decompose $\mathbf{u}_2(\tilde{A}^{1,1}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}, \tilde{\mathbf{C}}^{1,1}, \tilde{\mathbf{C}}^{1,1})$ as

$$\begin{aligned}
\mathbf{u}_2(\tilde{A}^{1,1}, \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1}, \tilde{\mathbf{C}}^{1,1}, \tilde{\mathbf{C}}^{1,1}) &= \sum_{1 \le i < m} a_{i,i}^{1,1} b_{i,i}^{1,1} \sum_{1 \le k \le m} c_{i,i}^{1,1} + c_{k,i}^{1,1} + c_{i,k}^{1,1} \\
&= \sum_{1 \le i < m} a_{i,i}^{1,1} b_{i,i}^{1,1} \left(m c_{i,i}^{1,1} + \sum_{1 \le k \le m} c_{k,i}^{1,1} + c_{i,k}^{1,1} \right)
\end{aligned}$$

and remark that, unfolding the expression of $\mathbf{u}$, the following sub-expression appears in Lemma 4:

$$\mathbf{s}_0(\tilde{A}^{1,1}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}) - \mathbf{u}_2(\tilde{A}^{1,1}, \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1}, \tilde{\mathbf{C}}^{1,1}, \tilde{\mathbf{C}}^{1,1}).$$

We can rewrite this sum as $-\mathbf{u}_2'(\tilde{A}^{1,1}, \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1})$, with

$$\mathbf{u}_2'(\tilde{A}^{1,1}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}) = \sum_{1 \le i < m} a_{i,i}^{1,1} b_{i,i}^{1,1} \left((m-9) c_{i,i}^{1,1} + \sum_{1 \le k \le m} c_{k,i}^{1,1} + c_{i,k}^{1,1} \right).$$

The same remark holds for $\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2}$, and shows that we can compute

$$\mathbf{s}_0(\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2}) - \mathbf{u}_2(\tilde{A}^{2,2}, \tilde{B}^{2,2},\ \tilde{\mathbf{C}}^{2,2}, \tilde{\mathbf{C}}^{2,2}, \tilde{\mathbf{C}}^{2,2})$$

as $-\mathbf{u}_2'(\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2})$. The other simplification comes from terms of the form $\mathbf{u}_3$ and $\mathbf{u}_4$. A quick verification shows that we can compute the sum of all $\mathbf{u}_3$ terms appearing in Lemma 4 as

$$\begin{aligned}&\mathbf{u}_3(\tilde{A}^{1,1}+\tilde{A}^{1,2}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}) + \mathbf{u}_3(\tilde{A}^{1,1}+\tilde{A}^{1,2}, \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,2})\\ +&\mathbf{u}_3(\tilde{A}^{2,1}+\tilde{A}^{2,2}, \tilde{B}^{1,2}, \tilde{\mathbf{C}}^{2,1}) + \mathbf{u}_3(\tilde{A}^{2,1}+\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2}),\end{aligned}$$

and similarly, all $\mathbf{u}_4$ terms can be grouped as

$$\begin{aligned}&\mathbf{u}_4(\tilde{A}^{1,1}, \tilde{B}^{1,1}-\tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,1}) + \mathbf{u}_4(\tilde{A}^{1,2}, \tilde{B}^{1,1}-\tilde{B}^{2,1}, \tilde{\mathbf{C}}^{2,1})\\ +&\mathbf{u}_4(\tilde{A}^{2,1}, \tilde{B}^{2,2}-\tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,2}) + \mathbf{u}_4(\tilde{A}^{2,2}, \tilde{B}^{2,2}-\tilde{B}^{1,2}, \tilde{\mathbf{C}}^{2,2}).\end{aligned}$$

Operation count. We have now obtained the final version of the multiplication algorithm. We review all components that appear, and count the number of summands in each term, as each summand can be computed in one base field multiplication (recall that multiplications by constants are not taken into account).

- $\mathbf{s}_0(\tilde{A}^{1,2}-\tilde{A}^{1,1}+\tilde{A}^{2,1},\ \tilde{B}^{2,1}+\tilde{B}^{1,2}+\tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1}-\tilde{\mathbf{C}}^{1,2}+\tilde{\mathbf{C}}^{2,1})$
  $\mathbf{s}_0(\tilde{A}^{1,2}+\tilde{A}^{2,1}-\tilde{A}^{2,2},\ \tilde{B}^{2,2}+\tilde{B}^{1,2}+\tilde{B}^{2,1},\ \tilde{\mathbf{C}}^{1,2}+\tilde{\mathbf{C}}^{2,2}-\tilde{\mathbf{C}}^{2,1})$
  each term uses $m-1$ multiplications.

