A Strassen-like Matrix Multiplication Suited for Squaring and Highest Power Computation*

Marco Bodrato

Centro "Vito Volterra" – Università di Roma Tor Vergata – Italy

Abstract. Strassen method is not the asymptotically fastest known matrix multiplication algorithm, but it is the most widely used for large matrices on finite fields. Since his manuscript was published, a number of variants have been proposed with various addition complexities. Here we describe a new one. The new variant is as good as those already known for a simple matrix multiplication, but can save operations either when more than two matrices are to be multiplied or for squaring. Moreover it can be proved optimal for this tasks. The biggest gain is shown for n^{th}-power computation, in this scenario the additive complexity can be halved, with respect to original Strassen's.

Keywords: Matrix, Strassen, fast multiplication, optimal squaring, exact computation.

1 Introduction

We analyse Strassen method [Str69] for matrix multiplication, because it is the most widely used, even if is not the asymptotically fastest known. The original Strassen algorithm and the Winograd variant use the same number of multiplications. They differ on the number of additions or subtractions, needing respectively 18 and 15 for a 2×2 matrix. None of the two algorithms has the minimal number of additions when used for squaring, so other similar combinations are investigated.

We propose here a new sequence, optimal for squaring. We then investigate a method for further reducing the number of linear combinations needed either for the consecutive multiplications of three or more matrices, or required by n^{th}-power computation or even by more general polynomials on matrices.

With the proposed algorithm, each squaring or multiplication during power computation requires only 9 additions for a 2×2 matrix. Moreover, when squaring, some of the sub-multiplications are squares, so that optimised squaring on the ring of elements, or for very small matrices, can be used.

While matrix product and Strassen algorithm can be applied to rectangular matrices, this paper focus on square matrices, because only for this kind of matrices squaring and higher power computation make sense. Anyway the main sequence is valid for rectangular matrices too, and standard peeling/padding techniques can be used for odd-sized one [DHSS94,HLJJ+96].

* Part of the project "Algoritmi ottimali per polinomi di matrici su campi finiti"

1.1 Applications

Strassen's and Strassen-like methods for matrix multiplication are considered not stable numerically; although some corrections are possible, the main application for a fast matrix multiplication algorithm is that of matrices on exact rings. We mean matrices over finite fields, integers, rationals, polynomials over finite fields and so on.

To obtain formulas valid on every finite field we started searching combinations valid for $\mathbb{F}_2$. Then we extended the sequences found this way to char 0 testing the same formulas with $\pm 1 \in \mathbb{Z}$ in the places where we had $1 \in \mathbb{F}_2$. 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 ring of matrices, and can be used recursively.

2 Matrices 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

While naïve algorithm requires $\Theta(d^3)$ operations, asymptotically faster algorithms exist. The first one, proposed in 1969 by Strassen [Str69], has the lower complexity $\Theta(d^{\log_2 7}) = \mathrm{O}(d^{2{,}8074})$. 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 recursively used 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. This additive complexity was reduced to 15 operations by a variant credited to Winograd, and this number of operations was shown to be the minimum [Pro76]. Since either the 7 multiplications and the 15 linear operations where the proved minimum, the search for other variants stopped.

2.2 New Proposed Sequence

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

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

$$(2.s)\begin{cases} S_1{=}A_{22}+A_{12} \\ S_2{=}A_{22}-A_{21} \\ S_3{=}S_2+A_{12}=A_{22}-A_{21}+A_{12} \\ S_4{=}S_3-A_{11}=A_{22}-A_{21}+A_{12}-A_{11} \end{cases} \qquad (2.t)\begin{cases} T_1{=}B_{22}+B_{12} \\ T_2{=}B_{22}-B_{21} \\ T_3{=}T_2+B_{12} \\ T_4{=}T_3-B_{11} \end{cases} \tag{2}$$

then the seven products and the final seven post-combinations:

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

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

This proposed method requires 7 multiplications and $4+4+7=15$ linear combinations, this was proved the best possible [Pro76], but is not better than already known sequences.

The sequence where 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 $X = A$, $X = B$ and $X = C$.

Then all the liftings of the sequence in $\mathbb{F}_2$ up to $\mathbb{Z}$, lifting 1 to $+1$ or -1, has been tried, again with the condition (2.s)$\equiv$(2.t). The result is the sequence above, valid for any finite field and in general 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:

$$\begin{cases} P_1 = S_1^2 \\ P_2 = S_2^2 \\ P_3 = S_3^2 \\ P_4 = A_{11}^2 \\ P_5 = A_{12}A_{21} \\ P_6 = S_4A_{12} \\ P_7 = A_{21}S_4 \end{cases} \tag{4}$$

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.

