A Strassen-like Matrix Multiplication Suited for Squaring and Higher Power Computation [*]

Marco Bodrato

bodrato@mail.dm.unipi.it
http://bodrato.it/papers/

ABSTRACT

Strassen's method is not the asymptotically fastest known matrix multiplication algorithm, but it is the most widely used for large matrices. Since his manuscript was published, a number of variants have been proposed with different addition complexities. Here we describe a new one. The new variant is at least as good as those already known for simple matrix multiplication, but can save operations either for chain products or for squaring. Moreover it can be proved optimal for these tasks. The largest saving is shown for n^{th}-power computation, in this scenario the additive complexity can be halved, with respect to original Strassen's.

Categories and Subject Descriptors

F.2.1 [**Analysis of algorithms and program complexity**]: Numerical algorithms and problems—*Computations on matrices*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.2.3 [**Discrete mathematics**]: Applications; I.1.2 [**Computing methodologies**]: Algorithms—*Algebraic algorithms*

General Terms

Algorithms, Performance, Theory.

Keywords

Polynomial matrix, exponentiation, optimal squaring, fast multiplication, recursive algorithm.

1. INTRODUCTION

When this work started, the main goal was to find a way to speed up the evaluation of a polynomial on a matrix. We started from the first monomial that needs some non-trivial computation: x^2.

[*]This is the author's version, it can be slightly different with respect to the officially published one.

To speed up this simple operation, we analysed Strassen's method [19] for matrix multiplication: it is the most widely used, even if is not the asymptotically fastest known. Every possible Strassen-like algorithm uses the same number of multiplications: they differ on the number of additions and subtractions. Winograd's variant [21] requires the minimal use of linear operations: 15 for a 2×2 matrix. None of the algorithms proposed so far did try to minimise operations for squaring.

The new sequence we propose in §2.2, is basically equivalent to Winograd's variant, with a simple additional property: symmetry. All the results in this paper derive from this symmetry, directly or as a side effect.

The direct result is optimality for squaring, because we can exploit the obvious symmetry of multiplying a single operand by itself. The main side effect is a reduction of the number of linear combinations needed either for the chain product of three or more matrices, or required by n^{th}-power computation or even by more general polynomials on matrices, the initial goal.

While matrix product and Strassen's algorithm can be applied to rectangular matrices, this paper focuses on square matrices, because only for this kind of matrices do squaring and higher power computation make sense. Nevertheless the new proposed sequence can be extended to any multiplication using standard techniques [6, 10].

To obtain formulas valid on every ring we started searching combinations valid for boolean matrices, on $\mathbb{F}_2$. Then we extended the sequences found this way to characteristic 0, testing all possible liftings of the same formulas where $1 \in \mathbb{F}_2$ was lifted to $\pm 1 \in \mathbb{Z}$. Since the obtained algorithms require only additions, subtractions and (non-commutative) multiplications, they will work on any ring. In particular they will work on the algebra of matrices, and can be used recursively.

Applications

Strassen's and Strassen-like methods for matrix multiplication are considered numerically unstable [2]; although some corrections are possible, the main application for a fast matrix multiplication algorithm is that of matrices on exact rings or algebras. We mean matrices over finite fields, integers, rationals, polynomials and so on.

Many algebraic algorithms in graph theory work by building a matrix somehow representing the graph, then computing some power of it, or repeatedly squaring it [9]. The best-known result is: check elements on the diagonal of the k^{th} power of the adjacency matrix to count closed k-walks in a graph. Adjacency matrices of graphs usually have boolean or integer entries, but also different definitions of the adjacency [18] need an algebra where exact computations are possible and Strassen-like methods can be applied.

Another direct application of matrix exponentiation can be cryptography [17, 15]. Moreover the reduced number of operations for chain products can be used in a lot of different contexts, e.g. for number theoretical algorithms where small 2×2 matrices with huge integers are multiplied, like the half-GCD [14] or fast Jacobi symbol computation [4].

2. MATRIX PRODUCT

Given two $d \times d$ matrices A, B, computation of the product $C = AB$, with a naïve implementation, directly applying the definition $C_{ij} = \sum_k A_{ik}B_{kj}$, requires d^3 multiplications and $d^3 - d^2$ additions.

In particular we will study the 2×2 case, so we will have:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{1}$$

In the Appendix §Q only, we will briefly discuss the commutative case. For the rest of the paper we will consider the non-commutative scenario, because for big $2k \times 2k$ matrices the A_{ij}, B_{ij} are not ring elements, but $k \times k$ matrices themselves.

2.1 Strassen-like Algorithms

The naïve algorithm requires $\Theta(d^3)$ operations, but asymptotically faster algorithms exist. The first one, proposed in 1969 by Strassen [19], has the lower complexity $\Theta(d^{\log_2 7}) = O(d^{2.8074})$. This is not the "fastest" known, but it's the most widely used, because asymptotically faster algorithms are efficient for very huge matrices only.

The basic idea consists in an optimisation for the product of 2×2 matrices, requiring only 7 instead of 8 multiplications. When used recursively it gives the product of $2^k \times 2^k$ matrices using 7^k ring multiplications, instead of $2^{3k} = 8^k$.

When applied to 2×2 split matrices, Strassen method requires also 18 linear operations (additions/subtractions) on the sub-matrices.

2.2 New Proposed Sequence

We propose here a new variant, requiring the minimum number of operations, and with an additional property: it is optimal for squaring too and can save operations for chain (three or more matrices) products. We start by describing it for the simple product.

