# On fast multiplication of a matrix by its transpose

Jean-Guillaume Dumas, Clément Pernet, Alexandre Sedoglavic

# On Fast Multiplication of a Matrix by its Transpose

Jean-Guillaume Dumas
Université Grenoble Alpes
Laboratoire Jean Kuntzmann, CNRS
UMR 5224, 38058 Grenoble, France

Clément Pernet
Université Grenoble Alpes
Laboratoire Jean Kuntzmann, CNRS
UMR 5224, 38058 Grenoble, France

Alexandre Sedoglavic
Université de Lille
UMR CNRS 9189 CRISTAL
59650 Villeneuve d'Ascq, France

## ABSTRACT

We present a non-commutative algorithm for the multiplication of a $2 \times 2$-block-matrix by its transpose using 5 block products (3 recursive calls and 2 general products) over $\mathbb{C}$ or any field of prime characteristic. We use geometric considerations on the space of bilinear forms describing $2 \times 2$ matrix products to obtain this algorithm and we show how to reduce the number of involved additions. The resulting algorithm for arbitrary dimensions is a reduction of multiplication of a matrix by its transpose to general matrix product, improving by a constant factor previously known reductions. Finally we propose schedules with low memory footprint that support a fast and memory efficient practical implementation over a prime field. To conclude, we show how to use our result in $L \cdot D \cdot L^\intercal$ factorization.

## CCS CONCEPTS

• **Computing methodologies** → **Exact arithmetic algorithms**; **Linear algebra algorithms**.

## KEYWORDS

algebraic complexity, fast matrix multiplication, SYRK, rank-k update, Symmetric matrix, Gram matrix, Wishart matrix

## 1 INTRODUCTION

Strassen's algorithm [20], with 7 recursive multiplications and 18 additions, was the first sub-cubic time algorithm for matrix product, with a cost of $O(n^{2.81})$. Summarizing the many improvements which have happened since then, the cost of multiplying two arbitrary $n \times n$ matrices $O(n^\omega)$ will be denoted by $\mathrm{MM}_\omega(n)$ (see [17] for the best theoretical value of $\omega$ known to date).

We propose a new algorithm for the computation of the product $A \cdot A^\intercal$ of a $2 \times 2$-block-matrix by its transpose using only 5 block multiplications over some base field, instead of 6 for the natural divide & conquer algorithm. For this product, the best previously known complexity bound was dominated by $\frac{2}{2^\omega - 4}\mathrm{MM}_\omega(n)$ over any field (see [11, § 6.3.1]). Here, we establish the following result:

Theorem 1.1. *The product of an $n \times n$ matrix by its transpose can be computed in $\frac{2}{2^\omega - 3}MM_\omega(n)$ field operations over a base field for which there exists a skew-orthogonal matrix.*

Our algorithm is derived from the class of Strassen-like algorithms multiplying $2 \times 2$ matrices in 7 multiplications. Yet it is a reduction of multiplying a matrix by its transpose to general matrix multiplication, thus supporting any admissible value for $\omega$. By exploiting the symmetry of the problem, it requires about half of the arithmetic cost of general matrix multiplication when $\omega$ is $\log_2 7$.

We focus on the computation of the product of an $n \times k$ matrix by its transpose and possibly accumulating the result to another matrix. Following the terminology of the BLAS3 standard [10], this operation is a symmetric rank $k$ update (SYRK for short).

## 2 MATRIX PRODUCT ALGORITHMS ENCODED BY TENSORS

Considered as $2 \times 2$ matrices, the matrix product $C = A \cdot B$ could be computed using Strassen algorithm by performing the following computations (see [20]):

$$\begin{array}{ll} \rho_1 \leftarrow a_{11}(b_{12} - b_{22}), & \\ \rho_2 \leftarrow (a_{11} + a_{12})b_{22}, & \rho_4 \leftarrow (a_{12} - a_{22})(b_{21} + b_{22}), \\ \rho_3 \leftarrow (a_{21} + a_{22})b_{11}, & \rho_5 \leftarrow (a_{11} + a_{22})(b_{11} + b_{22}), \\ \rho_6 \leftarrow a_{22}(b_{21} - b_{11}), & \rho_7 \leftarrow (a_{21} - a_{11})(b_{11} + b_{12}), \\ \multicolumn{2}{c}{\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} \rho_5+\rho_4-\rho_2+\rho_6 & \rho_6+\rho_3 \\ \rho_2+\rho_1 & \rho_5+\rho_7+\rho_1-\rho_3 \end{pmatrix}.} \end{array} \tag{1}$$

In order to consider this algorithm under a geometric standpoint, we present it as a tensor. Matrix multiplication is a bilinear map:

$$\begin{array}{ccc} \mathbb{K}^{m\times n} \times \mathbb{K}^{n\times p} & \to & \mathbb{K}^{m\times p}, \\ (X, Y) & \to & X \cdot Y, \end{array} \tag{2}$$

where the spaces $\mathbb{K}^{a\times b}$ are finite vector spaces that can be endowed with the Frobenius inner product $\langle M, N\rangle = \mathrm{Trace}(M^\intercal \cdot N)$. Hence, this inner product establishes an isomorphism between $\mathbb{K}^{a\times b}$ and its dual space $(\mathbb{K}^{a\times b})^\star$ allowing for example to associate matrix multiplication and the trilinear form $\mathrm{Trace}(Z^\intercal \cdot X \cdot Y)$:

$$\begin{array}{ccc} \mathbb{K}^{m\times n} \times \mathbb{K}^{n\times p} \times (\mathbb{K}^{m\times p})^\star & \to & \mathbb{K}, \\ (X, Y, Z^\intercal) & \to & \langle Z, X \cdot Y\rangle. \end{array} \tag{3}$$

As by construction, the space of trilinear forms is the canonical dual space of order three tensor product, we could associate the Strassen multiplication algorithm (1) with the tensor $\mathcal{S}$ defined by:

$$\begin{array}{l} \sum_{i=1}^7 S_{i1}\otimes S_{i2}\otimes S_{i3} = \left(\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}0&1\\0&-1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}0&0\\1&1\end{smallmatrix}\right)+ \\ \left(\begin{smallmatrix}1&1\\0&0\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}0&0\\0&1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}-1&0\\1&0\end{smallmatrix}\right)+\left(\begin{smallmatrix}0&0\\1&1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}0&1\\0&-1\end{smallmatrix}\right)+ \\ \left(\begin{smallmatrix}0&1\\0&-1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}0&0\\1&1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right)+\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)+ \\ \left(\begin{smallmatrix}0&0\\0&1\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}-1&0\\1&0\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}1&1\\0&0\end{smallmatrix}\right)+\left(\begin{smallmatrix}-1&0\\1&0\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}1&1\\0&0\end{smallmatrix}\right)\otimes\left(\begin{smallmatrix}0&0\\0&1\end{smallmatrix}\right) \end{array} \tag{4}$$

in $(\mathbb{K}^{m\times n})^\star \otimes (\mathbb{K}^{n\times p})^\star \otimes \mathbb{K}^{m\times p}$ with $m = n = p = 2$. Given any couple $(A, B)$ of $2 \times 2$-matrices, one can explicitly retrieve from tensor $\mathcal{S}$ the Strassen matrix multiplication algorithm computing $A \cdot B$ by the *partial* contraction $\{\mathcal{S}, A \otimes B\}$:

$$\begin{array}{c} \left((\mathbb{K}^{m\times n})^\star \otimes (\mathbb{K}^{n\times p})^\star \otimes \mathbb{K}^{m\times p}\right) \otimes \left(\mathbb{K}^{m\times n} \otimes \mathbb{K}^{n\times p}\right) \to \mathbb{K}^{m\times p}, \\ \mathcal{S} \otimes (A \otimes B) \to \sum_{i=1}^7 \langle S_{i1}, A\rangle \langle S_{i2}, B\rangle S_{i3}, \end{array} \tag{5}$$

while the *complete* contraction $\{\mathcal{S}, A \otimes B \otimes C^\intercal\}$ is $\mathrm{Trace}(A \cdot B \cdot C)$.