- $\mathbf{s}_1(\tilde{A}^{1,1}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}), \quad \mathbf{s}_1(\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2})$
  each term uses $|s_1|$ multiplications, with $|s_1| = (m^3-m)/3-(m-1) = (m^3-4m+3)/3$.

- $\mathbf{s}_2(\tilde{A}^{1,2}, \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,1},\ -\tilde{A}^{1,1}, \tilde{B}^{1,2}, -\tilde{\mathbf{C}}^{1,2},\quad \tilde{A}^{2,1}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{2,1})$
  $\mathbf{s}_2(\tilde{A}^{1,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{1,2},\quad \tilde{A}^{2,1}, \tilde{B}^{1,2},\quad \tilde{\mathbf{C}}^{2,2},\ -\tilde{A}^{2,2}, \tilde{B}^{2,1}, -\tilde{\mathbf{C}}^{2,1})$
  each term uses $|s_2|$ multiplications, with $|s_2| = 3|s_1| = m^3-4m+3$.

- $-\mathbf{u}_1(\ \tilde{A}^{1,1}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1},\quad \tilde{\mathbf{C}}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}),\quad -\mathbf{u}_1(\ \tilde{A}^{1,2}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1})$
  $-\mathbf{u}_1(-\tilde{A}^{1,1}, \tilde{B}^{2,1},\ -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}),\quad -\mathbf{u}_1(\ \tilde{A}^{2,1}, \tilde{B}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_1(\ \tilde{A}^{1,2}, \tilde{B}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1}),\quad -\mathbf{u}_1(\ \tilde{A}^{2,1}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_1(-\tilde{A}^{2,2}, \tilde{B}^{1,2},\ -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}),\quad -\mathbf{u}_1(\ \tilde{A}^{2,2}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2},\quad \tilde{\mathbf{C}}^{2,2},\quad \tilde{\mathbf{C}}^{2,2})$
  each term uses $(m-1)^2-(m-1)$ multiplications.

- $-\mathbf{u}_2'(\ \tilde{A}^{1,1}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}),\quad -\mathbf{u}_2(\ \tilde{A}^{1,2}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1})$
  $-\mathbf{u}_2(-\tilde{A}^{1,1}, \tilde{B}^{2,1},\ -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}),\quad -\mathbf{u}_2(\ \tilde{A}^{2,1}, \tilde{B}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_2(\ \tilde{A}^{1,2}, \tilde{B}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1}),\quad -\mathbf{u}_2(\ \tilde{A}^{2,1}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_2(-\tilde{A}^{2,2}, \tilde{B}^{1,2},\ -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}),\quad -\mathbf{u}_2'(\ \tilde{A}^{2,2}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2})$
  each term uses $m-1$ multiplications.

- $-\mathbf{u}_3(\tilde{A}^{1,1}+\tilde{A}^{1,2}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}),\quad -\mathbf{u}_3(\tilde{A}^{1,1}+\tilde{A}^{1,2}, \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_3(\tilde{A}^{2,1}+\tilde{A}^{2,2}, \tilde{B}^{1,2}, \tilde{\mathbf{C}}^{2,1}),\quad -\mathbf{u}_3(\tilde{A}^{2,1}+\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2})$
  each term uses $m-1$ multiplications.

- $-\mathbf{u}_4(\tilde{A}^{1,1}, \tilde{B}^{1,1} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,1}), \quad -\mathbf{u}_4(\tilde{A}^{1,2}, \tilde{B}^{1,1} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{2,1})$
  $-\mathbf{u}_4(\tilde{A}^{2,1}, \tilde{B}^{2,2} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,2}), \quad -\mathbf{u}_4(\tilde{A}^{2,2}, \tilde{B}^{2,2} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{2,2})$
  each term uses $m-1$ multiplications.

Summing, we obtain $(8m^3 + 24m^2 - 50m + 18)/3$ multiplications. Remember that $m = n+1$, and that we are multiplying matrices A, B of size $2n$. Thus, to obtain the cost for multiplication in size n, with n even, we replace m by $n/2+1$ in the previous sum, and we finally obtain a cost of $(n^3 + 12n^2 + 11n)/3$ multiplications.