At first four linear pre-combinations are required for each one of the two operands:

$$\begin{array}{ll} (2.s) & \left\{ \begin{array}{l} S_1 = A_{22} + A_{12} \\ S_2 = A_{22} - A_{21} \\ S_3 = S_2 + A_{12} = A_{22} - A_{21} + A_{12} \\ S_4 = S_3 - A_{11} = A_{22} - A_{21} + A_{12} - A_{11} \end{array} \right. \\ (2.t) & \left\{ \begin{array}{l} T_1 = B_{22} + B_{12} \\ T_2 = B_{22} - B_{21} \\ T_3 = T_2 + B_{12} \\ T_4 = T_3 - B_{11} \end{array} \right. \end{array} \tag{2}$$

then the seven products and the final post-combinations:

$$(3.p) \left\{ \begin{array}{lcl} P_1 &=& S_1T_1 \\ P_2 &=& S_2T_2 \\ P_3 &=& S_3T_3 \\ P_4 &=& A_{11}B_{11} \\ P_5 &=& A_{12}B_{21} \\ P_6 &=& S_4B_{12} \\ P_7 &=& A_{21}T_4 \end{array} \right. \quad (3.c) \left\{ \begin{array}{lcl} U_1 &=& P_3 + P_5 \\ U_2 &=& P_1 - U_1 \\ U_3 &=& U_1 - P_2 \\ C_{11} &=& P_4 + P_5 \\ C_{12} &=& U_3 - P_6 \\ C_{21} &=& U_2 - P_7 \\ C_{22} &=& P_2 + U_2 \end{array} \right. \tag{3}$$

The eight inputs A_{ij}, B_{ij} and the four outputs C_{ij} satisfy equation (1).

The proposed method above requires 7 multiplications and $4 + 4 + 7 = 15$ linear combinations; exactly the same as the Winograd's variant, this was proved the best possible [16].

The sequence was found with a computer-aided search within all possible linear combinations in $\mathbb{F}_2$, with one condition: the preparation phases (2.s) and (2.t) should be the same. Only 6 good combinations were found (Strassen's is one of them). The one chosen here is the best one for squaring. There is only one equivalent sequence, which can be obtained from the above by swapping $X_{11} \leftrightarrow X_{22}$ and $X_{12} \leftrightarrow X_{21}$ for all the three matrices A, B, and C.

Then all the liftings of the sequence in $\mathbb{F}_2$ up to $\mathbb{Z}$, lifting 1 to $+1$ or -1, have been tested, again with the condition (2.s)≡(2.t). The result is the sequence above, valid for any ring.

2.3 Operations for Squaring

At first we remember that matrix squaring has the same asymptotic behaviour as matrix multiplication. One side of the equivalence is obvious, because we can compute $A^2 = A \cdot A$ with a multiplication, the other side is easy too, by observing that

$$\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}^2 = \begin{pmatrix} AB & 0 \\ 0 & BA \end{pmatrix}.$$

Nevertheless we can hope to save at least some operations when dealing with only one operand.

When we use formulas (2) and (3) for squaring, so that we have $A = B$, we can observe that $\forall i, S_i = T_i$. Moreover the first four products $P_1 \dots P_4$ are themselves squares, while the last three products share the three operands: A_{12}, A_{21} and S_4. We can use only (2.s), then substitute (3.p) with:

$$\left\{ \begin{array}{lcl} P_1 &=& S_1^2 \\ P_2 &=& S_2^2 \\ P_3 &=& S_3^2 \\ P_4 &=& A_{11}^2 \\ P_5 &=& A_{12}A_{21} \\ P_6 &=& S_4A_{12} \\ P_7 &=& A_{21}S_4 \end{array} \right. \tag{4}$$

The squaring operation, then, requires half the pre-combination, since it operates on one matrix only. This was true for the original Strassen method too, but the new sequence is shorter.

Lemma 1. *Any division free bilinear algorithm (combination; product; combination) strategy to compute the square of a 2×2 matrix, requires at least three products that are not squares.*

Proof. Let $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ be a matrix. Consider the $\mathbb{F}_2$-vector space of linear combinations of the 16 possible (non-commutative) products of the four entries: $\{aa, ab, ba, \dots\}$.

Let $S = \langle aa, aa+ab+ba+bb, \ldots \rangle$ be the sub-space generated by all possible squares of $\mathbb{F}_2$-linear combinations of the entries, and $Q = \langle aa+bc, ab+bd, ca+dc, dd+cb \rangle$ be the one generated by the entries of M^2.

It is easy to verify that the intersection has dimension one $S \cap Q = \langle aa+bc+cb+dd \rangle$. Then a basis of Q must contain at least three elements that are not obtained by squaring or linear combinations of squares. $\square$

Shared Triple Product

The additive complexity (the number of linear operations) for computing the three products $A_{12}A_{21}$, $A_{21}S_4$, S_4A_{12}, is the same as the additive complexity of squaring, so that the saving extend to any recursion level thanks to the following lemmas.

LEMMA 2. *The additive complexity of computing the three products AB, BC, CA with the proposed sequence can be made the same as the additive complexity of computing three different squares (e.g. A^2, B^2, C^2).*

PROOF. By induction.

For a single recursion level: given the symmetry of (2.s) and (2.t), the linear combinations computed on the matrix A for the product AB can be used, without any re-computation for the product CA; the same holds for the other matrices. Thus we need a set of combinations for each matrix, as if we had to compute three squares.