The tensor formulation of matrix multiplication algorithm gives explicitly its symmetries (a.k.a. *isotropies*). As this formulation is

associated to the trilinear form $\mathrm{Trace}(A \cdot B \cdot C)$, given three invertible matrices $U, V, W$ of suitable sizes and the classical properties of the trace, one can remark that $\mathrm{Trace}(A \cdot B \cdot C)$ is equal to:

$$\begin{array}{l}\mathrm{Trace}\big((A \cdot B \cdot C)^{\mathsf{T}}\big) = \mathrm{Trace}(C \cdot A \cdot B) = \mathrm{Trace}(B \cdot C \cdot A),\\ \text{and } \mathrm{Trace}\big(U^{-1} \cdot A \cdot V \cdot V^{-1} \cdot B \cdot W \cdot W^{-1} \cdot C \cdot U\big).\end{array} \tag{6}$$

These relations illustrate the following theorem:

Theorem 2.1 ([8, § 2.8]). *The isotropy group of the $n \times n$ matrix multiplication tensor is $\mathrm{PSL}^{\pm}(\mathbb{K}^n)^{\times 3} \rtimes \mathfrak{S}_3$, where PSL stands for the group of matrices of determinant $\pm 1$ and $\mathfrak{S}_3$ for the symmetric group on 3 elements.*

The following definition recalls the *sandwiching* isotropy on matrix multiplication tensor:

Definition 2.1. *Given $\mathrm{g} = (U \times V \times W)$ in $\mathrm{PSL}^{\pm}(\mathbb{K}^n)^{\times 3}$, its action $\mathrm{g} \diamond \mathcal{S}$ on a tensor $\mathcal{S}$ is given by $\sum_{i=1}^{7} \mathrm{g} \diamond (S_{i1} \otimes S_{i2} \otimes S_{i3})$ where the term $\mathrm{g} \diamond (S_{i1} \otimes S_{i2} \otimes S_{i3})$ is equal to:*

$$(U^{-\mathsf{T}} \cdot S_{i1} \cdot V^{\mathsf{T}}) \otimes (V^{-\mathsf{T}} \cdot S_{i2} \cdot W^{\mathsf{T}}) \otimes (W^{-\mathsf{T}} \cdot S_{i3} \cdot U^{\mathsf{T}}). \tag{7}$$

Remark 2.1. *In $\mathrm{PSL}^{\pm}(\mathbb{K}^n)^{\times 3}$, the product $\circ$ of two isotropies $g_1$ defined by $u_1 \times v_1 \times w_1$ and $g_2$ by $u_2 \times v_2 \times w_2$ is the isotropy $g_1 \circ g_2$ equal to $u_1 \cdot u_2 \times v_1 \cdot v_2 \times w_1 \cdot w_2$. Furthermore,the complete contraction $\{g_1 \circ g_2, A \otimes B \otimes C\}$ is equal to $\{g_2, g_1{}^{\mathsf{T}} \diamond A \otimes B \otimes C\}$.*

The following theorem shows that all $2 \times 2$-matrix product algorithms with 7 coefficient multiplications could be obtained by the action of an isotropy on Strassen tensor:

Theorem 2.2 ([9, § 0.1]). *The group $\mathrm{PSL}^{\pm}(\mathbb{K}^n)^{\times 3}$ acts transitively on the variety of optimal algorithms for the computation of $2 \times 2$-matrix multiplication.*

Thus, isotropy action on Strassen tensor may define other matrix product algorithm with interesting computational properties.

## 2.1 Design of a specific $2 \times 2$-matrix product

This observation inspires our general strategy to design specific algorithms suited for particular matrix product.

Strategy 2.1. *By applying an undetermined isotropy:*

$$\mathrm{g} = U \times V \times W = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \times \begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix} \times \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \tag{8}$$

*on Strassen tensor $\mathcal{S}$, we obtain a parameterization $\mathcal{T} = \mathrm{g} \diamond \mathcal{S}$ of all matrix product algorithms requiring 7 coefficient multiplications:*

$$\mathcal{T} = \sum_{i=1}^{7} T_{i1} \otimes T_{i2} \otimes T_{i3},\ T_{i1} \otimes T_{i2} \otimes T_{i3} = \mathrm{g} \diamond S_{i1} \otimes S_{i2} \otimes S_{i3}. \tag{9}$$

*Then, we could impose further conditions on these algorithms and check by a Gröbner basis computation if such an algorithm exists. If so, there is subsequent work to do for choosing a point on this variety; this choice can be motivated by the additive cost bound and the scheduling property of the evaluation scheme given by this point.*

Let us first illustrate this strategy with the well-known Winograd variant of Strassen algorithm presented in [22].

Example 1. *Apart from the number of multiplications, it is also interesting in practice to reduce the number of additions in an algorithm. Matrices $S_{11}$ and $S_{61}$ in tensor (4) do not increase the additive cost bound of this algorithm. Hence, in order to reduce this complexity in an algorithm, we could try to maximize the number of such matrices involved in the associated tensor. To do so, we recall Bshouty's results on additive complexity of matrix product algorithms.*

Theorem 2.3 ([6]). *Let $e_{(i,j)} = (\delta_{i,k}\delta_{j,l})_{(k,l)}$ be the single entry elementary matrix. A $2 \times 2$ matrix product tensor could not have 4 such matrices as first (resp. second, third) component ([6, Lemma 8]). The additive complexity bound of first and second components are equal ([6, eq. (11)]) and at least $4 = 7 - 3$. The total additive complexity of $2 \times 2$-matrix product is at least 15 ([6, Theorem 1]).*

*Following our strategy, we impose on tensor $\mathcal{T}$ (9) the constraints*

$$T_{11} = e_{1,1} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right), \quad T_{12} = e_{1,2}, \quad T_{13} = e_{2,2} \tag{10}$$

*and obtain by a Gröbner basis computation [13] that such tensors are the images of Strassen tensor by the action of the following isotropies:*

$$\mathrm{w} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \times \left(\begin{smallmatrix} 1 & -1 \\ 0 & -1 \end{smallmatrix}\right) \times \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}. \tag{11}$$

*The variant of the Winograd tensor [22] presented with a renumbering as Algorithm 1 is obtained by the action of $\mathrm{w}$ with the specialization $w_{12} = w_{21} = 1 = -w_{11}, w_{22} = 0$ on the Strassen tensor $\mathcal{S}$. While the original Strassen algorithm requires 18 additions, only 15 additions are necessary in the Winograd Algorithm 1.*

**Algorithm 1** : $C = \mathrm{W}(A, B)$

**Input:** $A = \left(\begin{smallmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{smallmatrix}\right)$ and $B = \left(\begin{smallmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{smallmatrix}\right)$;
**Output:** $C = A \cdot B$

$s_1 \leftarrow a_{11} - a_{21}$, $s_2 \leftarrow a_{21} + a_{22}$, $s_3 \leftarrow s_2 - a_{11}$, $s_4 \leftarrow a_{12} - s_3$,
$t_1 \leftarrow b_{22} - b_{12}$, $t_2 \leftarrow b_{12} - b_{11}$, $t_3 \leftarrow b_{11} + t_1$, $t_4 \leftarrow b_{21} - t_3$.
$p_1 \leftarrow a_{11} \cdot b_{11}$, $p_2 \leftarrow a_{12} \cdot b_{21}$, $p_3 \leftarrow a_{22} \cdot t_4$, $p_4 \leftarrow s_1 \cdot t_1$,
$p_5 \leftarrow s_3 \cdot t_3$, $p_6 \leftarrow s_4 \cdot b_{22}$, $p_7 \leftarrow s_2 \cdot t_2$.
$c_1 \leftarrow p_1 + p_5$, $c_2 \leftarrow c_1 + p_4$, $c_3 \leftarrow p_1 + p_2$, $c_4 \leftarrow c_2 + p_3$,
$c_5 \leftarrow c_2 + p_7$, $c_6 \leftarrow c_1 + p_7$, $c_7 \leftarrow c_6 + p_6$.
**return** $C = \left(\begin{smallmatrix} c_3 & c_7 \\ c_4 & c_5 \end{smallmatrix}\right)$.

As a second example illustrating our strategy, we consider now the matrix squaring that was already explored by Bodrato in [3].

Example 2. *When computing $A^2$, the contraction (5) of the tensor $\mathcal{T}$ (9) with $A \otimes A$ shows that choosing a subset $J$ of $\{1, \ldots, 7\}$ and imposing $T_{i1} = T_{i2}$ as constraints with $i$ in $J$ (see [3, eq 4]) can save $|J|$ operations and thus reduce the computational complexity.*

*The definition (9) of $\mathcal{T}$, these constraints, and the fact that $U, V$ and $W$'s determinant are 1, form a system with $3 + 4|J|$ equations and 12 unknowns whose solutions define matrix squaring algorithms.*

*The algorithm [3, § 2.2, eq 2] is given by the action of the isotropy:*

$$\mathrm{g} = \left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right) \times \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right) \times \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right) \tag{12}$$

*on Strassen's tensor and is just Chatelin's algorithm [7, Appendix A], with $\lambda = 1$ (published 25 years before [3], but not applied to squaring).*

Remark 2.2. *Using symmetries in our strategy reduces the computational cost compared to the resolution of Brent's equations [4, § 5, eq 5.03] with an undetermined tensor $\mathcal{T}$. In the previous example by doing so, we should have constructed a system of at most 64 algebraic equations with $4(3(7 - |J|) + 2|J|)$ unknowns, resulting from the constraints on $\mathcal{T}$ and the relation $\mathcal{T} = \mathcal{S}$, expressed using Kronecker product as a single zero matrix in $\mathbb{K}^{8\times 8}$.*

We apply now our strategy on the $2 \times 2$ matrix product $A \cdot A^\intercal$.