Shared Triple Product When computing the three products $A_{12}A_{21}$, $A_{21}S_4$, S_4A_{12}, the linear pre-combinations described in (2.s) can be computed once for each matrix, this means one for each product. Thus the linear complexity of this product is the same as linear complexity of squaring.

It's not hard to see that this triple product can be used recursively. This means that the square of a matrix require half the pre-combinations than a general product, and this is true for any recursion level.

3 Operations Collapsing

When computing consecutive products, 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.

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

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 (5) splits in two sub-sequences: (5.c) and (5.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:

$$\forall A, B \in M_{d\times d}(\mathbb{F}); \forall \alpha, \beta \in \mathbb{F} : \psi(\alpha A \pm \beta B) = \alpha\psi(A) \pm \beta\psi(B) \tag{6}$$

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

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

Working on blocks, those functions commute with one another:

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

Any composition $\psi = \bigcirc_{i=1}^{n} \psi_i$ is linear, and equation (6) is always valid.

Carefully using one or both pre-combinations (2.s) and (5.s), the products (3.p), then one of the post-combinations (3.c) or (5.c), it is possible to build procedures taking in input the couple $\psi_a(A), \psi_b(B)$ and giving the output $\psi_c(C)$, for any needed a, b, c.

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 (5.c) followed by ψ^{-1}.

3.2 Optimality of the Intermediate Representation

By the results in [KKB88] we have linear complexity $\delta(s)$ of the pre-combination phase (for one matrix) and the linear complexity $\delta(c)$ of the post-combination satisfy the equation: $\delta(c) = \delta(s) + 3$.

Infact Strassen's had $\delta(s_S) = 5$ and $2\delta(s_S) + \delta(c_S) = 3\delta(s_S) + 3 = 18$; Winograd reached $\delta(s_W) = 4$ and $3\delta(s_W) + 3 = 15$, while the intermediate representation allows $\delta(s) = 3$ and $3\delta(s) + 3 = 12$, saving 3 other operations.

Further reductions are not possible, because the seven left-side (resp. right-side) operands in the multiplications (3.p) can not repeat [Pro76]. If we start from the four elements of a 2×2 matrix, at least 3 operations are necessary to obtain 7 unique combinations.

4 Computing the n^{th}-Power

There are basically two ways to compute the power A^n of a generic matrix:

- binary expansion of the exponent, then a clever sequence of the two operations: $M \leftarrow M^2 \quad ; \quad M \leftarrow AM^2$.
- evaluate the polynomial $p_n \equiv x^n \pmod P$ on the matrix, where P is the minimal polynomial of A.

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

4.1 Binary Algorithm

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 $2\times$(5.s)+(5.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 linear operations needed with Strassen's strategy.

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

4.2 Polynomial Shortcut

Since the transformation to and from the intermediate representation is linear, we can use it for partial results inside any evaluation algorithm using only multiplications and linear combinations.

We expect at least as many products as the degree $d = \deg(P)$ of the polynomial, then we expect to save at least $3d$ linear operations with the ψ-representation.

5 Conclusions

We have shown, in §2.2, a new sequence for Strassen-like matrix multiplication [Str69]. This new sequence is optimal with respect to multiplicative and additive complexity for generic 2×2 matrices and for recursive use.

As a bonus, with the same sequence, 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

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

Acknowledgements

The author thanks Maria Caterina Tarantino, for her support and encouragement for writing this paper; and Alberto Zanoni for proofreading.

References

BBB+06. Christian Batut, Karim Belabas, Dominique Bernardi, Henri Cohen, and Michel Olivier. *User's Guide to PARI/GP*, 2.4.2 edition, 2006.

DHSS94. 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.

DPZ07. Jean-Guillaume Dumas, Clment Pernet, and Wei Zhou. Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm. Technical Report 0707.2347v3, arXiv, 2007.

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 CCS-TR-96-17, Center for Computing Sciences, November 15 1996. available on-line.

KKB88. Michael Kaminski, David G. Kirkpatrick, and Nader H. Bshouty. Addition requirements for matrix and transpose matrix product. *Journal of Algorithms*, 9:354–364, 1988.