For any additional recursion level: let $A_{S1}, A_{S2}, \cdots, C_{S4}$ be the combined sub-matrices. We need to compute the following products:

$$\begin{array}{lllllll}
A_{S1}B_{S1} & , & B_{S1}C_{S1} & , & C_{S1}A_{S1} & , \\
A_{S2}B_{S2} & , & B_{S2}C_{S2} & , & C_{S2}A_{S2} & , \\
A_{S3}B_{S3} & , & B_{S3}C_{S3} & , & C_{S3}A_{S3} & , \\
A_{11}B_{11} & , & B_{11}C_{11} & , & C_{11}A_{11} & , \\
A_{12}B_{21} & , & B_{21}C_{S4} & , & C_{S4}A_{12} & , \\
A_{S4}B_{12} & , & B_{12}C_{21} & , & C_{21}A_{S4} & , \\
A_{21}B_{S4} & , & B_{S4}C_{12} & , & C_{12}A_{21} & .
\end{array}$$

Each one of the lines above represents a triple product, thus, by induction, is equivalent to the triplet of squares or triple products used by the squaring sequence. $\square$

COROLLARY 1. *Computing the square of a matrix requires half the pre-combinations than a general product, and this is true for any recursion level.*

PROOF. The statement is obvious for one recursion level, because of symmetry. Lemma 2 extends the saving to any recursion level. $\square$

3. OPERATIONS COLLAPSING

When computing chain products or a power, we can further reduce the number of linear combinations, collating the post-combination sequence of partial results with the pre-combination needed for the next multiplication.

Let us take the simplest example: compute the product of three matrices ABC. We can compute $\widetilde{A} = AB$, then $\widetilde{A}C$, but we do not really need to explicitly compute all the elements of $\widetilde{A}$, which is only a partial result, we can modify our sequence to obtain exactly, and only, the values needed for the next product.

The sequence below collapses formulas (3.c) and (2.s) skipping the unneeded value $\widetilde{A}_{22}$, as a result we save 2 operations.

$$\begin{pmatrix} \widetilde{A}_{11} \\ \widetilde{A}_{21} \\ \widetilde{A}_{12} \\ \widetilde{S}_1 \\ \widetilde{S}_2 \\ \widetilde{S}_3 \\ \widetilde{S}_4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & -1 & 1 & 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & -1 & 1 \\ 0 & 0 & 1 & -1 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \end{pmatrix} \qquad (5)$$

$$\left\{ \begin{array}{l} \left. \begin{array}{ll} \widetilde{A}_{11} & = P_4 + P_5 \\ \widetilde{A}_{12} & = P_3 - P_2 - P_6 + P_5 \\ \widetilde{S}_2 & = P_2 + P_7 \\ \widetilde{S}_1 & = P_1 - P_6 \end{array} \right\} (6.c) \\ \hdashline \left. \begin{array}{ll} \widetilde{S}_3 & = \widetilde{S}_2 + \widetilde{A}_{12} \\ \widetilde{A}_{21} & = \widetilde{S}_1 - \widetilde{S}_3 \\ \widetilde{S}_4 & = \widetilde{S}_3 - \widetilde{A}_{11} \end{array} \right\} (6.s) \end{array} \right. \qquad (6)$$

The above consideration can be generalised to any matrix chain product $\prod_{i=1}^{n} A_i$, saving $(n-2) \cdot 2$ combinations, but any such product should be re-implemented from scratch. So we need a more general approach.

3.1 Intermediate Representation

Equation (6) splits in two sub-sequences: (6.c) and (6.s). The first one computes four values from the products P_i, while the last three values only depend on already computed ones. This allows us to only compute the first four values, then store the intermediate result as: $\begin{pmatrix} \widetilde{A}_{11} & \widetilde{A}_{12} \\ \widetilde{S}_2 & \widetilde{S}_1 \end{pmatrix}$.

This intermediate representation is tightly linked with the standard one, and we can switch from one another simply applying the invertible linear function: ψ_1. This function only requires one addition and one subtraction, both in-place; the same occurs for its inverse that requires two subtractions.

$$\psi_1 \left(\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \right) = \begin{pmatrix} A_{11} & A_{12} \\ A_{22} - A_{21} & A_{22} + A_{12} \end{pmatrix}$$

$$\psi_1^{-1} \left(\begin{pmatrix} A_{11} & A_{12} \\ S_2 & S_1 \end{pmatrix} \right) = \begin{pmatrix} A_{11} & A_{12} \\ (\mathbf{S_1} - \mathbf{A_{12}}) - S_2 & (\mathbf{S_1} - \mathbf{A_{12}}) \end{pmatrix}$$

Since ψ_1 is linear, it commutes with linear combinations:

$$\forall A, B \in M_{d \times d}; \forall \alpha, \beta : \psi(\alpha A \pm \beta B) = \alpha \psi(A) \pm \beta \psi(B) \qquad (7)$$

When Strassen's algorithm is used with more levels of recursion, we can also recursively define deeper transform ψ_n:

$$\psi_{n+1} \left(\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \right) = \begin{pmatrix} \psi_n(A_{11}) & \psi_n(A_{12}) \\ \psi_n(A_{21}) & \psi_n(A_{22}) \end{pmatrix}$$

Those functions work on blocks and commute with one another:

$$\forall A, B \in M_{d \times d}; \forall a, b \leq \log_2 d : \psi_a \circ \psi_b = \psi_b \circ \psi_a$$

With an abuse of notation we can say that any composition $\psi = \bigcirc_{i=1}^{n} \psi_i$ is linear, and equation (7) is always valid.

Carefully using one or both pre-combinations (2.s) and (6.s), the products (3.p), then one of the post-combinations (3.c) or (6.c), it is possible to build procedures taking as input the couple $\psi_a(A), \psi_b(B)$ and giving the output $\psi_c(AB)$, for any needed a, b, c. All combinations are possible and handling all of them requires much care for details, a single example will be given in subsection 3.3.