## 2.2 $2 \times 2$-matrix product by its transpose

Applying our Strategy 2.1, we consider (9) a generic matrix multiplication tensor $\mathcal{T}$ and our goal is to reduce the computational complexity of the partial contraction (5) with $A \otimes A^\intercal$ computing $A \cdot A^\intercal$.

By the properties of the transpose operator and the trace, the following relations hold:

$$\begin{aligned}\left\langle T_{i2}, A^\intercal \right\rangle &= \mathrm{Trace}\big(T_{i2}{}^\intercal \cdot A^\intercal\big) = \mathrm{Trace}\big((A \cdot T_{i2})^\intercal\big), \\ &= \mathrm{Trace}\big(A \cdot T_{i2}\big) = \mathrm{Trace}\big(T_{i2} \cdot A\big) = \left\langle T_{i2}{}^\intercal, A \right\rangle.\end{aligned} \tag{13}$$

Thus, the partial contraction (5) satisfies here the following relation:

$$\sum_{i=1}^{7} \big\langle T_{i1}, A \big\rangle \big\langle T_{i2}, A^\intercal \big\rangle T_{i3} = \sum_{i=1}^{7} \big\langle T_{i1}, A \big\rangle \langle T_{i2}{}^\intercal, A \rangle T_{i3}. \tag{14}$$

*2.2.1 Supplementary symmetry constraints.* Our goal is to save computations in the evaluation of (14). To do so, we consider the subsets $J$ of $\{1, \ldots, 7\}$ and $H$ of $\big\{(i,j) \in \{2, \ldots, 7\}^2 | i \neq j, i \notin J, j \notin J\big\}$ in order to express the following constraints:

$$T_{i1} = T_{i2}{}^\intercal,\ i \in J, \quad T_{j1} = T_{k2}{}^\intercal,\ T_{k1} = T_{j2}{}^\intercal,\ (j,k) \in H. \tag{15}$$

The constraints of type $J$ allow one to save preliminary additions when applying the method to matrices $B = A^\intercal$: since then operations on $A$ and $A^\intercal$ will be the same. The constraints of type $H$ allow to save multiplications especially when dealing with a block-matrix product: in fact, if some matrix products are transpose of another, only one of the pair needs to be computed as shown in Section 3.

We are thus looking for the largest possible sets $J$ and $H$. By exhaustive search, we conclude that the cardinality of $H$ is at most 2 and then the cardinality of $J$ is at most 3. For example, choosing the sets $J = \{1, 2, 5\}$ and $H = \{(3, 6), (4, 7)\}$ we obtain for these solutions the following parameterization expressed with a primitive element $z = v_{11} - v_{21}$:

$$\begin{aligned}
v_{11} &= z + v_{21}, \\
v_{22} &= \big(2\, v_{21}(v_{21} + z) - 1\big)v_{21} + z^3, \\
v_{12} &= -\big({v_{21}}^2 + (v_{21} + z^2)^2 + 1\big)v_{21} - z, \\
u_{11} &= -\big((z + v_{21})^2 + {v_{21}}^2\big)(w_{21} + w_{22}), \\
u_{21} &= -\big((z + v_{21})^2 + {v_{21}}^2\big)(w_{11} + w_{12}), \\
u_{12} &= -\big((z + v_{21})^2 + {v_{21}}^2\big)w_{22}, \\
u_{22} &= \big((z + v_{21})^2 + {v_{21}}^2\big)w_{12}, \\
\big((z + v_{21})^2 + {v_{21}}^2\big)^2 + 1 = 0,\ & w_{11}w_{22} - w_{12}w_{21} = 1.
\end{aligned} \tag{16}$$

Remark 2.3. *As $\big((z + v_{21})^2 + {v_{21}}^2\big)^2 + 1 = 0$ occurs in this parameterization, field extension could not be avoided in these algorithms if the field does not have—at least—a square root of $-1$. We show in Section 3 that we can avoid these extensions with block-matrix products and use our algorithm directly in any field of prime characteristic.*

*2.2.2 Supplementary constraint on the number of additions.* As done in Example 1, we could also try to reduce the additive complexity and use 4 pre-additions on $A$ (resp. $B$) [6, Lemma 9] and 7 post-additions on the products to form $C$ [6, Lemma 2]. In the current situation, if the operations on $B$ are exactly the transpose of that of $A$, then we have the following lower bound:

Lemma 2.1. *Over a non-commutative domain, 11 additive operations are necessary to multiply a $2 \times 2$ matrix by its transpose with a bilinear algorithm that uses 7 multiplications.*

Indeed, over a commutative domain, the lower left and upper right parts of the product are transpose of one another and one can save also multiplications. Differently, over non-commutative domains, $A \cdot A^\intercal$ is not symmetric in general (say $ac + bd \neq ca + db$) and all four coefficients need to be computed. But one can still save 4 additions, since there are algorithms where pre-additions are the same on $A$ and $A^\intercal$. Now, to reach that minimum, the constraints (15) must be combined with the minimal number 4 of pre-additions for $A$. Those can be attained only if 3 of the $T_{i1}$ factors do not require any addition [6, Lemma 8]. Hence, those factors involve only one of the four elements of $A$ and they are just permutations of $e_{11}$. We thus add these constraints to the system for a subset $K$ of $\{1, \ldots, 7\}$:

$$|K| = 3 \text{ and } T_{i1} \text{ is in } \left\{ \left(\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right), \left(\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right), \left(\begin{smallmatrix}0&0\\1&0\end{smallmatrix}\right), \left(\begin{smallmatrix}0&0\\0&1\end{smallmatrix}\right) \right\} \text{ and } i \text{ in } K. \tag{17}$$

*2.2.3 Selected solution.* We choose $K = \{1, 2, 3\}$ similar to (10) and obtain the following isotropy that sends Strassen tensor to an algorithm computing the symmetric product more efficiently:

$$\mathsf{a} = \left(\begin{smallmatrix}z^2&0\\0&z^2\end{smallmatrix}\right) \times \left(\begin{smallmatrix}z&-z\\0&z^3\end{smallmatrix}\right) \times \left(\begin{smallmatrix}-1&1\\1&0\end{smallmatrix}\right), \quad z^4 = -1. \tag{18}$$

We remark that $\mathsf{a}$ is equal to $\mathsf{d} \circ \mathsf{w}$ with $\mathsf{w}$ the isotropy (11) that sends Strassen tensor to Winograd tensor and with:

$$\mathsf{d} = D_1 \otimes D_2 \otimes D_3 = \left(\begin{smallmatrix}z^2&0\\0&z^2\end{smallmatrix}\right) \times \left(\begin{smallmatrix}z&0\\0&-z^3\end{smallmatrix}\right) \times \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right),\ z^4 = -1. \tag{19}$$

Hence, the induced algorithm can benefit from the scheduling and additive complexity of the classical Winograd algorithm. In fact, our choice $\mathsf{a} \diamond \mathcal{S}$ is equal to $(\mathsf{d} \circ \mathsf{w}) \diamond \mathcal{S}$ and thus, according to remark (2.1) the resulting algorithm expressed as the total contraction

$$\{(\mathsf{d} \circ \mathsf{w}) \diamond \mathcal{S}, (A \otimes A^\intercal \otimes C)\} = \{\mathsf{w} \diamond \mathcal{S}, d^\intercal \diamond (A \otimes A^\intercal \otimes C)\} \tag{20}$$

could be written as a slight modification of Algorithm 1 inputs.

Precisely, as $\mathsf{d}$'s components are diagonal, the relation $\mathsf{d}^\intercal = \mathsf{d}$ holds; hence, we could express input modification as:

$$\Big(D_1{}^{-1} \cdot A \cdot D_2\Big) \otimes \Big(D_2{}^{-1} \cdot A^\intercal \cdot D_3\Big) \otimes \Big(D_3{}^{-1} \cdot C \cdot D_1\Big). \tag{21}$$

The above expression is trilinear and the matrices $D_i$ are scalings of the identity for $i$ in $\{1, 3\}$, hence our modifications are just:

$$\left(\frac{1}{z^2} A \cdot D_2\right) \otimes \Big(D_2{}^{-1} \cdot A^\intercal\Big) \otimes z^2 C. \tag{22}$$

Using notations of Algorithm 1, this is $C = \mathrm{W}\big(A \cdot D_2, D_2{}^{-1} \cdot A^\intercal\big)$.

