Matrix multiplication over word-size modular rings using approximate formulae

Brice Boyer, Jean-Guillaume Dumas

▶ To cite this version:

Brice Boyer, Jean-Guillaume Dumas. Matrix multiplication over word-size modular rings using approximate formulae. ACM Transactions on Mathematical Software, 2016, 42 (3-20), 10.1145/2829947 . hal-00987812

HAL Id: hal-00987812
https://hal.science/hal-00987812

Submitted on 6 May 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Matrix multiplication over word-size prime fields using Bini's approximate formula

Brice Boyer* Jean-Guillaume Dumas†

Abstract

Bini's approximate formula (or border rank) for matrix multiplication achieves a better complexity than Strassen's matrix multiplication formula. In this paper, we show a novel way to use the approximate formula in the special case where the ring is $\mathbf{Z}/p\mathbf{Z}$. Besides, we show an implementation à la FFLAS–FFPACK, where p is a word-size prime number, that improves on state-of-the-art $\mathbf{Z}/p\mathbf{Z}$ matrix multiplication implementations.

1 Introduction

A fast reliable matrix multiplication implementation over $\mathbf{Z}/p\mathbf{Z}$ is crucial in exact linear algebra. Indeed, many algorithm rely on fast matrix multiplication as a building block (computation of factorised forms, characteristic polynomial, black box methods,...). But also matrix multiplication on other rings such as the integer ring $\mathbf{Z}$, polynomials $\mathbf{Z}/p\mathbf{Z}[X]$ or Galois fields $\mathbf{F}_q$ reduce to matrix multiplication over $\mathbf{Z}/p\mathbf{Z}$.

In [DGP08], matrix product over $\mathbf{Z}/p\mathbf{Z}$ is computed efficiently with Strassen–Winograd's subcubic matrix multiplication algorithm ([Str69, Win71]). A cascade algorithm is used: first a few recursive steps of Strassen's algorithm are performed, second, below a threshold, numerical BLAS matrix multiplication routines are called. The numerical routines are used in an exact manner and reductions modulo p are delayed as much as possible.

We use similar techniques in this article. Our main contribution is the use of an approximate algorithm to compute the exact matrix multiplication. We first recall Bini's algorithm ([BCLR79]) in Section 2. Then in Section 3, we show two fashions for the application of Bini's algorithm to get an exact matrix multiplication: first via rounding, second via $p-$adic expansion. We finally provide

*North Carolina State University, Department of Mathematics, Raleigh, NC, USA; bbboyer@ncsu.edu. This material is based on work supported in part by the National Science Foundation under Grant CCF-1115772 (Kaltofen).

†Laboratoire J. Kuntzmann, Université de Grenoble. 51, rue des Mathématiques, umr CNRS 5224, bp 53X, F38041 Grenoble, France; Jean-Guillaume.Dumas@imag.fr. This material is based on work supported in part by the Agence Nationale pour la Recherche under Grant ANR-11-BS02-013 HPAC (Dumas).

in Section 4 a new schedule for the algorithm minimizing memory usage and compare in Section 5 with existing exact matrix multiplication implementations.

2 Bini's formula

2.1 Algorithm.

We first recall Bini's approximate formula. Let $A \in \mathbf{K}^{m\times k}$, $B \in \mathbf{K}^{k\times n}$, and let $C = A \times B$ be the matrix product. We consider the special case $(m, k, n) = (3, 2, 2)$ and use a parameter ε. Bini's approximate formula computes a matrix

$$C_\varepsilon = A \times B + \varepsilon D(\varepsilon), \tag{1}$$

where D is a polynomial on $\mathbf{K}^{3\times 2}(\varepsilon)$, with 10 multiplications only. We write a bilinear formula for the multiplication $C = A \times B$:

$$C = \sum_{r=1}^{\mu} \langle A, X_r\rangle \langle B, Y_r\rangle Z_r \tag{2}$$

where $\langle\cdot,\cdot\rangle$ is the Frobenius (term-wise) product and X, Y and Z are given[1] in the Table 1.

We now write Bini's $(3, 2, 2)-$algorithm (Table 2) using[2] Table 1 and Equation (2). We use the following notations troughout the rest of the paper: we divide the matrix A of size $m \times k$ into six submatrices `Aij` (or A_{ij}) of equal size $m/3 \times k/2$, where i and j correspond to row and column indices.

We stress that the formula (2) automatically yields (by duality, [HM73]) a $(2, 2, 3)$ formula by simply considering $(Y_r^\top, X_r^\top, Z_r^\top)$, and a $(2, 3, 2)$ formula using $(Z_r^\top, X_r, Y_r^\top)$. Actually, we use $(\varepsilon Z_r^\top, X_r, {}^1\!/\!_\varepsilon Y_r^\top)$ in our algorithm so that the pre-addition phase contain only scalings by ε, and divisions by ε only occur in the post-addition phase, just like in Table 2. Combining those three formulae yields a $(12, 12, 12)$ square matrix algorithm via the $(12, 12, 12) = (4\cdot3, 6\cdot2, 6\cdot2)$, then $(4, 6, 6) = (2 \cdot 2, 2 \cdot 3, 3 \cdot 2)$ and $(2, 2, 3)$ variants. This algorithm has an $\log_{12}(1000) \approx 2.780$ exponent (smaller than Strassen's $\log_2(7) \approx 2.807$).