Each transform ψ_n requires additions/subtractions on half the elements of the matrix, and saves one fourth of linear operation each time the operand is used in a product. So it is worth transforming each operand used more than twice for a sequence of operations.

For example, to compute $A^7 = (A^2 \cdot A)^2 \cdot A$, we should start with $\psi(A)$.

Every intermediate result should be stored ψ-transformed, because this saves at least two operations. Conversely, the final result of the computation should be obtained with the original post-sequence (3.c), which is shorter than (6.c) followed by ψ^{-1}.

3.2 Intermediate Representation Optimality

On the side of memory footprint, the intermediate representation uses the same memory used by the usual dense unstructured matrix representation. If there are no relations *a-priori*, there is no way to squeeze the information in a smaller space.

We can then claim the following

LEMMA 3. *The additive complexity obtained with Intermediate Representation is optimal for* 2×2 *dense matrices stored as four entries.*

We will use two results. The first, due to Probert [16], says that the seven multiplicand must be all different combinations of values from the matrix. The second result, by Kaminski et al. [11], give us a relation between the minimal additive complexity $\delta(s)$ of the pre-combination phase for each one of the two operands and the additive complexity $\delta(c)$ of the post-combination:

$$\delta(c) \geq \delta(s) + 3. \tag{8}$$

Note that for both Strassen's original formulas ($\delta(s_S) = 5$, $\delta(c_S) = \delta(s_S) + 3$ for a grand total of $2\delta(s_S) + \delta(c_S) = 3\delta(s_S) + 3 = 18$ combinations) and Winograd's variant (i.e. $\delta(s_W) = 4$ and the total is $3\delta(s_W)+3 = 15$) we have equality in relation (8).

PROOF. When both operands and the result use the intermediate representation, a product consists in the precomputation (6.s), followed by the products (3.p) and recombined with (6.c).

Since we store 4 values, and we need 7 different ones, at least 3 linear combinations are required. The count of $\delta(s) = 3$ operations for equation (6.s) is then minimal.

The number of combinations in (6.c) gives $\delta(c) = 6$. We have equality in (8), so that also this value is minimal. The grand-total $2\delta(s)+\delta(c) = 3\delta(s)+3 = 12$ is then the minimum number of linear combinations required for a product of 2×2 matrices obtained with 7 products. $\square$

We did not prove that there can not exist any other sequence, solving an equation similar to (5) for some Strassen-like product, with a smaller number of operations. We can only claim that such a sequence can not give also an optimal representation with respect to stored elements.

3.3 An Example for Chain Products

A possible application for the intermediate representation is the computation of chain products. While computing $\prod_{i=1}^{n} B_i$ we may represent all partial product $R_j = \prod_{i=1}^{j} B_i$ ψ-transformed.

Then we will loop on the primitive $R_{j+1} \leftarrow R_j B_{j+1}$, where the two R are transformed, but B is not. We give here all details starting from the 2×2 matrices

$$\psi(R_j) = \begin{pmatrix} R_{11} & R_{12} \\ S_2 & S_1 \end{pmatrix}, \quad B_{j+1} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

We use (6.s) on R_j and (2.t) on B,

$$\begin{cases} S_3 = S_2 + R_{12} \\ R_{21} = S_1 - S_3 \\ S_4 = S_3 - R_{11} \end{cases} \qquad \begin{cases} T_1 = B_{22} + B_{12} \\ T_2 = B_{22} - B_{21} \\ T_3 = T_2 + B_{12} \\ T_4 = T_3 - B_{11} \end{cases}$$

then the seven products in (3.p), and finally (6.c)

$$\begin{cases} P_1 = S_1 T_1 \\ P_2 = S_2 T_2 \\ P_3 = S_3 T_3 \\ P_4 = R_{11} B_{11} \\ P_5 = R_{12} B_{21} \\ P_6 = S_4 B_{12} \\ P_7 = R_{21} T_4 \end{cases} \qquad \begin{cases} \widetilde{R}_{11} = P_4 + P_5 \\ \widetilde{R}_{12} = P_3 - P_2 - P_6 + P_5 \\ \widetilde{S}_2 = P_2 + P_7 \\ \widetilde{S}_1 = P_1 - P_6 \end{cases}$$

to obtain

$$\psi(R_{j+1}) = \begin{pmatrix} \widetilde{R}_{11} & \widetilde{R}_{12} \\ \widetilde{S}_2 & \widetilde{S}_1 \end{pmatrix}.$$

We saved one pre-computation and one post-computation for each product.

For the final result, when we actually need the true value of R_n we can directly use the post-computation from (3.c) in the last step, or we can apply ψ^{-1} after it.

3.4 Impact on Complexity

A referee suggested to compute the total number of operations needed for the product of two $d \times d$ matrices with $d = 2^k$ a power of 2, using k recursions. It was done by Strassen in his original paper [19], and his formula can easily be generalised.

Let l be the number of linear operations needed in any Strassen-like sequence, the operation count is:

$$d^{\lg 7} \text{ prods } + \left(\frac{l}{3} d^{\lg 7} - \frac{l}{3} d^2\right) \text{adds} = \left(1 + \frac{l}{3}\right) d^{\lg 7} - \frac{l}{3} d^2.$$

Where lg is $\log_2$, the logarithm to the base 2.