Mez07. Marc Mezzarobba. *Génération automatique de procédures numériques pour les fonctions D-finies*. PhD thesis, Master parisien de recherche en informatique, August 2007.

Pro76. Robert L. Probert. On the additive complexity of matrix multiplication. *SIAM Journal of Computation*, 5(2):187–203, June 1976.

Str69. Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

Wak70. Abraham Waksman. On Winograd's algorithm for inner products. *IEEE Transactions on Computers*, 19(4):360–361, April 1970.

Q Small 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 matrices, we trivially have

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

This easy formula can be generalised with the small program in figure 1, requiring

$$d \quad \text{squares}, \quad d^3 - d^2 - \binom{d}{2} \quad \text{products and} \quad d^3 - d^2 - \binom{d}{2} \quad \text{additions.} \tag{7}$$

Input : $d \times d$ matrix A
Output : $d \times d$ matrix Q

for $i = 1 \dots d$
  $Q_{i,i} \leftarrow (A_{i,i})^2$
for $i = 1 \dots d-1$
  for $j = i+1 \dots d$
    $c \leftarrow A_{i,j} \cdot A_{j,i}$
    $Q_{i,i} \leftarrow Q_{i,i} + c$
    $Q_{j,j} \leftarrow Q_{j,j} + c$
    $c \leftarrow A_{i,i} + A_{j,j}$
    $Q_{i,j} \leftarrow Q_{i,j} + A_{i,j} \cdot c$
    $Q_{j,i} \leftarrow Q_{j,i} + A_{j,i} \cdot c$
    for $k = 1 \dots d, k \neq i \wedge k \neq j$
      $Q_{j,i} \leftarrow Q_{j,i} + A_{j,k} \cdot A_{k,i}$
      $Q_{i,j} \leftarrow Q_{i,j} + A_{i,k} \cdot A_{k,j}$

Fig. 1. *Naïve implementation of squaring in the commutative case*

This trivial optimisation should be compared to Waksman multiplication [Wak70] 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 small matrices with respect to the total number of ring products required. In the table below we compare our matrix squaring with the best known algorithms to compute the product of two matrices.

Dimension	2	3	4	5	6	7	8	9	16
Product [Mez07, p.30]	7	23	46	**93**	141	**235**	316	**473**	2212
Naïve squaring (7)	**5**	**18**	46	95	171	280	428	621	3736
Combined squaring	**5**	**18**	**41**	**93**	**141**	**235**	**302**	**473**	**2156**

Number of ring products required for squaring small matrices.

In particular we should notice that:

- for the 5×5 squaring, Waksman requires 93 products, equation (7) 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 (7)), 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 may be better "quality".

R Other Results in $\mathbb{F}_2$

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

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, plus Winograd's and 6 other ones;
2. one containing the original Strassen's, plus 3 others;
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.

S Scheduling

As already said in §R, the new proposal is in the same class of Winograd's, so that the scheduling work done for that sequence [HLJJ+96,DPZ07] can be recycled here; the result is that it should be possible to substitute the new proposed one to any similar code already implemented, and it will not be slower.

For example, the schedule proposed in [DHSS94], with two temporaries only, can be revisited for the sequence seen in §2.2 as follows:

$W_{kn} \leftarrow B_{22} + B_{12}$	$W_{kn} \leftarrow W_{kn} + B_{12}$	$W_{mk} \leftarrow A_{12} \cdot B_{21}$	$C_{21} \leftarrow C_{11} - C_{21}$
$W_{mk} \leftarrow A_{22} + A_{12}$	$W_{mk} \leftarrow W_{mk} + A_{12}$	$C_{11} \leftarrow C_{11} + W_{mk}$	$C_{22} \leftarrow C_{22} + C_{11}$
$C_{21} \leftarrow W_{mk} \cdot W_{kn}$	$C_{11} \leftarrow W_{mk} \cdot W_{kn}$	$C_{12} \leftarrow C_{11} - C_{12}$	$C_{11} \leftarrow A_{11} \cdot B_{11}$
$W_{mk} \leftarrow A_{22} - A_{21}$	$W_{mk} \leftarrow W_{mk} - A_{11}$	$C_{11} \leftarrow C_{21} - C_{11}$	$C_{11} \leftarrow C_{11} + W_{mk}$
$W_{kn} \leftarrow B_{22} - B_{21}$	$C_{12} \leftarrow W_{mk} \cdot B_{12}$	$W_{kn} \leftarrow W_{kn} - B_{11}$	
$C_{22} \leftarrow W_{mk} \cdot W_{kn}$	$C_{12} \leftarrow C_{12} + C_{22}$	$C_{21} \leftarrow A_{21} \cdot W_{kn}$	