We represent in Figure 1 the dependency graph for Bini's $(3, 2, 2)$ algorithm. In this figure, we show scalar division by ε with dotted lines while we draw dashed lines for scalar multiplication by ε. We notice that similarly to Bodrato's algorithm for the Strassen multiplication ([Bod10]), Bini's algorithm has symmetries. These symmetries can be used in the scheduling of the algorithm, taking for instance advantage of independent pre-additions for parallelization, or for possibly saving operations for the squaring in the $(12, 12, 12)$ algorithm.

[1] This table corrects some typos in the matrix $w_r^{(s)}(\varepsilon)$ in [Bin80, eq. (5.2)] (namely signs in Z_4, Z_7 and Z_9).

[2] We have actually permuted the indices $r = 0, \ldots, 9$ in Z_r for the products P_r.

Table 1: Bini's approximate $(3, 2, 2)$ formula.

$$X_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad Y_0 = \begin{bmatrix} \varepsilon & 0 \\ 0 & 1 \end{bmatrix} \quad Z_0 = \begin{bmatrix} 1/\varepsilon & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$X_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad Y_1 = \begin{bmatrix} 0 & 0 \\ -1 & -1 \end{bmatrix} \quad Z_1 = \begin{bmatrix} 1/\varepsilon & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad Y_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad Z_2 = \begin{bmatrix} -1/\varepsilon & -1/\varepsilon \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$X_3 = \begin{bmatrix} 0 & \varepsilon \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad Y_3 = \begin{bmatrix} -\varepsilon & 0 \\ 1 & 0 \end{bmatrix} \quad Z_3 = \begin{bmatrix} 1/\varepsilon & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$X_4 = \begin{bmatrix} 1 & \varepsilon \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad Y_4 = \begin{bmatrix} 0 & \varepsilon \\ 0 & 1 \end{bmatrix} \quad Z_4 = \begin{bmatrix} 0 & 1/\varepsilon \\ 0 & -1 \\ 0 & 0 \end{bmatrix}$$

$$X_5 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y_5 = \begin{bmatrix} 1 & 0 \\ 0 & \varepsilon \end{bmatrix} \quad Z_5 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1/\varepsilon \end{bmatrix}$$

$$X_6 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \quad Y_6 = \begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix} \quad Z_6 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1/\varepsilon \end{bmatrix}$$

$$X_7 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad Y_7 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad Z_7 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1/\varepsilon & -1/\varepsilon \end{bmatrix}$$

$$X_8 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ \varepsilon & 0 \end{bmatrix} \quad Y_8 = \begin{bmatrix} 0 & 1 \\ 0 & -\varepsilon \end{bmatrix} \quad Z_8 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1/\varepsilon \end{bmatrix}$$

$$X_9 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \varepsilon & 1 \end{bmatrix} \quad Y_9 = \begin{bmatrix} 1 & 0 \\ \varepsilon & 0 \end{bmatrix} \quad Z_9 = \begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 1/\varepsilon & 0 \end{bmatrix}$$