Then we can resume all complexities in a single table:

Method	l	operation count
Strassen	18	$7d^{\lg 7} - 6d^2$
Winograd	15	$6d^{\lg 7} - 5d^2$
ψ-representation	12	$5d^{\lg 7} - 4d^2$
squaring	11	$\frac{14}{3} d^{\lg 7} - \frac{11}{3} d^2$
ψ-squaring	9	$4d^{\lg 7} - 3d^2$

Where ψ-operations consider both operands and result ψ-transformed.

Unfortunately the evaluation in real world implementation is much more complex. On one side the results above are unfair because in most situations addition and product (and squaring) costs are not the same. On the other side we know that frequently the recursion does not reach 2×2 matrices, but stops when some threshold is crossed.

4. COMPUTING THE POWER

There are mainly two ways to compute the power A^n of a generic matrix.

Which one is faster, depends on many parameters and implementation details, and is out of the scope of this paper. Here we will shortly outline the two strategies, and focus on the possible savings in linear operations for both algorithms by using $\psi(A)$, the intermediate representation of A.

4.1 Binary Algorithm

The classical fast exponentiation, widely known and used, is based on the binary expansion of the exponent, followed by a clever sequence of the two operations:
$M \leftarrow M^2 \quad ; \quad M \leftarrow AM^2$.

Strassen's original algorithm would require 18 linear combination for every product or squaring.

If we use the intermediate representation for every partial result, each squaring would cost only 9 combinations. If we have $\psi(A)$ computed in a first step, every product would require 2×(6.s)+(6.c)= 12 combinations.

The best results can be reached if we store all the results of the first computation from the sequence (2.s) and we keep them till the end of exponentiation. In this case also the products will require only 9 combinations, so that we halved the additive complexity with respect to Strassen's strategy.

But remember, we did not change the number of multiplications.

4.2 Polynomial Shortcut

Another way to compute the power of $A \in M_{d\times d}$ is:
- compute P, the minimal polynomial of A;
- compute the polynomial $p_n \equiv x^n \pmod{P}$;
- evaluate the polynomial p_n on the matrix.

The polynomial p_n has degree at most $d-1$. There are several methods to evaluate $p_n(A)$, but all of them need $d-2$ matrix products or squarings and some linear operations.

Thanks to the linearity of the intermediate representation, we can use it for partial results in any evaluation algorithm, and we expect to save $3(d-2)$ linear operations with the ψ-representation.

5. IMPLEMENTATIONS

The sequence proposed in §2.2 was implemented by the author for M4RI [1]: a library for linear algebra over $\mathbb{F}_2$, giving a very small (1%) but unexpected speed up for plain products.

Another unexpected result is a speed up for multiplication of rectangular matrices with the extended algebra described by D'Alberto and Nicolau [6]. Here the advantage of the new sequence with respect to Winograd's comes from the fact that the new one uses many times the sub-matrix A_{22} and only a few times A_{11} (the same for B). The former being smaller than the latter when the matrix is unevenly split. Thus the new sequence also have the same requisite as the one independently proposed by Loos [12].

5.1 Sketches on Scheduling

The new proposed sequence can be obtained from Winograd's formulas by applying permutations and sign changes, so that the scheduling work done for that sequence [3, 7, 10] can be recycled; the result is that it should be possible to substitute the new sequence into any code already implementing Winograd's, basically with no need to rearrange memory usage.

All the papers above consider only two possible operations: $C \leftarrow A\cdot B$ and $C \leftarrow C + A\cdot B$. In fast GCD [14] or Jacobi symbol computation [4], another operation is typically used: $A \leftarrow A \cdot B$, where one of the two operands gets overwritten.

Only one implementation was found by the author, in the GMP-4.3 library, using Winograd's and six temporary variables. The new version of the GMP library [8] contains an implementation by the author of the new sequence. It uses only four temporaries thanks to the following schedule.

1	$U_1 \leftarrow A_{12} \cdot B_{21}$	9	$T \leftarrow T + B_{12}$	17	$U_1 \leftarrow S \cdot B_{12}$
2	$A_{22} \leftarrow A_{22} - A_{21}$	10	$A_{22} \leftarrow A_{12} \cdot T$	18	$T \leftarrow B_{22} + B_{12}$
3	$A_{12} \leftarrow A_{22} - A_{12}$	11	$A_{22} \leftarrow A_{22} - U_1$	19	$U_2 \leftarrow A_{12} \cdot T$
4	$S \leftarrow A_{12} + A_{11}$	12	$T \leftarrow T - B_{11}$	20	$A_{12} \leftarrow U_1 - A_{22}$
5	$U_2 \leftarrow A_{11} \cdot B_{11}$	13	$U_1 \leftarrow A_{21} \cdot T$	21	$A_{21} \leftarrow U_2 - A_{21}$
6	$A_{11} \leftarrow U_2 + U_1$	14	$A_{12} \leftarrow A_{12} + A_{21}$	22	$A_{22} \leftarrow U_2 - A_{22}$
7	$T \leftarrow B_{22} - B_{21}$	15	$A_{21} \leftarrow U_1 + A_{22}$		
8	$U_2 \leftarrow A_{22} \cdot T$	16	$A_{22} \leftarrow A_{22} - U_2$		

Table 1: Scheduling for the $A \leftarrow A \cdot B$ operation.

If B can be overwritten as well, the temporary T can be removed, using B_{21} for it. Anyway, the main purpose of this section is not to exhaust the subject of scheduling, but the opposite: to remark the fact that a lot of work can be done on new sequences, probably exploring all of them as collected in Appendix R.

For example the sequence in table 1 is not exactly the one described for squaring, because some sign was changed...