T Timings

Some simple timing test where done on straightforward implementations of matrix squaring in GP-Pari [BBB+06]. We compare the internal cubic GP-Pari implementation (Version 2.3.2, with GMP 4.2.1) and a GP-Pari script implementing Strassen-Winograd's ("SW", one level only) scheduled as in [DHSS94] with another scripts using the new proposed sequence described in §2.2 (one recursion level), then one implementing $\psi(A) \to \psi(A^2)$ which is the building brick for exponentiation, and a last one using also fig.1 for 2×2 (or 3×3) matrices. The last column gives percentage difference between ψ and non-ψ implementation.

Time for squaring in ms, for a random $d \times d$ matrix with entries in $\mathbb{F}_p$.

Prime p	dimension	GP-Pari	SW	§2.2	$+\,\psi$	$+$ §Q	ψ%
2	4×4	**0.0251**	0.1167	0.0926	0.0853	0.4118	8.6
2	8×8	**0.1305**	0.3635	0.2867	0.2703	*n.a.*	6.1
2	16×16	**0.8407**	1.4694	1.2208	1.1625	*n.a.*	5.0
2	32×32	**5.7166**	7.7358	6.8055	6.5900	*n.a.*	3.3
2	64×64	**42.4831**	50.4117	46.4033	45.4916	*n.a.*	2.0
2	128×128	334.0690	330.9655	312.3103	**308.9310**	*n.a.*	1.1
17	4×4	**0.0342**	0.1328	0.1010	0.0941	0.4144	7.3
17	32×32	**9.1563**	10.8326	9.9207	9.6982	*n.a.*	2.3
17	64×64	70.8103	75.2069	70.5172	**69.7586**	*n.a.*	1.1
65537	4×4	**0.0354**	0.1323	0.1012	0.0944	0.4145	7.2
65537	32×32	**9.5294**	11.2689	10.2857	10.0672	*n.a.*	2.2
65537	64×64	74.2142	77.7857	73.7142	**72.5000**	*n.a.*	1.7
$\cong 2^{32}$	4×4	**0.0600**	0.1598	0.1278	0.1195	0.4436	6.9
$\cong 2^{32}$	16×16	**2.8718**	3.3760	3.0840	3.0105	*n.a.*	2.4
$\cong 2^{32}$	32×32	22.2372	22.7796	21.6610	**21.3220**	*n.a.*	1.6
$\cong 2^{64}$	4×4	**0.0749**	0.1773	0.1432	0.1345	0.4553	6.5
$\cong 2^{64}$	16×16	**3.8781**	4.3109	4.0210	3.9453	*n.a.*	1.9
$\cong 2^{64}$	32×32	30.4137	30.3103	29.0000	**28.6896**	*n.a.*	1.1
$\cong 2^{256}$	4×4	**0.1605**	0.2703	0.2289	0.2205	0.5424	3.8
$\cong 2^{256}$	8×8	**1.1449**	1.3655	1.2521	1.2247	*n.a.*	2.2
$\cong 2^{256}$	16×16	9.0000	9.1186	8.7118	**8.6101**	*n.a.*	1.2
$\cong 2^{512}$	4×4	**0.3121**	0.4129	0.3691	0.3589	0.6620	2.8
$\cong 2^{512}$	8×8	2.3655	2.4579	2.3277	**2.2941**	*n.a.*	1.5
$\cong 2^{1024}$	4×4	0.8063	0.8679	0.8151	**0.8041**	1.0209	1.4
$\cong 2^{1024}$	6×6	2.6686	2.6338	2.5354	**2.5126**	2.8667	0.9
$\cong 2^{2048}$	4×4	2.4768	2.3109	2.2436	2.2226	**2.1617**	0.9
$\cong 2^{2048}$	6×6	8.0450	7.4573	7.3104	7.2729	**6.7914**	0.5
$\cong 2^{8192}$	4×4	22.2100	19.9916	19.8151	19.7310	**16.1092**	0.4
$\cong 2^{65536}$	4×4	561.5714	499.0000	496.7857	494.7143	**393.4286**	0.4

Tested with GP-Pari 2.3.2 using GMP 4.2.1 on a 1.4GHz Pentium-M.