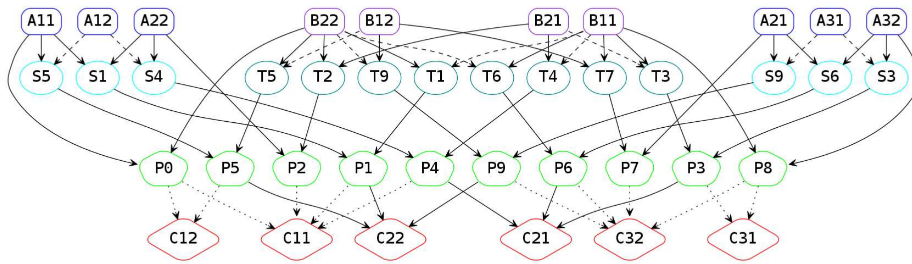

Figure 1: Bini's algorithm dependencies

Table 2: Bini's algorithm

$$
\begin{array}{llll}
S_1 & \leftarrow A_{11} + A_{22} & T_1 & \leftarrow B_{22} + \varepsilon \cdot B_{11} \\
 & & T_2 & \leftarrow B_{21} + B_{22} \\
S_3 & \leftarrow A_{32} + \varepsilon \cdot A_{31} & T_3 & \leftarrow B_{11} + \varepsilon \cdot B_{21} \\
S_4 & \leftarrow A_{22} + \varepsilon \cdot A_{12} & T_4 & \leftarrow B_{21} - \varepsilon \cdot B_{11} \\
S_5 & \leftarrow A_{11} + \varepsilon \cdot A_{12} & T_5 & \leftarrow B_{22} + \varepsilon \cdot B_{12} \\
S_6 & \leftarrow A_{21} + A_{32} & T_6 & \leftarrow B_{11} + \varepsilon \cdot B_{22} \\
 & & T_7 & \leftarrow B_{11} + B_{12} \\
S_9 & \leftarrow A_{21} + \varepsilon \cdot A_{31} & T_9 & \leftarrow B_{12} - \varepsilon \cdot B_{22} \\
P_0 & \leftarrow A_{11} \times B_{22} & P_1 & \leftarrow S_1 \times T_1 \\
P_2 & \leftarrow A_{22} \times T_2 & P_3 & \leftarrow S_3 \times T_3 \\
P_4 & \leftarrow S_4 \times T_4 & P_5 & \leftarrow S_5 \times T_5 \\
P_6 & \leftarrow S_6 \times T_6 & P_7 & \leftarrow A_{21} \times T_7 \\
P_8 & \leftarrow A_{32} \times B_{11} & P_9 & \leftarrow S_9 \times T_9 \\
C_{11} & \leftarrow (P_1 - P_2 + P_4 - P_0)/\varepsilon & C_{12} & \leftarrow (P_5 - P_0)/\varepsilon \\
C_{21} & \leftarrow P_4 - P_3 + P_6 & C_{22} & \leftarrow P_1 - P_5 + P_9 \\
C_{31} & \leftarrow (P_3 - P_8)/\varepsilon & C_{32} & \leftarrow (P_6 - P_7 + P_9 - P_8)/\varepsilon
\end{array}
$$

2.2 Complexity.

We give an idea of the number of operations in Strassen–Winograd's and Bini's algorithm for one level of recursion only. On the one hand, we have $W(m,k,n) = 7W(m/2, k/2, n/2) + 4m/2k/2 + 4k/2n/2 + 7m/2n/2$, that is 7 multiplications, 4 pre-additions on each side, and 7 post-additions, see [Win71] and [BDPZ09, Alg. 1]). On the other hand, one has $B(m,k,n) = 10W(m/3, k/2, n/2) + 10m/3k/2 + 14k/2n/2 + 16m/3n/2$, that is 10 multiplications, 10 additions or scalings on the left side, 14 additions or scalings on the right, and 16 final additions or scalings by ε.

We write the relative change $\frac{B(6m,6k,6n)-W(6m,6k,6n)}{W(6m,6k,6n)} \approx -\frac{1}{21}$. For $6m = 6k = 6n = 3\,000$, and one level of recursion, one would get $\approx -4.6\%$. We will not consider two levels of recursions of Bini's algorithm, for reasons that will appear later, so the second level will be a call to Strassen–Winograd's algorithm. In that case, the relative change will be $\approx -4.5\%$.

3 Application to exact matrix multiplication

We first recall the transformation in [Bin80] from an approximate formula (Equation (1)) to and exact algorithm $C = A \times B$: it consists in finding a special set of $d+1$ scalars α_i, where $d = \deg_\varepsilon(\varepsilon D(\varepsilon))$, and $d+1$ pair-wise distinct scalars ε_i. Then it is possible (invertible Vandermonde matrix in ε_i^j) to make sure that $\sum_{i=1}^{d+1} \alpha_i = 1$ and for $j = 1, \ldots, d$ that $\sum_{i=1}^{d+1} \alpha_i \varepsilon_i^j = 0$. Then

$\sum_{i=1}^{d+1} \alpha_i C_{\varepsilon_i}$ breaks into $\sum_{i=1}^{d+1} \alpha_i A \times B + \sum_{i=1}^{d+1} \alpha_i \varepsilon_i D(\varepsilon_i)$. The first part reduces to $A \times B$, while the second one gives 0 (after switching the summation operand between $\sum_{i=1}^{d+1}$ and $\varepsilon_i D(\varepsilon_i) = \sum_{j=1}^{d} d_j \varepsilon_i^j$).

It is not practicable, at least for the size of matrices we consider, to perform three $\varepsilon-$approximate multiplication while competing against Strassen–Winograd's algorithm implementation, as exemplified in our experiments of Table 6: we get only 5-10% speed-up for one call to the approximate algorithm compared to one call of Strassen–Winograd's. However, we apply a different method requiring only one call to the approximate multiplication, for the special case where the field is $\mathbf{Z}/p\mathbf{Z}$.

We have used this approximate algorithm in the following two cases, both for a storage on `double` floating point machine words. First, we consider the case $\varepsilon = 2^{-27}$ (Section 3.1), where the idea is to store two exact integers in one `double` as $x + \varepsilon y$. Then any term in ε^2 will be neglected, as ε^2 approaches the machine precision, and a final rounding will remove the $\varepsilon-$approximations. Then in Section 3.2, we take $\varepsilon = p$ within the numerical routines and the $p-$approximations are removed by a final reduction modulo p.