Allowing our isotropies to have determinant different from 1, we rescale $D_2$ by a factor $1/z$ to avoid useless 4th root as follows:

$$Q = \frac{D_2}{z} = \left(\begin{smallmatrix}1&0\\0&-z^2\end{smallmatrix}\right) = \left(\begin{smallmatrix}1&0\\0&-y\end{smallmatrix}\right), \quad z^4 = -1 \tag{23}$$

where $y$ designates the expression $z^2$ that is a root of $-1$. Hence, our algorithm to compute the symmetric product is:

$$C = \mathrm{W}\left(A \cdot \frac{D_2}{z}, \left(\frac{D_2}{z}\right)^{-1} \cdot A^\intercal\right) = \mathrm{W}\Big(A \cdot Q, \Big(A \cdot (Q^{-1})^\intercal\Big)^\intercal\Big). \tag{24}$$

In the next sections, we describe and extend this algorithm to higher-dimensional symmetric products $A \cdot A^\intercal$ with a $2^\ell m \times 2^\ell m$ matrix $A$.

## 3 FAST $2 \times 2$-BLOCK RECURSIVE SYRK

The algorithm presented in the previous section is non-commutative and thus we can extend it to higher-dimensional matrix product by a divide and conquer approach. To do so, we use in the sequel upper case letters for coefficients in our algorithms instead of lower case previously (since these coefficients now represent matrices). Thus, new properties and results are induced by this shift of perspective. For example, the coefficient $Y$ introduced in (23) could now be transposed in (24); that leads to the following definition:

Definition 3.1. *An invertible matrix is* skew-orthogonal *if the following relation* $Y^\top = -Y^{-1}$ *holds.*

If $Y$ is skew-orthogonal, then of the 7 recursive matrix products involved in expression (24): 1 can be avoided ($P_6$) since we do not need the upper right coefficient anymore, 1 can be avoided since it is the transposition of another product ($P_7 = {P_4}^\top$) and 3 are recursive calls to SYRK. This results in Algorithm 2.

**Algorithm 2** `syrk`: symmetric matrix product

**Input:** $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$; a skew-orthogonal matrix $Y$.

**Output:** The lower left triangular part of $C = A \cdot A^\top = \begin{pmatrix} C_{11} & {C_{21}}^\top \\ C_{21} & C_{22} \end{pmatrix}$.

▷ *4 additions and 2 multiplications by $Y$:*

$S_1 \leftarrow (A_{21} - A_{11}) \cdot Y$, $S_2 \leftarrow A_{22} - A_{21} \cdot Y$,
$S_3 \leftarrow S_1 - A_{22}$, $S_4 \leftarrow S_3 + A_{12}$.

▷ *3 recursive SYRK ($P_1$, $P_2$, $P_5$) and 2 generic ($P_3$, $P_4$) products:*

$P_1 \leftarrow A_{11} \cdot {A_{11}}^\top$, $P_2 \leftarrow A_{12} \cdot {A_{12}}^\top$,
$P_3 \leftarrow A_{22} \cdot {S_4}^\top$, $P_4 \leftarrow S_1 \cdot {S_2}^\top$, $P_5 \leftarrow S_3 \cdot {S_3}^\top$.

▷ *2 symmetric additions (half additions):*

$\mathrm{Low}(U_1) \leftarrow \mathrm{Low}(P_1) + \mathrm{Low}(P_5)$, ▷ *$U_1$, $P_1$, $P_5$ are symm.*
$\mathrm{Low}(U_3) \leftarrow \mathrm{Low}(P_1) + \mathrm{Low}(P_2)$, ▷ *$U_3$, $P_1$, $P_2$ are symm.*

▷ *2 complete additions ($P_4$ and $P_3$ are not symmetric):*

$\mathrm{Up}(U_1) \leftarrow \mathrm{Low}(U_1)^\top$, $U_2 \leftarrow U_1 + P_4$, $U_4 \leftarrow U_2 + P_3$,

▷ *1 half addition ($U_5 = U_1 + P_4 + {P_4}^\top$ is symmetric):*

$\mathrm{Low}(U_5) \leftarrow \mathrm{Low}(U_2) + \mathrm{Low}({P_4}^\top)$.

**return** $\begin{pmatrix} \mathrm{Low}(U_3) & \\ U_4 & \mathrm{Low}(U_5) \end{pmatrix}$.

Proposition 3.1 (Appendix A.1). *Algorithm 2 is correct for any skew-orthogonal matrix $Y$.*

### 3.1 Skew orthogonal matrices

Algorithm 2 requires a skew-orthogonal matrix. Unfortunately there are no skew-orthogonal matrices over $\mathbb{R}$, nor $\mathbb{Q}$. Hence, we report no improvement in these cases. In other domains, the simplest skew-orthogonal matrices just use a square root of $-1$.

*3.1.1 Over the complex field.* Therefore Algorithm 2 is directly usable over $\mathbb{C}^{n\times n}$ with $Y = i\,\mathrm{I}_n \in \mathbb{C}^{n\times n}$. Further, usually, complex numbers are emulated by a pair of floats so then the multiplications by $Y = i\,\mathrm{I}_n$ are essentially free since they just exchange the real and imaginary parts, with one sign flipping. Even though over the complex the product ZHERK of a matrix by its *conjugate* transpose is more widely used, ZSYRK has some applications, see for instance [1].

*3.1.2 Negative one is a square.* Over some fields with prime characteristic, square roots of $-1$ can be elements of the base field, denoted $i$ in $\mathbb{F}$ again. There, Algorithm 2 only requires some pre-multiplications by this square root (with also $Y = i\,\mathrm{I}_n \in \mathbb{F}^{n\times n}$), but *within the field.* Proposition 3.2 thereafter characterizes these fields.

Proposition 3.2. *Fields with characteristic two, or with an odd characteristic $p \equiv 1 \bmod 4$, or finite fields that are an even extension, contain a square root of $-1$.*

Proof. If $p = 2$, then $1 = 1^2 = -1$. If $p \equiv 1 \bmod 4$, then half of the non-zero elements $x$ in the base field of size $p$ satisfy $x^{\frac{p-1}{4}} \neq \pm 1$ and then the square of the latter must be $-1$. If the finite field $\mathbb{F}$ is of cardinality $p^{2k}$, then, similarly, there exists elements $x^{\frac{p^k-1}{2}\frac{p^k+1}{2}}$ different from $\pm 1$ and then the square of the latter must be $-1$. □

*3.1.3 Any field with prime characteristic.* Finally, we show that Algorithm 2 can also be run without any field extension, even when $-1$ is not a square: form the skew-orthogonal matrices constructed in Proposition 3.3, thereafter, and use them directly as long as the dimension of $Y$ is even. Whenever this dimension is odd, it is always possible to pad with zeroes so that $A \cdot A^\top = \begin{pmatrix} A & 0 \end{pmatrix} \cdot \begin{pmatrix} A^\top \\ 0 \end{pmatrix}$.

Proposition 3.3. *Let $\mathbb{F}$ be a field of characteristic $p$, there exists $(a, b)$ in $\mathbb{F}^2$ such that the matrix:*

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \otimes \mathrm{I}_n = \begin{pmatrix} a\,\mathrm{I}_n & b\,\mathrm{I}_n \\ -b\,\mathrm{I}_n & a\,\mathrm{I}_n \end{pmatrix} \quad \text{in} \quad \mathbb{F}^{2n\times 2n} \tag{25}$$

*is skew-orthogonal.*

Proof. Using the relation

$$\begin{pmatrix} a\,\mathrm{I}_n & b\,\mathrm{I}_n \\ -b\,\mathrm{I}_n & a\,\mathrm{I}_n \end{pmatrix} \begin{pmatrix} a\,\mathrm{I}_n & b\,\mathrm{I}_n \\ -b\,\mathrm{I}_n & a\,\mathrm{I}_n \end{pmatrix}^\top = (a^2 + b^2)\,\mathrm{I}_{2n}, \tag{26}$$

it suffices to prove that there exist $a, b$ such that $a^2 + b^2 = -1$. In characteristic 2, $a = 1, b = 0$ is a solution as $1^2 + 0^2 = -1$. In odd characteristic, there are $\frac{p+1}{2}$ distinct square elements ${x_i}^2$ in the base prime field. Therefore, there are $\frac{p+1}{2}$ distinct elements $-1 - {x_i}^2$. But there are only $p$ distinct elements in the base field, thus there exists a couple $(i, j)$ such that $-1 - {x_i}^2 = {x_j}^2$ [19, Lemma 6]. □

Proposition 3.3 shows that skew-orthogonal matrices do exist for any field with prime characteristic. For Algorithm 2, we need to build them mostly for $p \equiv 3 \bmod 4$ (otherwise use Proposition 3.2).