A.2 The odd case

Finally, we discuss the extension of the previous construction to the case where the matrix size is odd, of the form $2n+1$. To our knowledge, no previous mention of the optimizations arising in this case appeared before. We still split the input matrices A and B into four blocks, which are now

$A^{1,1}$ of size $((n+1)\times(n+1))$, $A^{1,2}$ of size $((n+1)\times n)$, $A^{2,1}$ of size $(n\times(n+1))$, $A^{2,2}$ of size $(n\times n)$,

and similarly for B. The matrices $\tilde{A}^{i,j}$ and $\tilde{B}^{i,j}$ are defined as before, with now

$$\tilde{A}^{i,j} = L_{(i)}\, A^{i,j}\, R_{(j)} \quad \text{and} \quad \tilde{B}^{i,j} = L_{(i)}\, B^{i,j}\, L^t_{(j)}$$

and $L_{(1)} = L_{n+1}, L_{(2)} = L_n, R_{(1)} = L_{n+1}$ and $R_{(2)} = L_n$. Let $m = n+2$; then the sizes of these matrices are as follows:

- $\tilde{A}^{1,1}$ and $\tilde{B}^{1,1}$ have size $(m \times m)$ and $\tilde{A}^{1,2}$ and $\tilde{B}^{1,2}$ have size $(m \times (m-1))$
- $\tilde{A}^{2,1}$ and $\tilde{B}^{2,1}$ have size $((m-1) \times m)$ and $\tilde{A}^{2,2}$ and $\tilde{B}^{2,2}$ have size $((m-1) \times (m-1))$.

As before, we let $\tilde{C} = \tilde{A}\tilde{B}$, which we decompose into blocks

$$\tilde{C} = \begin{bmatrix} \tilde{C}^{1,1} & \tilde{C}^{1,2} \\ \tilde{C}^{2,1} & \tilde{C}^{2,2} \end{bmatrix};$$

again, $C = AB$ is given by

$$C = \begin{bmatrix} C^{1,1} & C^{1,2} \\ C^{2,1} & C^{2,2} \end{bmatrix},$$

where we discard the last row and column of all blocks $\tilde{C}^{i,j}$.

To use the strategy given before in this case, we pad all blocks $\tilde{A}^{i,j}$ and $\tilde{B}^{i,j}$ with zeros as needed, to give them size $(m \times m)$. Thus, we add one zero column to $\tilde{A}^{1,2}$ and $\tilde{A}^{2,2}$, and one zero row to $\tilde{A}^{2,1}$ and $\tilde{A}^{2,2}$; we do the same to $\tilde{B}$. As it turns out, it is most efficient to insert this extra row/column, not at the last entry, but at the last-but-one: previous optimizations already reduced the number of operations involving the last row and column, so zeroing it would not be optimal.

With this choice, we review the various sums introduced in the previous subsection and indicate the possible savings:

- $\mathbf{s}_1(\tilde{A}^{1,1}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}), \quad \mathbf{s}_1(\tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2})$
  no savings are apparent in the first sum. In the second one, one can dismiss all terms where two indices are equal to $m-1$; there are $m-1$ such terms.

- $\mathbf{s}_0(\tilde{A}^{1,2} - \tilde{A}^{1,1} + \tilde{A}^{2,1},\ \tilde{B}^{2,1} + \tilde{B}^{1,2} + \tilde{B}^{1,1},\ \tilde{\mathbf{C}}^{1,1} - \tilde{\mathbf{C}}^{1,2} + \tilde{\mathbf{C}}^{2,1})$
  $\mathbf{s}_0(\tilde{A}^{1,2} + \tilde{A}^{2,1} - \tilde{A}^{2,2},\ \tilde{B}^{2,2} + \tilde{B}^{1,2} + \tilde{B}^{2,1},\ \tilde{\mathbf{C}}^{1,2} + \tilde{\mathbf{C}}^{2,2} - \tilde{\mathbf{C}}^{2,1})$
  no savings are apparent in the first sum. In the second one, one can dismiss the term of index $m-1$.