3.1 Case $\varepsilon = 2^{-27}$.

We recover the correct result by rounding to the nearest integer, but we need to make sure that the error is not too large and that no exact digit is lost during the approximate computations. We consider one level of recursion.

We need to loose no digit in the computation of the exact part C. In the upper and lower band of of matrix C (*i.e.* submatrices `C1*` and `C3*`) in Table 2, we need that the ε part of the P_i's product is correct. In the central band, we need the integral part to be correct. More precisely, we need to make sure that:

- No digit is lost in the ε part of the computation of $P_3 = (A_{32} + \varepsilon \cdot A_{31}) \times (B_{11} + \varepsilon \cdot B_{21})$ (from examination of all the products one can see that P_3 is the worst case);
- No digit is lost during the post addition phase
- The residual errors vanish when rounded to the nearest.

We represent elements modulo p in the standard representation $\{0, \ldots, p-1\}$. Let $M = \lfloor k/2 \rfloor (p-1)^2$ be the largest element that can occur in a product of A_{ij} by B_{kl}. This is a dot product between $\lfloor k/2 \rfloor-$long rows and columns of a submatrix of A and B whose entries are bounded by $p-1$, in the case where k is not even, we would take the floor of $k/2$ and apply pealing techniques. We then consider the number $\lambda = M + 2M\varepsilon + M\varepsilon^2$ that is the largest possible entry, occurring for instance in P_3, the number $\lambda_0 = M + 2M\varepsilon$ which is of interest and the error $\lambda_1 = M\varepsilon^2$. If M fills 27 bits, then λ_0 will use all the bits in its `double` representation. Indeed, for $M = 2^{27} - 1$, one has $\lambda_0 = 2^{27}\left(1 + 2^{-27} - 2^{-53}\right)$, which fills all the bits in the mantissa of a `double`). Then, λ_1 cannot change any bit (the exponent in λ_0 is 27 so 2^{-54} is below precision). For any smaller value

of M, the error term λ_1 will of course not affect λ_0. Now, all the digits in P_3 that need to be correct are correct, and so is the middle band of C. The same reasoning applies for the other entries. We now require that $\lfloor k/2 \rfloor (p-1)^2 = M < 2^{27}$.

Now, we consider the approximation error. For one recursion level, the components of the error polynomial is either of the form $D_{1,1}(\varepsilon) = -A_{12} \times B_{11}$, or the form $D_{2,1}(\varepsilon) = -\varepsilon A_{12} \times B_{11} - \varepsilon A_{31} \times B_{21} + A_{12} \times B_{21} + A_{21} \times B_{22} - A_{22} \times B_{11} - A_{31} \times B_{11} - A_{32} \times B_{21} + A_{32} \times B_{22}$.

In fact, the term in ε^2 disappears as explained previously. We examine carefully the coefficient of degree ε in the $\varepsilon D_{2,1}$ expression: the terms A_{12}, A_{21} and B_{22} are only involved additively and the term B_{11} only appears with a $-$. If we want to bound by above, we may assume the first ones are $p-1$ and the second one is 0. There remains $-A_{32}B_{21} + (A_{32} + B_{21})(p-1) + \varepsilon(p-1)^2$. This expression is maximum when $A_{32} = B_{21} = (p-1)/2$. So the polynomial $D(\varepsilon)$, for $\varepsilon > 0$, is bounded: $D(\varepsilon) \leqslant \frac{7}{4} \lfloor k/2 \rfloor (p-1)^2$. The same arguments give a similar bound from below: $D(\varepsilon) \geqslant -3\lfloor k/2 \rfloor (p-1)^2$ For our rounding purposes, we require that $\|\varepsilon D(\varepsilon)\|_\infty < \frac{1}{2}$, so we require $\lfloor k/2 \rfloor (p-1)^2 = M < \frac{1}{6\varepsilon} = \frac{1}{6} 2^{27}$.

We have thus proved the following:

Proposition 1. *For $\varepsilon = 2^{-27}$ and an (m, k, n) matrix multiplication over a ring $\mathbf{Z}/p\mathbf{Z}$, the rounding to the nearest integer of the output C_ε in Equation (1) of one call to Bini's $(3, 2, 2)-$approximate algorithm with `double` floating point arithmetic, gives the exact result C, provided that:*

$$2\lfloor k/2 \rfloor (p-1)^2 < \frac{1}{3} 2^{27}.$$

Remark 2. These bounds are tight. Indeed, the proof gives explicit matrices A and B, for instance $A^\top = \begin{pmatrix} * & * & p-1 \\ 0 & p-1 & p-1 \end{pmatrix}$ and $B = \begin{pmatrix} p-1 & * \\ p-1 & 0 \end{pmatrix}$, or in the balanced representation, if we denote $p_m = \frac{p-1}{2}$, $A^\top = \begin{pmatrix} * & p_m & p_m \\ -p_m & p_m & p_m \end{pmatrix}$ and $B = \begin{pmatrix} p_m & * \\ p_m & -p_m \end{pmatrix}$.