For this, without the extended Riemann hypothesis (ERH), it is possible to use the decomposition of primes into squares:

(1) Compute first a prime $r = 4pk + (3-1)p - 1$, then the relations $r \equiv 1 \bmod 4$ and $r \equiv -1 \bmod p$ hold;
(2) Thus, results of [5] allow one to decompose primes into squares and give a couple $(a, b)$ in $\mathbb{Z}^2$ such that $a^2 + b^2 = r$. Finally, we get $a^2 + b^2 \equiv -1 \bmod p$.

By the prime number theorem the first step is polynomial in $\log(p)$, as is the second step (square root modulo a prime, denoted `sqrt`, has a cost close to exponentiation and then the rest of Brillhart's algorithm is GCD-like). In practice, though, it is faster to use the following Algorithm 3, even though the latter has a better asymptotic complexity bound only if the ERH is true.

**Algorithm 3** SoS: Sum of squares decomposition over a finite field

**Input:** $p \in \mathbb{P}\backslash\{2\}$, $k \in \mathbb{Z}$.
**Output:** $(a, b) \in \mathbb{Z}^2$, s.t. $a^2 + b^2 \equiv k \bmod p$.

1: **if** $\left(\frac{k}{p}\right) = 1$ **then** ▷ *$k$ is a square mod $p$*
2: **return** $(\mathtt{sqrt}(k), 0)$.
3: **else** ▷ *Find smallest quadratic non-residue*
4: $s \leftarrow 2$; **while** $\left(\frac{s}{p}\right) == 1$ **do** $s \leftarrow s + 1$
5: $c \leftarrow \mathtt{sqrt}(s-1)$ ▷ *$s-1$ must be a square*
6: $r \leftarrow ks^{-1} \bmod p$
7: $a \leftarrow \mathtt{sqrt}(r)$ ▷ *Now $k \equiv a^2 s \equiv a^2(1+c^2) \bmod p$*
8: **return** $(a, ac \bmod p)$

Proposition 3.4. *Algorithm 3 is correct and, under the ERH, runs in expected time $\widetilde{O}(\log^3(p))$.*

Proof. If $k$ is square then the square of one of its square roots added to the square of zero is a solution. Otherwise, the lowest quadratic non-residue (LQNR) modulo $p$ is one plus a square $b^2$ (1 is always a square so the LQNR is larger than 2). For any generator of $\mathbb{Z}_p$, quadratic non-residues, as well as their inverses ($s$ is invertible as it is non-zero and $p$ is prime), have an odd discrete logarithm. Therefore the multiplication of $k$ and the inverse of the LQNR must be a square $a^2$. This means that the relation $k = a^2(1 + b^2) = a^2 + (ab)^2$ holds. Now for the running time, under ERH, the LQNR should be lower than $3\log^2(p)/2 - 44\log(p)/5 + 13$ [21, Theorem 6.35]. The expected number of Legendre symbol computations is $O(\log^2(p))$ and this dominates the modular square root computations. □

Remark 3.1. *Another possibility is to use randomization: instead of using the lowest quadratic non-residue (LQNR), randomly select a non-residue $s$, and then decrement it until $s-1$ is a quadratic residue ($1$ is a square so this will terminate)*[1]*. Also, when computing $t$ sum of squares modulo the same prime, one can compute the LQNR* only once *to get all the sum of squares with an expected cost bounded by $\widetilde{O}(\log^3(p) + t\log^2(p))$.*

Remark 3.2. *Except in characteristic 2 or in algebraic closures, where every element is a square anyway, Algorithm 3 is easily extended over any finite field: compute the LQNR in the base prime field, then use Tonelli-Shanks or Cipolla-Lehmer algorithm to compute square roots in the extension field.*

*Denote by $\mathtt{SoS}(q, k)$ this algorithm decomposing $k$ as a sum of squares within any finite field $\mathbb{F}_q$. This is not always possible over infinite fields, but there Algorithm 3 still works anyway for the special case $k = -1$: just run it in the prime subfield.*

## 3.2 Conjugate transpose

Note that Algorithm 2 remains valid if transposition is replaced by *conjugate transposition*, provided that there exists a matrix $Y$ such that $Y \cdot \overline{Y}^{\mathsf{T}} = -\mathrm{I}$. This is not possible anymore over the complex field, but works for any even extension field, thanks to Algorithm 3: if $-1$ is a square in $\mathbb{F}_q$, then $Y = \sqrt{-1} \cdot \mathrm{I}_n$ still works; otherwise there exists a square root $i$ of $-1$ in $\mathbb{F}_{q^2}$, from Proposition 3.2. In the latter case, thus build $(a, b)$, both in $\mathbb{F}_q$, such that $a^2 + b^2 = -1$. Now $Y = (a + ib) \cdot \mathrm{I}_n$ in ${\mathbb{F}_{q^2}}^{n\times n}$ is appropriate: indeed, since $q \equiv 3 \bmod 4$, we have that $\overline{a + ib} = (a + ib)^q = a - ib$.

[1] In practice, the running time seems very close to that of Algorithm 3 anyway, see, e.g. the implementation in Givaro rev. 7bdefe6, https://github.com/linbox-team/givaro.

# 4 ANALYSIS AND IMPLEMENTATION

## 4.1 Complexity bounds

Theorem 4.1. *Algorithm 2 requires $\frac{2}{2^\omega - 3} C_\omega n^\omega + o(n^\omega)$ field operations, over $\mathbb{C}$ or over any field with prime characteristic.*

Proof. Algorithm 2 is applied recursively to compute three products $P_1, P_2$ and $P_7$, while $P_4$ and $P_5$ are computed in $\mathrm{MM}_\omega(n) = C_\omega n^\omega + o(n^\omega)$ using a general matrix multiplication algorithm. We will show that applying the skew-orthogonal matrix $Y$ to a $n \times n$ matrix costs $yn^2$ for some constant $y$ depending on the base field. Then applying Remark 4.1 thereafter, the cost $T(n)$ of Algorithm 2 satisfies:

$$T(n) \le 3T(n/2) + 2C_\omega(n/2)^\omega + (7.5 + 2y)(n/2)^2 + o\left(n^2\right) \quad (27)$$

and $T(4)$ is a constant. Thus, by the master Theorem:

$$T(n) \le \frac{2C_\omega}{2^\omega - 3} n^\omega + o\left(n^\omega\right) = \frac{2}{2^\omega - 3}\mathrm{MM}_\omega(n) + o\left(n^\omega\right). \quad (28)$$

If the field is $\mathbb{C}$ or satisfies the conditions of Proposition 3.2, there is a square root $i$ of $-1$. Setting $Y = i\,\mathrm{I}_{n/2}$ yields $y = 1$. Otherwise, in characteristic $p \equiv 3 \bmod 4$, Proposition 3.3 produces $Y$ equal to $\left(\begin{smallmatrix} a & b \\ -b & a \end{smallmatrix}\right) \otimes \mathrm{I}_{n/2}$ for which $y = 3$. As a subcase, the latter can be improved when $p \equiv 3 \bmod 8$: then $-2$ is a square (indeed, $\left(\frac{-2}{p}\right) = \left(\frac{-1}{p}\right)\left(\frac{2}{p}\right) = (-1)^{\frac{p-1}{2}}(-1)^{\frac{p^2-1}{8}} = (-1)(-1) = 1$). Therefore, in this case set $a = 1$ and $b \equiv \sqrt{-2} \bmod p$ such that the relation $a^2 + b^2 = -1$ yields $Y = \left(\begin{smallmatrix} 1 & \sqrt{-2} \\ -\sqrt{-2} & 1 \end{smallmatrix}\right) \otimes \mathrm{I}_{n/2}$ for which $y = 2$. □

To our knowledge, the best previously known result was with a $\frac{2}{2^\omega - 4}$ factor instead, see e.g. [11, § 6.3.1]. Table 1 summarizes the arithmetic complexity bound improvements.

| Problem | Alg. | $O(n^3)$ | $O\left(n^{\log_2(7)}\right)$ | $O(n^\omega)$ |
|---|---|---|---|---|
| $A \cdot A^{\mathsf{T}} \in \mathbb{F}^{n\times n}$ | [11] | $n^3$ | $\frac{2}{3}\,\mathrm{MM}_{\log_2(7)}(n)$ | $\frac{2}{2^\omega - 4}\,\mathrm{MM}_\omega(n)$ |
| | Alg. 2 | $0.8n^3$ | $\frac{1}{2}\,\mathrm{MM}_{\log_2(7)}(n)$ | $\frac{2}{2^\omega - 3}\,\mathrm{MM}_\omega(n)$ |