- $\mathbf{s}_2(\tilde{A}^{1,2}, \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,1},\ -\tilde{A}^{1,1}, \tilde{B}^{1,2}, -\tilde{\mathbf{C}}^{1,2},\quad \tilde{A}^{2,1}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{2,1})$
  $\mathbf{s}_2(\tilde{A}^{1,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{1,2},\quad \tilde{A}^{2,1}, \tilde{B}^{1,2},\quad \tilde{\mathbf{C}}^{2,2},\ -\tilde{A}^{2,2}, \tilde{B}^{2,1}, -\tilde{\mathbf{C}}^{2,1})$
  no savings are apparent in the first sum. In the second one, one can dismiss terms where two indices are equal to $m-1$; there are $3(m-1)$ such terms.

- $-\mathbf{u}_1(\ \tilde{A}^{1,1}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1},\quad \tilde{\mathbf{C}}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}),\ -\mathbf{u}_1(\ \tilde{A}^{1,2}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1})$
  $-\mathbf{u}_1(-\tilde{A}^{1,1}, \tilde{B}^{2,1},\ -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}),\ -\mathbf{u}_1(\ \tilde{A}^{2,1}, \tilde{B}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_1(\ \tilde{A}^{1,2}, \tilde{B}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1}),\ -\mathbf{u}_1(\ \tilde{A}^{2,1}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_1(-\tilde{A}^{2,2}, \tilde{B}^{1,2},\ -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}),\ -\mathbf{u}_1(\ \tilde{A}^{2,2}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2},\quad \tilde{\mathbf{C}}^{2,2},\quad \tilde{\mathbf{C}}^{2,2})$
  the presence of extra rows and columns of zeros reduces the cost of the second and third lines by $m-2$, and by $2(m-2)$ for all lines from the fourth on. In total, we save $12(m-2)$ multiplications.

- $-\mathbf{u}_2'(\ \tilde{A}^{1,1}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}),\ -\mathbf{u}_2(\ \tilde{A}^{1,2}, \tilde{B}^{1,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1})$
  $-\mathbf{u}_2(-\tilde{A}^{1,1}, \tilde{B}^{2,1},\ -\tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}),\ -\mathbf{u}_2(\ \tilde{A}^{2,1}, \tilde{B}^{1,2},\quad \tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,1}, -\tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_2(\ \tilde{A}^{1,2}, \tilde{B}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1}),\ -\mathbf{u}_2(\ \tilde{A}^{2,1}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2}, -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_2(-\tilde{A}^{2,2}, \tilde{B}^{1,2},\ -\tilde{\mathbf{C}}^{2,1},\quad \tilde{\mathbf{C}}^{1,2},\quad \tilde{\mathbf{C}}^{2,2}),\ -\mathbf{u}_2'(\ \tilde{A}^{2,2}, \tilde{B}^{2,2},\quad \tilde{\mathbf{C}}^{2,2})$
  the presence of zeros saves one multiplication at each line, except the first one, for a total of 7.

- $-\mathbf{u}_3(\tilde{A}^{1,1} + \tilde{A}^{1,2}, \tilde{B}^{1,1}, \tilde{\mathbf{C}}^{1,1}),\quad -\mathbf{u}_3(\tilde{A}^{1,1} + \tilde{A}^{1,2}, \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,2})$
  $-\mathbf{u}_3(\tilde{A}^{2,1} + \tilde{A}^{2,2}, \tilde{B}^{1,2}, \tilde{\mathbf{C}}^{2,1}),\quad -\mathbf{u}_3(\tilde{A}^{2,1} + \tilde{A}^{2,2}, \tilde{B}^{2,2}, \tilde{\mathbf{C}}^{2,2})$
  the presence of zeros saves one multiplication at each line, except the first one, for a total of 3.

- $-\mathbf{u}_4(\tilde{A}^{1,1}, \tilde{B}^{1,1} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,1}),\quad -\mathbf{u}_4(\tilde{A}^{1,2}, \tilde{B}^{1,1} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{2,1})$
  $-\mathbf{u}_4(\tilde{A}^{2,1}, \tilde{B}^{2,2} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{1,2}),\quad -\mathbf{u}_4(\tilde{A}^{2,2}, \tilde{B}^{2,2} - \tilde{B}^{2,1}, \tilde{\mathbf{C}}^{2,2})$
  we save 3 multiplications, as in the previous case.

Without savings, the cost reported in the previous section was $(8m^3+24m^2-50m+18)/3$ multiplications. The savings add up to $16m-14$, whence a total of $(8m^3+24m^2-98m+60)/3$ multiplications. To multiply matrices of odd size n, we actually have $m = (n-1)/2+2$, whence a total cost of $(n^3+15n^2+14n-6)/3$.