Remark 3. If we use a balanced representation (*i.e.* where an element modulo an odd p is represented between $\frac{1-p}{2}$ and $\frac{p-1}{2}$), we can prove that the bound is reduced to $\|D(\varepsilon)\|_\infty \leqslant \frac{3}{2} \lfloor k/2 \rfloor (p-1)^2$, thus gaining in Proposition 1:

$$\lfloor k/2 \rfloor (p-1)^2 < \frac{1}{3} 2^{27}.$$

3.2 Case $\varepsilon = p$.

We consider again one level of recursion. The elements S_i for all i and T_i for $i \neq 4, 9$ are $\geqslant 0$ and $\leqslant 2(p-1)$ or $\leqslant (p-1) + \varepsilon(p-1) = (p-1)(p+1)$. Elements T_4 and T_9 are $\geqslant -p(p-1)$ and $\leqslant p-1$. A careful examination of the coefficients in C, before any division by ε, shows that the elements are bounded by $\pm\lfloor k/2 \rfloor (p-1)^2 (p+1)^2$.

Finally, our algorithm clearly computes an exact multiplication $C_\varepsilon = A \times B$ since the remainder is $pD(p) \equiv 0 \pmod p$. We have then proved:

Proposition 4. *For $\varepsilon = p$ and an (m, k, n) matrix multiplication over a ring $\mathbf{Z}/p\mathbf{Z}$, the reduction modulo p of the output C_ε in Equation (1) of one call to Bini's $(3, 2, 2)-$approximate algorithm with `double` floating point arithmetic, gives the exact result C, provided that:*

$$\lfloor k/2 \rfloor (p-1)^2 (p+1)^2 < 2^{53}.$$

Remark 5. With a balanced representation and an odd prime, a careful examination of the bounds then gives:

$$1/2 \lfloor k/2 \rfloor (p-1)^2 p(p+1) < 2^{53}.$$

4 Memory usage and scheduling

In this section, we provide schedules for Bini's approximate multiplication, in a similar fashion to [BDPZ09]. These schedules are then implemented. We try to require as little extra memory (temporaries) as possible.

4.1 Scheduling for the $(3, 2, 2)$ multiplication.

We created the next schedule with only two temporaries (Table 3). In that table, we write `e` for the parameter ε. A star (`*`) represents a matrix multiplication while a dot (`.`) represents a scalar multiplication. The first column represents the operation number, the third column the algorithmic variable (*cf.* Table 2).

The extra memory used by X is $m/3 \times k/2$ and the extra memory occupied by Y is $k/2 \times n/2$ except for the last but one (step 34) where actually $m/3 \times n/2$ is needed instead. But, at step 34, X is also usable so $A_{32}B_{11}$ can be stored in a combination of X and Y if $k/2 \times (m/3 + n/2) \geqslant m/3 \times n/2$. We have proven:

Lemma 1. *The extra memory used for one level of recursion in Table 3 is* $\max(m/3 \times n/2, (m/3 + n/2) \times k/2)$.

This requirement is smaller than Strassen–Winograd's (*cf.* [BDPZ09]) where X is of size $m/2 \times \max(k/2, n/2)$ and Y has size $k/2 \times n/2$. For instance, for $m = n = k$, we have $\frac{5}{12}m^2$ for Bini's and $\frac{1}{2}m^2$ for Strassen–Winograd's.

Lemma 2. *Two temporaries are a minimum in Table 3.*

algorithm (see [BDPZ09, HLJJ+96]). This can also be proven "by hand", considering a finished pebble game on Figure 1 and trying to use only one extra pebble only for the previous moves until no 'previous' move is possible.

Remark 6. There is *a priori* no clear reason whether we should put more effort into reducing the number of operations, the number of allocations or a balance between the two. For instance, in the $(2, 3, 2)$ algorithm, elements `A12` and `A22`, are only involved with a product with ε, and three times each; the same happens for `B`. One could save operations by creating a temporary `eA12 := e .  A12` and reuse it three times. Our implementations show that recomputing does not affect negatively the timings.

Table 3: Schedule for Bini's algorithm using only two temporaries

#	operation	var	#	operation	var
1	C11 := A11 * B22	P0	19	Y := B12 - e . B22	T9
2	X := A11 + e . A12	S5	20	C32 := X * Y	P9
3	Y := e . B12 + B22	T5	21	C22 := C22 + C32	C22
4	C22 := X * Y	P5	22	X := A21 + A32	S6
5	C12 := (C22 - C11)/e	C12	23	Y := B11 + e . B22	T6
6	Y := B21 + B22	T2	24	C31 := X * Y	P6
7	C31 := A22 * Y	P2	25	C21 := C21 + C31	C21
8	C11 := C11 + C31	C11	26	C32 := C32 + C31	C32
9	X := A11 + A22	S1	27	Y := B11 + B12	T7
10	Y := e . B11 + B22	T1	28	C31 := A21 * Y	P7
11	C21 := X * Y	P1	29	C32 := C32 - C31	C32
12	C22 := C21 - C22	C22	30	X := e . A31 + A32	S3
13	C11 := C21 - C11	C11	31	Y := B11 + e . B21	T3
14	X := e . A12 + A22	S4	32	C31 := X * Y	P3
15	Y := B21 - e . B11	T4	33	C21 := C21 - C31	C21
16	C21 := X * Y	P4	34	Y := A32 * B11	P8
17	C11 := (C21 + C11)/e	C11	35	C31 := (C31 - Y)/e	C31
18	X := A21 + e . A31	S9	36	C32 := (C32 - Y)/e	C32