**Table 1: Arithmetic complexity bounds leading terms.**

Alternatively, over $\mathbb{C}$, the $3M$ method (Karatsuba) for non-symmetric matrix multiplication reduces the number of multiplications of real matrices from 4 to 3 [15]: if $RR_\omega(n)$ is the cost of multiplying $n \times n$ matrices over $\mathbb{R}$, then the $3M$ method costs $3RR_\omega(n) + o(n^\omega)$ operations over $\mathbb{R}$. Adapting this approach to the symmetric case yields a $2M$ method to compute the product of a complex matrix by its transpose, using only 2 real products: $H = A \cdot B^{\mathsf{T}}$ and $G = (A + B) \cdot (A^{\mathsf{T}} - B^{\mathsf{T}})$. Combining those into $(G - H^{\mathsf{T}} + H) + i(H + H^{\mathsf{T}})$, yields the product $(A + iB) \cdot (A^{\mathsf{T}} + iB^{\mathsf{T}})$. This approach costs $2RR_\omega + o(n^\omega)$ operations in $\mathbb{R}$.

Classical algorithm [11, § 6.3.1] applies a divide and conquer approach directly on the complex field. This would use only the equivalent of $\frac{2}{2^\omega - 4}$ complex floating point $n \times n$ products. Using

the $3M$ method for the complex products, this algorithm uses overall $\frac{6}{2^\omega-4}RR_\omega + o(n^\omega)$ operations in $\mathbb{R}$. Finally, Algorithm 2 only costs $\frac{2}{2^\omega-3}$ complex multiplications for a leading term bounded by $\frac{6}{2^\omega-3}\mathrm{RR}_\omega$, better than $2\mathrm{RR}_\omega$ for $\omega > \log_2(6) \approx 2.585$. This is summarized in Table 2, replacing $\omega$ by 3 or $\log_2(7)$.

| Problem | Alg. | $\mathrm{MM}_3(n)$ | $\mathrm{MM}_{\log_2 7}(n)$ | $\mathrm{MM}_\omega(n)$ |
|---|---|---|---|---|
| $A \cdot B \in \mathbb{C}^{n\times n}$ | naive | $8n^3$ | $4\,\mathrm{RR}_{\log_2(7)}(n)$ | $4\,\mathrm{RR}_\omega(n)$ |
| | 3M | $6n^3$ | $3\,\mathrm{RR}_{\log_2(7)}(n)$ | $3\,\mathrm{RR}_\omega(n)$ |
| $A \cdot A^\intercal \in \mathbb{C}^{n\times n}$ | 2M | $4n^3$ | $2\,\mathrm{RR}_{\log_2(7)}(n)$ | $2\,\mathrm{RR}_\omega(n)$ |
| | [11] | $3n^3$ | $2\,\mathrm{RR}_{\log_2(7)}(n)$ | $\frac{6}{2^\omega-4}\,\mathrm{RR}_\omega(n)$ |
| | Alg. 2 | $2.4n^3$ | $\frac{3}{2}\,\mathrm{RR}_{\log_2(7)}(n)$ | $\frac{6}{2^\omega-3}\,\mathrm{RR}_\omega(n)$ |

**Table 2: Symmetric multiplication over $\mathbb{C}$: leading term of the cost in number of operations over $\mathbb{R}$.**

Remark 4.1. *Each recursive level of Algorithm 2 is composed of 9 block additions. An exhaustive search on all symmetric algorithms derived from Strassen's showed that this number is minimal in this class of algorithms. Note also that 3 out of these 9 additions in Algorithm 2 involve symmetric matrices and are therefore only performed on the lower triangular part of the matrix. Overall, the number of scalar additions is $6n^2 + 3/2n(n+1) = 15/2n^2 + 1.5n$, nearly half of the optimal in the non-symmetric case [6, Theorem 1].*

To further reduce the number of additions, a promising approach is that undertaken in [2, 16]. This is however not clear to us how to adapt our strategy to their recursive transformation of basis.

## 4.2 Implementation and scheduling

This section reports on an implementation of Algorithm 2 over prime fields. We propose in Table 3 and Figure 1 a schedule for the operation $C \leftarrow A \cdot A^\intercal$ using no more extra storage than the unused upper triangular part of the result $C$.

| # | operation | loc. | # | operation | loc. |
|---|---|---|---|---|---|
| 1 | $S_1 = (A_{21} - A_{11}) \cdot Y$ | $C_{21}$ | 9 | $U_1 = P_1 + P_5$ | $C_{12}$ |
| 2 | $S_2 = A_{22} - A_{21} \cdot Y$ | $C_{12}$ | | $\mathrm{Up}(U_1) = \mathrm{Low}(U_1)^\intercal$ | $C_{12}$ |
| 3 | $P_4{}^\intercal = S_2 \cdot S_1{}^\intercal$ | $C_{22}$ | 10 | $U_2 = U_1 + P_4$ | $C_{12}$ |
| 4 | $S_3 = S_1 - A_{22}$ | $C_{21}$ | 11 | $U_4 = U_2 + P_3$ | $C_{21}$ |
| 5 | $P_5 = S_3 \cdot S_3{}^\intercal$ | $C_{12}$ | 12 | $U_5 = U_2 + P_4{}^\intercal$ | $C_{22}$ |
| 6 | $S_4 = S_3 + A_{12}$ | $C_{11}$ | 13 | $P_2 = A_{12} \cdot A_{12}{}^\intercal$ | $C_{12}$ |
| 7 | $P_3 = A_{22} \cdot S_4{}^\intercal$ | $C_{21}$ | 14 | $U_3 = P_1 + P_2$ | $C_{11}$ |
| 8 | $P_1 = A_{11} \cdot A_{11}{}^\intercal$ | $C_{11}$ | | | |

**Table 3: Memory placement and schedule of tasks to compute the lower triangular part of $C \leftarrow A \cdot A^\intercal$ when $k \le n$. The block $C_{12}$ of the output matrix is the only temporary used.**

For the more general operation $C \leftarrow \alpha A \cdot A^\intercal + \beta C$, Table 4 and Figure 2 propose a schedule requiring only an additional $n/2 \times n/2$ temporary storage. These algorithms have been implemented as the `fsyrk` routine in the `fflas-ffpack` library for dense linear algebra over a finite field [14, from commit 0a91d61e].

Figure 3 compares the computation speed in effective Gfops (defined as $n^3/(10^9 \times \text{time})$) of this implementation over $\mathbb{Z}/131071\mathbb{Z}$

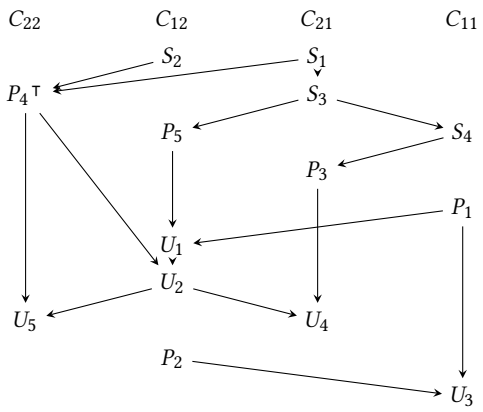


**Figure 1: DAG of the tasks and their memory location for the computation of $C \leftarrow A \cdot A^\intercal$ presented in Table 3.**

| operation | loc. | operation | loc. |
|---|---|---|---|
| $S_1 = (A_{21} - A_{11}) \cdot Y$ | tmp | $P_1 = \alpha A_{11} \cdot A_{11}{}^\intercal$ | tmp |
| $S_2 = A_{22} - A_{21} \cdot Y$ | $C_{12}$ | $U_1 = P_1 + P_5$ | $C_{12}$ |
| $\mathrm{Up}(C_{11}) = \mathrm{Low}(C_{22})^\intercal$ | $C_{11}$ | $\mathrm{Up}(U_1) = \mathrm{Low}(U_1)^\intercal$ | $C_{12}$ |
| $P_4{}^\intercal = \alpha S_2 \cdot S_1{}^\intercal$ | $C_{22}$ | $U_2 = U_1 + P_4$ | $C_{12}$ |
| $S_3 = S_1 - A_{22}$ | tmp | $U_4 = U_2 + P_3$ | $C_{21}$ |
| $P_5 = \alpha S_3 \cdot S_3{}^\intercal$ | $C_{12}$ | $U_5 = U_2 + P_4{}^\intercal + \beta \mathrm{Up}(C_{11})^\intercal$ | $C_{22}$ |
| $S_4 = S_3 + A_{12}$ | tmp | $P_2 = \alpha A_{12} \cdot A_{12}{}^\intercal + \beta C_{11}$ | $C_{11}$ |
| $P_3 = \alpha A_{22} \cdot S_4{}^\intercal + \beta C_{21}$ | $C_{21}$ | $U_3 = P_1 + P_2$ | $C_{11}$ |