6. CONCLUSIONS

We have shown, in §2.2, a new sequence for Strassen-like matrix multiplication. This new sequence is optimal with respect to multiplicative and additive complexity for generic 2×2 matrices and for recursive use.

Thanks to the additional property of symmetry, half of the preliminary operations can be saved when the product involves one operand only. The squaring case also has the maximal number of recursive multiplications being themselves squares. Again the new sequence is optimal.

The sequence can be shortened even more with some linear pre-computations. We have shown an in-place transform for matrices, requiring 2 operations per recursion level, after which any product costs 3 operations less. Transformed and standard matrices can be mixed, and some gain can be achieved for chain products even if none of the operands is transformed in advance.

We then propose the use of our new sequence for every implementation of Strassen's matrix multiplication. It is not worse than the widely used Winograd sequence for simple multiplications, but it can give performance gain for squaring, chain products and general polynomial computation.

Acknowledgements

The authors want to thank the anonymous reviewers for their valuable suggestions and corrections.

7. REFERENCES

[1] Martin Albrecht and Gregory Bard. *The M4RI Library – Version 20090409*. The M4RI Team, 2009.

Dimension	2	3	4	5	6	7	8	9	16
Product [13, p.30]	7	23	46	**93**	141	**235**	316	**473**	2212
Naïve squaring (9)	**5**	**18**	46	95	171	280	428	621	3736
Combined squaring	**5**	**18**	**41**	**93**	**141**	**235**	**302**	**473**	**2156**

Table 2: Number of ring products required for squaring small matrices.

[2] Dario Bini and Victor Pan. *Polynomial and Matrix Computations*, volume 1. Birkhauser, Boston, USA, 1994.

[3] Brice Boyer, Jean-Guillaume Dumas, Clément Pernet, and Wei Zhou. Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm. In *ISSAC '09: Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, pages 55–62, Seoul, Corea, 2009. ACM.

[4] Richard P. Brent and Paul Zimmermann. An $O(M(n)\log n)$ algorithm for the Jacobi symbol. In Guillaume Hanrot, François Morain, and Emmanuel Thomé, editors, *Proceedings of the 9th Algorithmic Number Theory Symposium (ANTS-IX)*, volume 6197 of *LNCS*, Nancy, France, July 19-23, 2010. Springer.

[5] Nader H. Bshouty. On the additive complexity of 2x2 matrix multiplication. *Information processing letters*, 56(6):329–336, December 1995.

[6] Paolo D'Alberto and Alexandru Nicolau. Adaptive Winograd's matrix multiplications. *Transactions on Mathematical Software*, 36(1):1–23, 2009.

[7] Craig C. Douglas, Michael Heroux, Gordon Slishman, and Roger M. Smith. GEMMW: A portable level 3 BLAS Winograd variant of Strassen's matrix–matrix multiply algorithm. *Journal of Computational Physics*, 110(1):1–10, 1994.

[8] Torbjörn Granlund. *GNU MP – The GNU Multiple Precision Arithmetic Library.* The GMP development team, 2010.

[9] Te C. Hu. Revised matrix algorithms for shortest paths. *SIAM Journal on Applied Mathematics*, 15(1):207–218, Jan 1967.

[10] 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 CCS-TR-96-17, Center for Computing Sciences, November 15 1996.

[11] Michael Kaminski, David G. Kirkpatrick, and Nader H. Bshouty. Addition requirements for matrix and transpose matrix product. *Journal of Algorithms*, 9:354–364, 1988.

[12] Sarah M. Loos and David S. Wise. Strassen's matrix multiplication relabeled. http://src.acm.org/loos/loos.html, December 2009.

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

[14] Niels Möller. On Schönhage's algorithm and subquadratic integer GCD computation. *Mathematics of Computation*, 77(261):589–607, 2008.

[15] Vittorio Ottaviani, Alberto Zanoni, and Massimo Regoli. Conjugation as public key agreement protocol in mobile cryptography. In *SECRYPT*, Athens, Greece, 2010.

[16] Robert L. Probert. On the additive complexity of matrix multiplication. *SIAM Journal on Computing*, 5(2):187–203, June 1976.

[17] Eligijus Sakalauskas, Povilas Tvarijonas, and Andrius Raulynaitis. Key agreement protocol (KAP) using conjugacy and discrete logarithm problems in group representation level. *Informatica*, 18(1):115–124, 2007.

[18] René Schott and George Stacey Staples. Nilpotent adjacency matrices, random graphs and quantum random variables. *Journal of Physics A: Mathematical and Theoretical*, 41(15), 2008.

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

[20] Abraham Waksman. On Winograd's algorithm for inner products. *IEEE Transactions on Computers*, 19(4):360–361, April 1970.

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

APPENDIX

Q. COMMUTATIVE MATRIX SQUARING

For very small matrices, recursive methods are often too expensive, particularly when we can use commutativity of the base ring.

For 2×2 and 3×3 matrices, we trivially have

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^2 = \begin{pmatrix} a^2+bc & b(a+d) \\ c(a+d) & d^2+bc \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}^2 = \begin{pmatrix} a^2+bd+cg & b(a+e)+ch & c(a+i)+bf \\ d(a+e)+fg & e^2+bd+fh & cd+f(e+i) \\ g(a+i)+dh & h(e+i)+bg & i^2+fh+cg \end{pmatrix}$$

The formula can be generalised to any dimension requiring

$$d \text{ squares}, d^3-d^2-\binom{d}{2} \text{ products}, d^3-d^2-\binom{d}{2} \text{ additions.} \quad (9)$$

This trivial optimisation should be compared to Waksman multiplication [20] for $d\times d$ matrices, and to recursive use of Strassen-like algorithm. It wins only for dimension 2 and 3. Nevertheless its combined use with one or more recursions of the sequence proposed in §2.2 can give the best algorithm for squaring very small matrices if we count the total number of ring products required. In the table 2 we compare our matrix squaring with the best known algorithms to compute the product of two matrices.