4.2 Schedules for other shapes and other properties

We have implemented schedules for the $(2, 2, 3)$ shape (essentially the same as Table 3 up to exchanging `Aij` with `Bji`, and `Cij` with `Cji`) and for the $(2, 3, 2)$ shape. In the later case, we created 4 temporaries in our implementation.

Just as in [BDPZ09], we can allow the overwriting of (part of) A and/or B. The following Table 4 shows how to perform the multiplication with no extra memory, overwriting a third of matrix `A` (namely `A11` and `A12`). A similar in-place schedule overwriting `B` could be written up. The schedule in Table 4 has strong requirements on the sizes of the inputs, for instance, the size of a submatrix of `C` needs to be equal to one of `A` (operation `# 2` and `# 34`), larger than one of `B` (`# 3`), and the size of a submatrix of `A` needs to be larger that that of one of `B` (`# 14`). In order to relax these requirements, one can write an in-place schedule that overwrites parts of `A` and `B`. This phenomenon is common with [BDPZ09].

Finally, we could also imagine schedules for the product with accumulation $C \leftarrow \alpha A \times B + \beta C$ that require less memory than the naive approach, and may save operations.

Table 4: Schedule for Bini's algorithm, inplace, overwriting a third of A.

#	operation	var
1	`C11 := A11 * B22`	`P0`
2	`C21 := A11 + e . A12`	`S5`
3	`C32 := e . B12 + B22`	`T5`
4	`C22 := X * Y`	`P5`
5	`C12 := (C22 - C11)/e`	`C12`
6	`C21 := B21 + B22`	`T2`
7	`C31 := A22 * Y`	`P2`
8	`C11 := C11 + C31`	`C11`
9	`A11 := A11 + A22`	`S1`
10	`C32 := e . B11 + B22`	`T1`
11	`C21 := X * Y`	`P1`
12	`C22 := C21 - C22`	`C22`
13	`C11 := C21 - C11`	`C11`
14	`A12 := e . A12 + A22`	`S4`
15	`C32 := B21 - e . B11`	`T4`
16-36	Same as Table 3, replacing `X:=` with `A11:=` and `Y:=` with `A12:=`.	

5 Implementation

We implemented in the FFLAS–FFPACK[3] (*cf.* [DGP08, BDPZ09]) library the algorithm corresponding to the schedule in Table 3. Only one level of recursion is performed, which means that the matrix multiplication within the algorithm are calls to the implementation of Strassen–Winograd's algorithm, that is the `fgemm` routine of FFLAS.

5.1 Achievable size of p.

First of all, according to Propositions 1 and 4, we can take moduli as large as those reported in the Table 5:

Table 5: Largest moduli for delayed reduction in Bini's algorithm

	$\varepsilon = 2^{-27}$		$\varepsilon = p$	
size	Modular	Balanced	Modular	Balanced
1 000	212	599	2 060	2 450
2 000	150	424	1 732	2 060
3 000	123	346	1 565	1 861
4 000	106	300	1 456	1 732

[3]See http://linalg.org/projects/fflas-ffpack/.

Then, we want to compare against the implementation of Winograd's algorithm in the FFLAS–FFPACK library. Our implementation uses the `double` type only, but we need to compete against the `float` and `double` implementation of Winograd's algorithm in FFLAS. Indeed, when changed form `double` to `float`, our bounds in Propositions 1 and 4 would employ 2^{12} instead of 2^{27} and (resp.) 2^{24} instead of 2^{53}, but the bounds linking k and p would be too restrictive. Hence the implementation on `double` type only. However, the library FFLAS provides a very efficient matrix multiplication implementation over $\mathbf{Z}/p\mathbf{Z}$ for small p taking advantage of a `float` representation, since BLAS `sgemm` routine is close to twice faster than `dgemm`. Therefore, we compare to both the `float` and `double` implementations of FFLAS.

One can find precise bounds linking the prime p, dimension k, the number of recursive calls l, the floating point representation, and the field representation in [DGP08]. For instance, with $k = 3\,000$ and one level of recursion on a `float` type, one can perform no intermediate modular reduction for $p < 39$ (full delayed reduction), which is rather small. When p is larger and modular reductions cannot be delayed, FFLAS on `float` then uses smaller, more numerous, BLAS blocks and performs more reductions. This can still prove to be faster than FFLAS on `double`, because we get the benefit from faster BLAS on `float` type. So there is a trade-off between more reductions and smaller blocking with faster BLAS on `float` and fewer reductions and larger blocking with slower BLAS of `double`.