**Table 4: Memory placement and schedule of tasks to compute the lower triangular part of $C \leftarrow \alpha A \cdot A^\intercal + \beta C$ when $k \le n$. The block $C_{12}$ of the output matrix as well as an $n/2 \times n/2$ block tmp are used as temporary storages.**

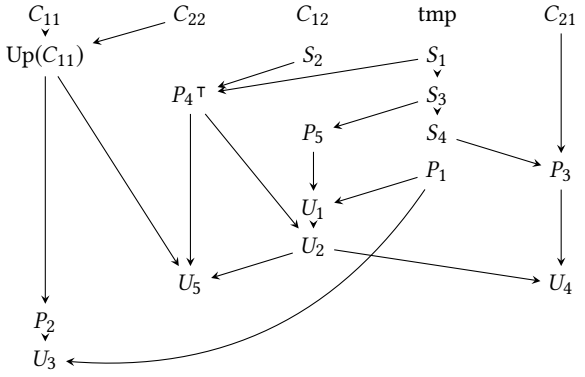


**Figure 2: DAG of the tasks and their memory location for the computation of $C \leftarrow \alpha A \cdot A^\intercal + \beta C$ presented in Table 4.**

with that of the double precision BLAS routines `dsyrk`, the classical cubic-time routine over a finite field (calling `dsyrk` and performing modular reductions on the result), and the classical divide and conquer algorithm [11, § 6.3.1]. The `fflas-ffpack` library is linked with OpenBLAS [23, v0.3.6] and compiled with `gcc-9.2` on an Intel skylake i7-6700 running a Debian GNU/Linux system (v5.2.17).

The slight overhead of performing the modular reductions is quickly compensated by the speed-up of the sub-cubic algorithm

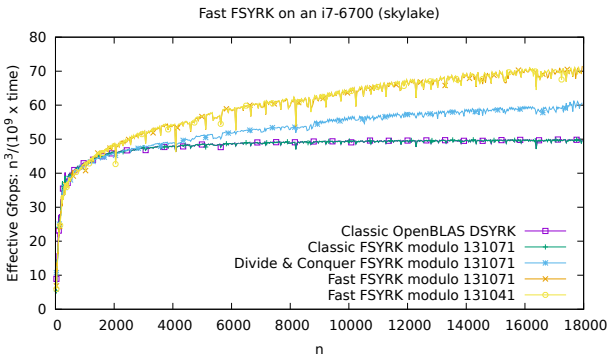


**Figure 3: Speed of an implementation of Algorithm 2**

(the threshold for a first recursive call is near $n = 2000$). The classical divide and conquer approach also speeds up the classical algorithm, but starting from a larger threshold, and hence at a slower pace. Lastly, the speed is merely identical modulo 131041, where square roots of $-1$ exist, thus showing the limited overhead of the preconditioning by the matrix $Y$.

## 5 SYRK WITH BLOCK DIAGONAL SCALING

Symmetric rank k updates are a key building block for symmetric triangular factorization algorithms, for their efficiency is one of the bottlenecks. In the most general setting (indefinite factorization), a block diagonal scaling by a matrix $D$, with 1 or 2 dimensional diagonal blocks, has to be inserted within the product, leading to the operation: $C \leftarrow C - A \cdot D \cdot A^\intercal$.

Handling the block diagonal structure over the course of the recursive algorithm may become tedious and quite expensive. For instance, a $2 \times 2$ diagonal block might have to be cut by a recursive split. We will see also in the following that non-squares in the diagonal need to be dealt with in pairs. In both cases it might be necessary to add a column to deal with these cases: this is potentially $O(\log_2(n))$ extra columns in a recursive setting.

Over a finite field, though, we will show in this section, how to factor the block-diagonal matrix $D$ into $D = \Delta \cdot \Delta^\intercal$, *without needing any field extension*, and then compute instead $(A \cdot \Delta) \cdot (A \cdot \Delta)^\intercal$. Algorithm 6, deals with non-squares and $2 \times 2$ blocks only once beforehand, introducing no more than 2 extra-columns overall. Section 5.1 shows how to factor a diagonal matrix, without resorting to field extensions for non-squares. Then Sections 5.2.1 and 5.2.2 show how to deal with the $2 \times 2$ blocks depending on the characteristic.

### 5.1 Factoring non-squares within a finite field

First we give an algorithm handling pairs of non-quadratic residues.

Proposition 5.1. *Algorithm 4 is correct.*

Proof. Given $\alpha$ and $\beta$ quadratic non-residues, the couple $(a, b)$, such that $\alpha = a^2 + b^2$, is found by the algorithm of Remark 3.2. Second, as $\alpha$ and $\beta$ are quadratic non-residues, over a finite field their quotient is a residue since: $\left(\beta\alpha^{-1}\right)^{\frac{q-1}{2}} = \frac{-1}{-1} = 1$. Third, if $c$ denotes $-bda^{-1}$ then $c^2 + d^2$ is equal to $(-bd/a)^2 + d^2$ and thus to $(b^2/a^2 + 1)d^2$; this last quantity is equal to $(\alpha)d^2/a^2$ and then

**Algorithm 4** : `nrsyf`: Sym. factorization. of a pair of non-residues

**Input:** $(\alpha, \beta) \in {\mathbb{F}_q}^2$, both being quadratic non-residues.
**Output:** $Y \in {\mathbb{F}_q}^{2\times 2}$, s.t. $Y \cdot Y^\intercal = \left(\begin{smallmatrix} \alpha & 0 \\ 0 & \beta \end{smallmatrix}\right)$.

1: $(a, b) \leftarrow \mathsf{SoS}(q, \alpha)$; ▷ $\alpha = a^2 + b^2$
2: $d \leftarrow a\,\mathtt{sqrt}(\beta\alpha^{-1})$; ▷ $d^2 = a^2\beta\alpha^{-1}$
3: $c \leftarrow -bda^{-1}$; ▷ $ac + bd = 0$
4: **return** $Y \leftarrow \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$.

to $\alpha(a\sqrt{\beta/\alpha})^2/a^2 = \alpha(a^2\beta/\alpha)/a^2 = \beta$. Fourth, $a$ (or w.l.o.g. $b$) is invertible. Indeed, $\alpha$ is not a square, therefore it is non-zero and thus one of $a$ or $b$ must be non-zero. Finally, we obtain the cancellation $ac + bd = a(-dba^{-1}) + bd = -db + bd = 0$ and the matrix product $Y \cdot Y^\intercal$ is $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)\left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right) = \left(\begin{smallmatrix} a^2+b^2 & ac+bd \\ ac+bd & c^2+d^2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} \alpha & 0 \\ 0 & \beta \end{smallmatrix}\right)$. □

Using Algorithm 4, one can then factor any diagonal matrix within a finite field as a symmetric product with a tridiagonal matrix. This can then be used to compute efficiently $A \cdot D \cdot A^\intercal$ with $D$ a diagonal matrix: factor $D$ with a tridiagonal matrix $D = \Delta \cdot \Delta^\intercal$, then pre-multiply $A$ by this tridiagonal matrix and run a fast symmetric product on the resulting matrix. This is shown in Algorithm 5, where the overhead, compared to simple matrix multiplication, is only $O(n^2)$ (that is $O(n)$ square roots and $O(n)$ column scalings).