In particular we should notice that:

- for the 5×5 squaring, Waksman requires 93 products, equation (9) proposes 90 products plus 5 squares, which may be better in some cases;

- for the 6×6 case, Waksman needs 141 multiplications (sums $\cong 480$), our new sequence splits in four 3×3 squares (using (9)), and three 3×3 products (Waksman), totalling $23 \cdot 3 + 15 \cdot 4 = 129$ products and $3 \cdot 4 = 12$ squarings; same number of operations (fewer sums $\cong 390$), but with a better product/square ratio.

R. OTHER RESULTS IN GF(2)

There are some obvious transformations which allow us to take a Strassen-like matrix multiplication algorithm and to obtain another [5].

Let J be the matrix $J = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$: the two transformations $A \cdot B = (B^T \cdot A^T)^T$ and $A \cdot B = J\left((JAJ) \cdot (JBJ)\right)J$ (which is the symmetry we made mention of in §2.2), are the only two preserving both the kind of operations and the existing symmetry.

By adding other transformations not requiring additional linear operations, namely $A \cdot B = J((JA) \cdot B) = (A \cdot (BJ))J$, we can group all the possible Strassen-like sequences in four equivalence classes:

1. one containing the proposed sequence, the symmetric equivalent, Winograd's and 5 different ones;
2. one containing the original Strassen's and 3 more;
3. one with $A_{11} \cdot (B_{11}+B_{12})$; $(A_{12}+A_{22}) \cdot (B_{12}+B_{22})$; $(A_{12}+A_{21}+A_{22}) \cdot (B_{12}+B_{21})$; $A_{12} \cdot (B_{21}+B_{22})$; $(A_{21}+A_{22}) \cdot B_{21}$; $A_{21} \cdot (B_{11}+B_{21})$; $(A_{11}+A_{12}+A_{21}+A_{22}) \cdot B_{12}$, and 15 more sequences
4. and the one represented by $(A_{11}+A_{12}+A_{22}) \cdot (B_{11}+B_{12}+B_{22})$; $A_{11} \cdot (B_{12}+B_{22})$; $A_{22} \cdot B_{21}$; $(A_{11}+A_{12}+A_{21}+A_{22}) \cdot (B_{11}+B_{12})$; $(A_{12}+A_{22}) \cdot (B_{11}+B_{12}+B_{21}+B_{22})$; $A_{21} \cdot B_{11}$; $(A_{11}+A_{12}) \cdot B_{22}$, containing also 7 other sequences.

Totalling only 36 possible combinations in $\mathbb{F}_2$. Obviously every multiplication sequence in characteristic zero must be a lifting of one of them.

Any search for possible schedulings should test at least one sequence in each one of the classes above, because Strassen's and Winograd's are not the only sequences available.

T. TIMINGS

The new sequence has many possible applications, and the intermediate representation adds many possible variants. It is quite difficult to implement all possible variations and to show a graph giving all needed informations at a glance.

The author implemented some simple functions to compute 2×2 matrix product and squaring, using some different strategies. The entries are integers, for this test implementation the `mpz` type provided by GMP-5.0.1 [8] was used. Timings have been measured on a 32-bits Centrino, running Debian GNU/Linux. With different architectures the actual numbers can be different, but the shapes should be similar.

Figure 1 shows relative timings for matrix-matrix multiplication, comparing four different algorithms. The naïve 8-products algorithm is used as a reference, its timings are normalised as 100%. Two algorithms are indistinguishable: Winograd, and the new proposed sequence. This was expected, because the operation count is exactly the same; the threshold between the naïve algorithm and Strassen-like is somewhere around 500 bits.

The fourth algorithm named "psi multiplication" is the product with both operands and the result using the intermediate representation, three linear operations are saved. As a result it wins on the naïve algorithm a little bit earlier, and it is a little bit faster than plain Winograd for all entries sizes.

The second graph in Figure 2 shows relative timings for squaring. The algorithm are named in the key sorted by their timings on the right side of the graph (operands with entries of 800-bits).

Naïve multiplication and naïve squaring are the same algorithm. Using a single operand, squaring compute the squares of two entries. Squaring an integer is faster than performing a multiplication, thanks to the underling GMP library, that's why the timings are different.

Winograd uses only one squaring on the entries, and its threshold with respect to naïve squaring is around 700 bits, much larger than the multiplication threshold.

Used for squaring, the new sequence is much faster than Winograd. For this implementation and this dimension, we can argue that the speed saving mostly come from the high number of multiplications replaced by squarings on entries. The "psi-squaring" uses the intermediate representation and saves some more linear operation; we can observe that the additional speed-up is not large.

The real winner in this graph is the algorithm exploiting the commutativity of integers, the one described in Appendix Q. We remember here that it can only win for 2×2 or 3×3 matrices, where the total number of non-linear operations (products or squarings) can be reduced.