5.2 Timings.

The timings presented in Table 6 were performed on a `x86_64` Gentoo Linux laptop with a 2,3GHz Intel® Core™ i7 and 6Gb of RAM, using atlas-3.10.1 BLAS, averaging on four runs (in seconds). The compilation was done with `g++-4.8.2` enabling `-Ofast` and AVX extensions (support introduced in the latest SVN-1.8.0 version of FFLAS–FFPACK). The first column represents the (m, k, n) dimensions of matrices, in thousands; or simply (n) when all dimensions are equal. The entries are created randomly in the ring $\mathbf{Z}/p\mathbf{Z}$. The second column is the modulo used for the multiplication (not necessarily prime). The third and fourth column contain our matrix multiplication algorithm from Section 3.1 and (resp.) Section 3.2. The symbol † signifies that the $(2, 3, 2)$ variant was benchmarked. Then, the following two columns are the reference implementation of Winograd algorithm in FFLAS, using the `double` and `float` types. Finally, the relative change (in percent) is computed as the ratio $(t_b - t_w)/t_w \cdot 100$ where t_b is this paper implementation's timings and t_w if the best of FFLAS reference implementations. The self optimised FFLAS threshold for `double` was $1\,000$, while it was $1\,640$ for `float`. This threshold essentially determines when Strassen–Winograd's algorithm takes advantage over the naïve (BLAS) algorithm on a given representation (`float` or `double`). We did exactly one recursive level for Bini's algorithm (hence no threshold there). The symbol "n/a" (not available) corresponds entries in the table that cannot be filled, *i.e.* the corresponding algorithm could not be implemented because the modulo p is too large. The

columns titled 'M' correspond to a standard modular representation of a field, while 'MB' refers to a balanced representation. The boldface for timings emphasizes the best time for a modular (M) and balanced (MB) representation among all tested algorithms for a given row.

Table 6: Timings (s) of Bini's algorithm *vs.* Winograd's (`fgemm`) on $\mathbf{Z}/p\mathbf{Z}$ using `Modular` and `ModularBalanced` representation.

dimensions (×1 000)	p	Sec. 3.1 $\varepsilon = 2^{-27}$		Sec. 3.2 $\varepsilon = p$		FFLAS `double`		FFLAS `float`		rel. change (%)	
		M	MB	M	MB	M	MB	M	MB	M	MB
(1.5)	141	0.31	0.31	0.31	0.31	0.32	0.32	**0.25**	**0.25**	+21.4	+22.2
(1.5)	451	n/a	**0.31**	**0.31**	0.31	0.32	0.32	0.32	0.32	−2.99	−3.28
(1.5)	1 001	n/a	n/a	**0.31**	**0.31**	0.32	0.32	0.41	0.43	−2.95	−2.92
(1.5)	1 501	n/a	n/a	**0.31**	**0.31**	0.32	0.32	1.50	1.52	−2.76	−2.98
(2.1)	1 001	n/a	n/a	**0.81**	**0.82**	0.85	0.85	1.07	1.10	−5.64	−4.60
(2.1)	1 501	n/a	n/a	**0.80**	**0.82**	0.86	0.86	4.33	4.37	−5.77	−4.62
(2.7)	1 001	n/a	n/a	**1.70**	**1.72**	2.03	2.02	2.42	2.43	−16.3	−15.0
(2.7)	1 501	n/a	n/a	**1.71**	**1.71**	2.03	2.03	9.25	9.25	−15.9	−16.0
(3.3)	1 001	n/a	n/a	**3.08**	**3.08**	3.47	3.47	4.48	4.47	−11.1	−11.0
(3.3)	1 501	n/a	n/a	**3.09**	**3.09**	3.46	3.45	17.0	16.8	−10.7	−10.6
(3.9)	1 001	n/a	n/a	**4.94**	**4.97**	5.39	5.54	6.89	6.96	−8.34	−10.5
(3.9)	1 501	n/a	n/a	n/a	**4.94**	**5.39**	5.51	27.8	28.1	n/a	−10.4
(3.0, 2.7, 2.7)	1 001	n/a	n/a	**1.88**	**1.89**	2.00	2.00	2.75	2.71	−6.02	−5.62
(2.7, 3.0, 2.7)	1 001	n/a	n/a	**1.83**	**1.85**	2.16	2.16	2.69	2.73	−15.4	−14.7
(2.7, 2.7, 3.0)	1 001	n/a	n/a	**1.86**†	**1.87**†	2.26	2.25	2.73	2.76	−17.4	−16.9
(3.6, 2.7, 2.7)	1 001	n/a	n/a	**2.26**	**2.28**	2.39	2.38	3.18	3.15	−5.34	−4.43
(2.7, 3.6, 2.7)	1 001	n/a	n/a	**2.20**	**2.20**	2.55	2.55	3.16	3.18	−14.0	−13.6
(2.7, 2.7, 3.6)	1 001	n/a	n/a	**2.23**†	**2.22**†	2.65	2.65	3.15	3.14	−16.0	−16.2
(4.2, 2.7, 2.7)	1 001	n/a	n/a	**2.62**	**2.63**	2.76	2.76	3.68	3.64	−5.11	−4.65
(2.7, 4.2, 2.7)	1 001	n/a	n/a	**2.54**	**2.59**	2.98	2.98	3.79	3.68	−14.4	−13.6
(2.7, 2.7, 4.2)	1 001	n/a	n/a	**2.58**†	**2.59**†	3.08	3.09	3.71	3.70	−16.1	−16.1
(2.7, 3.0, 3.0)	1 001	n/a	n/a	**2.04**	**2.05**	2.39	2.39	2.97	3.01	−14.5	−14.2
(3.0, 2.7, 3.0)	1 001	n/a	n/a	**2.04**†	**2.06**†	2.17	2.17	2.98	3.01	−5.72	−5.03
(3.0, 3.0, 2.7)	1 001	n/a	n/a	**2.04**	**2.05**	2.09	2.09	2.95	2.99	−2.43	−2.31
(2.7, 3.6, 3.6)	1 001	n/a	n/a	**2.44**	**2.44**	2.80	2.81	3.55	3.56	−12.8	−13.1
(3.0, 2.7, 3.6)	1 001	n/a	n/a	**2.46**†	**2.44**†	2.60	2.60	3.52	3.48	−5.13	−5.89
(3.6, 3.6, 2.7)	1 001	n/a	n/a	**2.42**	**2.44**	2.50	2.50	3.47	3.47	−2.79	−2.16