THIS PAGE IS NOT PART OF THE ARTICLE[1]

BibTEX entry

```
@TechReport{Bodrato:CIVV2008,
  author =   {Marco Bodrato},
  title =   {A {Strassen}-like Matrix Multiplication
  Suited for Squaring and Highest Power Computation},
  institution =  {Centro "Vito Volterra", Universit\`a di Roma "Tor Vergata"},
  year =   {2008},
  number =   {622},
  pages =   {10},
  month =   {December},
  note =   {\url{http://bodrato.it/papers/\#CIVV2008}},
}
```

[1] Paper edited with Emacs-21 and LaTeX-3.0 on a Debian GNU/Linux box.

Π Paraleipómena[2]

Π.1 Scheduling for `addmul` with Odd-Sized Matrices [January the 3^{rd}, 2009]

Another operation often useful is to compute the product and add it up to an existing matrix. In general we can use a linear combination $C \leftarrow \alpha \cdot C + \beta \cdot A \times B$ with two coefficients α, β in the algebra of entries.

The same scheduling proposed in [DAN07] can be adapted here, with almost the same notation. The algorithm below computes $C+ = A * B$ or better

$$\begin{pmatrix} C_0 & C_1 \\ C_2 & C_3 \end{pmatrix} \leftarrow \begin{pmatrix} C_0 & C_1 \\ C_2 & C_3 \end{pmatrix} + \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} \times \begin{pmatrix} B_0 & B_1 \\ B_2 & B_3 \end{pmatrix}.$$

It uses also the operation $C- = A * B$, so it should be generalised to be used in practical programming. The operands have sizes $\sigma(A) = m \times n$, $\sigma(B) = n \times p$, $\sigma(C) = m \times p$. If any size is odd, matrices are split in such a way that A_0, B_0, C_0 are the biggest, for example $\sigma(A_0) = \lceil \frac{m}{2} \rceil \times \lceil \frac{n}{2} \rceil$, $\sigma(A_3) = \lfloor \frac{m}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$.

Definitions for both matrix addition and matrix product must be relaxed somehow, cropping or zero-padding operands as required by result size.

Computation	Operand Sizes
$S = A_3 - A_2$	$\sigma(S) = \lfloor \frac{m}{2} \rfloor \times \lceil \frac{n}{2} \rceil$
$T = B_3 - B_2$	$\sigma(T) = \lfloor \frac{n}{2} \rfloor \times \lceil \frac{p}{2} \rceil$
$U = S * T$	$\sigma(U) = \lfloor \frac{m}{2} \rfloor \times \lfloor \frac{p}{2} \rfloor,\ \sigma(S) = \lfloor \frac{m}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor,\ \sigma(T) = \lfloor \frac{n}{2} \rfloor \times \lfloor \frac{p}{2} \rfloor$
$C_3+ = U$	
$C_1- = U$	
$U = A_1 * B_2$	$\sigma(U) = \lceil \frac{m}{2} \rceil \times \lceil \frac{p}{2} \rceil$
$C_0+ = U$	
$C_0+ = A_0 * B_0$	
$S = S + A_1$	$\sigma(S) = \lceil \frac{m}{2} \rceil \times \lceil \frac{n}{2} \rceil$
$T = T + B_1$	$\sigma(T) = \lceil \frac{n}{2} \rceil \times \lceil \frac{p}{2} \rceil$
$U+ = S * T$	
$C_1+ = U$	
$S = A_0 - S$	
$C_1+ = S * B_1$	$\sigma(S * B_1) = \lceil \frac{m}{2} \rceil \times \lfloor \frac{p}{2} \rfloor$
$T = B_0 - T$	
$C_2+ = A_2 * T$	$\sigma(A_2 * T) = \lfloor \frac{m}{2} \rfloor \times \lceil \frac{p}{2} \rceil$
$S = A_3 + A_1$	$\sigma(S) = \lfloor \frac{m}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$
$T = B_3 + B_1$	$\sigma(T) = \lfloor \frac{n}{2} \rfloor \times \lfloor \frac{p}{2} \rfloor$
$U- = S * T$	$\sigma(U) = \lceil \frac{m}{2} \rceil \times \lfloor \frac{p}{2} \rfloor$
$C_2 = C_2 - U$	
$C_3 = C_3 - U$	

References

DAN07. Paolo D'Alberto and Alexandru Nicolau. Adaptive Winograd's matrix multiplications. *ACM Transactions on Mathematical Software*, Vol. 36, No. 1, 2008.

[2] Added notes available only in the on-line version.