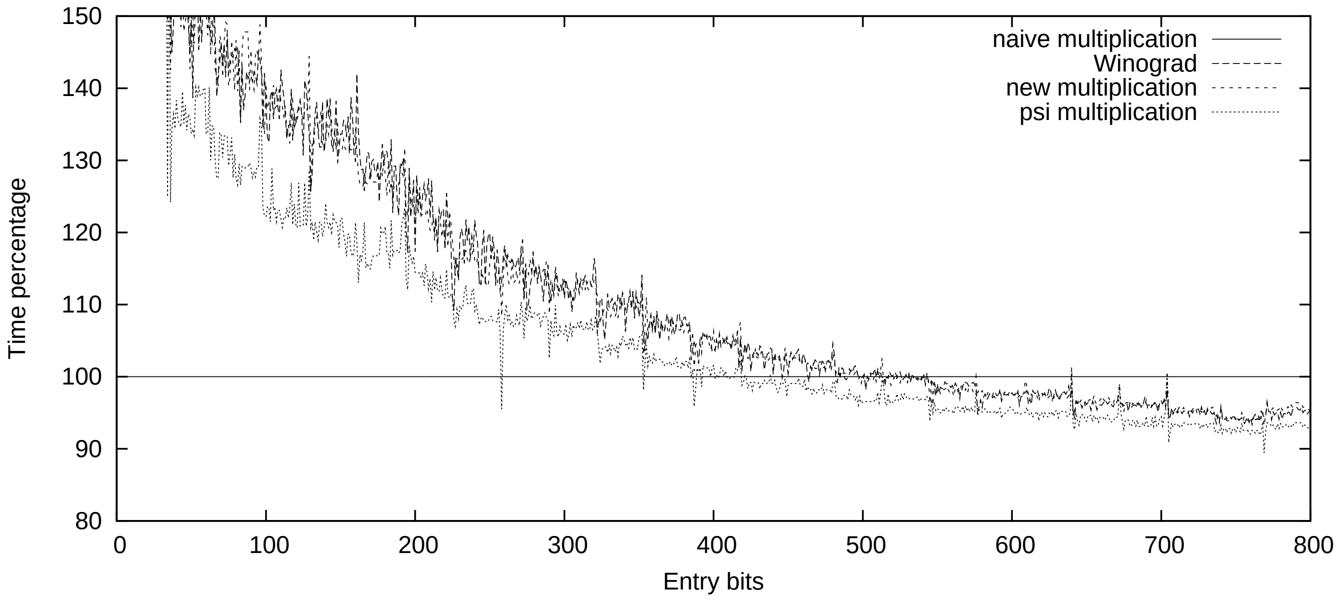

Figure 1: Matrix-matrix product timing comparisons.

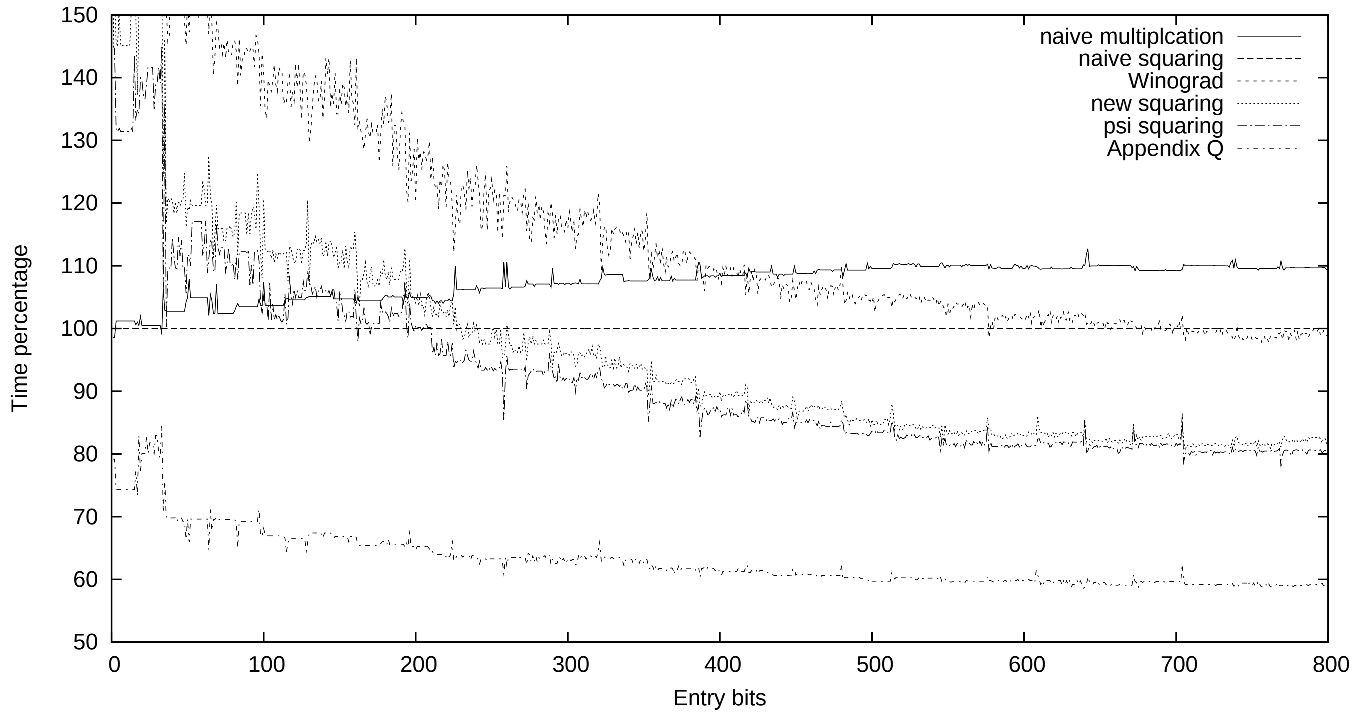

Figure 2: Matrix squaring timing comparisons.