6 Discussion

The timings show that our implementation is competitive with Winograd's algorithm implementation, usually providing an $\approx 5\%$ speed-up, and it is always faster than FFLAS using a `double` representation. As expected, for small primes, the `float` representation performs better, but this phenomenon is only relevant for small moduli (≈ 400 and less). A general fast efficient implementation of

matrix multiplication needs to consider a threshold (via automatic benchmarking, *cf.* [BDG+14]) below which switch to `float` representation is preferable (if memory allows). This is not done yet in FFLAS. The balanced representation allows to gain an $\approx 10\%$ speed-up on size 3 900 where the standard representation could not be used. The best speed-up of $\approx 15\%$ around sizes 2 700 to 3300 could be explained by optimal size BLAS block calls.

The ability to adapt to the sizes for the recursion (one may choose to divide m or n or k by 3 instead of 2) makes it possible to always perform better than `fgemm`; that would not be the case had we not implemented the $(2, 3, 2)$ shape. We notice that timings are almost identical for a constant product mnk, which is not the case for `fgemm`. Therefore, our "triple cascading" (Bini+Winograd+BLAS) improves on the standard cascading (Winograd+BLAS) in a generic fashion: we automatically get better speed-up by pluging in a new algorithm that performs better on some domain of its parameters, and that can plug-in (recursively) the best available routines.

We may also gain speed-ups for larger k and p than those described in Table 5 by first doing a few recursive levels of Winograd's algorithm and finishing by the algorithm discussed in this paper, when the dimension k becomes small enough in a recursive step. This would be a (Winograd+Bini+(Winograd+BLAS)) cascade.

References

[BCLR79] D. Bini, M. Capovani, G. Lotti, and F. Romani. $O(n^{2.7799})$ complexity for matrix multiplication. *Information Processing Letters*, 8:234–235, 1979.

[BDG+14] Brice Boyer, Jean-Guillaume Dumas, Pascal Giorgi, Clément Pernet, and B. David Saunders. Principles of design for containers and solutions in the LinBox library. Abstract accepted for ICMS 2014, August 2014.

[BDPZ09] Brice Boyer, Jean-Guillaume Dumas, Clément Pernet, and Wei 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, pages 55–62, New York, NY, USA, 2009. ACM.

[Bin80] D. Bini. Relations between exact and approximate bilinear algorithms. applications. *Calcolo*, 17:87–97, 1980. 10.1007/BF02575865.

[Bod10] Marco Bodrato. A Strassen-like matrix multiplication suited for squaring and higher power computation. In Stephen M. Watt, editor, *ISSAC 2010: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, pages 273–280. ACM, July 2010.

[DGP08] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages. *ACM Trans. Math. Softw.*, 35(3):1–42, 2008.

[HLJJ+96] Steven Huss-Lederman, Elaine M. Jacobson, Jeremy R. Johnson, Anna Tsao, and Thomas Turnbull. Strassen's algorithm for matrix multiplication : Modeling analysis, and implementation. Technical report, Center for Computing Sciences, November 1996. CCS-TR-96-17.

[HM73] J. Hopcroft and J. Musinski. Duality applied to the complexity of matrix multiplication. *SIAM J. COMPUT*, 2:159–173, 1973.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[Win71] Shmuel Winograd. On multiplication of 2x2 matrices. *Linear Algebra and Application*, 4:381–388, 1971.