**Algorithm 5** `syrkd`: sym. matrix product with diagonal scaling

**Input:** $A \in {\mathbb{F}_q}^{m\times n}$ and $D = \mathrm{Diag}(d_1, \ldots, d_n) \in {\mathbb{F}_q}^{n\times n}$
**Output:** $A \cdot D \cdot A^\intercal$ in ${\mathbb{F}_q}^{m\times m}$

1: **if** number of quadratic non-residues in $\{d_1, \ldots, d_n\}$ is odd **then** Let $d_\ell$ be one of the quadratic non-residues
2: $\bar{D} \leftarrow \mathrm{Diag}(d_1, \ldots, d_n, d_\ell) \in {\mathbb{F}_q}^{(n+1)\times(n+1)}$
3: $\bar{A} \leftarrow (\,A\ 0\,) \in {\mathbb{F}_q}^{m\times(n+1)}$ ▷ *Augment $A$ with a zero column*
4: **else**
5: $\bar{D} \leftarrow \mathrm{Diag}(d_1, \ldots, d_n) \in {\mathbb{F}_q}^{n\times n}$
6: $\bar{A} \leftarrow A \in {\mathbb{F}_q}^{m\times n}$
7: **for all** quadratic residues $d_j$ in $\bar{D}$ **do**
8: $\bar{A}_{*,j} \leftarrow \mathtt{sqrt}(d_j) \cdot \bar{A}_{*,j}$ ▷ *Scale col. $j$ of $\bar{A}$ by a sq. root of $d_j$*
9: **for all** distinct pairs of quadratic non-residues $(d_i, d_j)$ in $\bar{D}$ **do**
10: $\Delta \leftarrow \mathtt{nrsyf}(d_i, d_j)$ ▷ $\Delta \cdot \Delta^\intercal = \left(\begin{smallmatrix} d_i & 0 \\ 0 & d_j \end{smallmatrix}\right)$ *using Algorithm 4*
11: $(\,\bar{A}_{*,i}\ \bar{A}_{*,j}\,) \leftarrow (\,\bar{A}_{*,i}\ \bar{A}_{*,j}\,) \cdot \Delta$;
12: **return** $\mathtt{syrk}(\bar{A})$ ▷ $\bar{A} \cdot \bar{A}^\intercal$ *using Algorithm 2*

### 5.2 Antidiagonal and antitriangular blocks

In general, an $L \cdot D \cdot L^\intercal$ factorization may have antitriangular or antidiagonal blocks in $D$ [12]. In order to reduce to a routine for fast symmetric multiplication with diagonal scaling, these blocks need to be processed once for all, which is what this section is about.

*5.2.1 Antidiagonal blocks in odd characteristic.* In odd characteristic, the 2-dimensional blocks in an $L \cdot D \cdot L^\intercal$ factorization are only of the form $\left(\begin{smallmatrix} 0 & \beta \\ \beta & 0 \end{smallmatrix}\right)$, and always have the symmetric factorization:

$$\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)\left(\begin{smallmatrix} \frac{1}{2}\beta & 0 \\ 0 & -\frac{1}{2}\beta \end{smallmatrix}\right)\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)^\intercal = \left(\begin{smallmatrix} 0 & \beta \\ \beta & 0 \end{smallmatrix}\right). \tag{29}$$

This shows the reduction to the diagonal case (note the requirement that 2 is invertible).

*5.2.2 Antitriangular blocks in characteristic 2.* In characteristic 2, some $2 \times 2$ blocks might not be reduced further than an antitriangular form: $\left(\begin{smallmatrix} 0 & \beta \\ \beta & \gamma \end{smallmatrix}\right)$, with $\gamma \neq 0$.

In characteristic 2 every element is a square, therefore those antitriangular blocks can be factored as shown in Eq. (30):

$$\left(\begin{smallmatrix} 0 & \beta \\ \beta & \gamma \end{smallmatrix}\right) = \left(\left(\begin{smallmatrix} \beta\gamma^{-1/2} & 0 \\ 0 & \gamma^{1/2} \end{smallmatrix}\right)\left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right)\right)\left(\left(\begin{smallmatrix} \beta\gamma^{-1/2} & 0 \\ 0 & \gamma^{1/2} \end{smallmatrix}\right)\left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right)\right)^{\mathsf{T}}. \quad (30)$$

Therefore the antitriangular blocks also reduce to the diagonal case.

*5.2.3 Antidiagonal blocks in characteristic 2.* The symmetric factorization in this case might require an extra row or column [18] as shown in Eq. (31):

$$\left(\begin{smallmatrix} 1 & 0 \\ 0 & \beta \end{smallmatrix}\right)\left(\begin{smallmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right)\left(\left(\begin{smallmatrix} 1 & 0 \\ 0 & \beta \end{smallmatrix}\right)\left(\begin{smallmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right)\right)^{\mathsf{T}} = \left(\begin{smallmatrix} 0 & \beta \\ \beta & 0 \end{smallmatrix}\right) \bmod 2. \quad (31)$$

A first option is to augment $A$ by one column for each antidiagonal block, by applying the 2×3 factor in Eq. (31). However one can instead combine a diagonal element, say $x$, and an antidiagonal block as shown in Eq. (32).

$$\left(\begin{smallmatrix} \sqrt{x} & \sqrt{x} & \sqrt{x} \\ 1 & 0 & 1 \\ 0 & \beta & \beta \end{smallmatrix}\right)\left(\begin{smallmatrix} \sqrt{x} & \sqrt{x} & \sqrt{x} \\ 1 & 0 & 1 \\ 0 & \beta & \beta \end{smallmatrix}\right)^{\mathsf{T}} = \left(\begin{smallmatrix} x & 0 & 0 \\ 0 & 0 & \beta \\ 0 & \beta & 0 \end{smallmatrix}\right) \bmod 2. \quad (32)$$

Hence, any antidiagonal block can be combined with any 1×1 block to form a symmetric factorization.

There remains the case when there are no 1×1 blocks. Then, one can use Eq. (31) once, on the first antidiagonal block, and add column to $A$. This indeed extracts the antidiagonal elements and creates a 3×3 identity block in the middle. Any one of its three ones can then be used as $x$ in a further combination with the next antidiagonal blocks. Algorithm 6 sums up the use of Eqs. (29) to (32).

**Algorithm 6** : `syrkbd`: sym. matrix product with block diag. scaling

**Input:** $A \in \mathbb{F}_q^{m\times n}$; $B \in \mathbb{F}_q^{n\times n}$, block diagonal with scalar or 2-dimensional blocks of the form $\left(\begin{smallmatrix} 0 & \beta \\ \beta & \gamma \end{smallmatrix}\right)$ with $\beta \neq 0$

**Output:** $A \cdot B \cdot A^{\mathsf{T}} \in \mathbb{F}_q^{m\times m}$

$\bar{A} \leftarrow A \in \mathbb{F}_q^{m\times n}$; $\bar{D} \leftarrow \mathrm{I}_n$
**for all** scalar blocks in $B$ at position $j$ **do** $\bar{D}_j \leftarrow B_{j,j}$
**if** $q$ is odd **then** ▷ *Use Eq. (29)*
  **for all** symmetric antidiagonal blocks in $B$ at $(j, j+1)$ **do**
    $\beta \leftarrow B_{j,j+1} (= B_{j+1,j})$
    $\bar{D}_j \leftarrow \frac{1}{2}\beta$ ; $\bar{D}_{j+1} \leftarrow -\frac{1}{2}\beta$
    $(\bar{A}_{*,i}\ \bar{A}_{*,j}) \leftarrow (\bar{A}_{*,i}\ \bar{A}_{*,j})\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$
**else**
  **for all** antitriangular blocks in $B$ at position $(j, j+1)$ **do**
    $\beta \leftarrow B_{j,j+1} (= B_{j+1,j})$ ; $\delta \leftarrow \mathtt{sqrt}(B_{j+1,j+1})$;
    $\bar{A}_{*,j} \leftarrow \beta\delta^{-1} \cdot \bar{A}_{*,j}$ ▷ *Scale column $j$ of $\bar{A}$*
    $\bar{A}_{*,j+1} \leftarrow \delta \cdot \bar{A}_{*,j+1}$ ▷ *Scale column $j+1$ of $\bar{A}$*
    $\bar{A}_{*,j+1} \leftarrow \bar{A}_{*,j+1} + \bar{A}_{*,j}$ ▷ *Use Eq. (30)*
    Swap columns $j$ and $j+1$ of $\bar{A}$
  **if** there are $n/2$ antidiagonal blocks in $B$ **then** ▷ *Use Eq. (31)*
    $\beta \leftarrow B_{1,2} (= B_{2,1})$
    $\bar{A}_{*,2} \leftarrow \beta \cdot \bar{A}_{*,2}$ ; $\bar{A} \leftarrow (\bar{A}\ \bar{A}_{*,1}+\bar{A}_{*,2}) \in \mathbb{F}_q^{m\times(n+1)}$
    $\ell \leftarrow 1$ ; $\delta \leftarrow 1$
  **else**
    $\delta \leftarrow \mathtt{sqrt}(\bar{D}_{\ell,\ell})$ where $\ell$ is s.t. $\bar{D}_{\ell,\ell}$ is a scalar block
  **for all** remaining antidiagonal blocks in $B$ at $(j, j+1)$ **do**
    $\beta \leftarrow B_{j,j+1} (= B_{j+1,j})$ ▷ *Use Eq. (32)*
    $\bar{A}_{*,\ell} \leftarrow \delta \cdot \bar{A}_{*,\ell}$ ; $\bar{A}_{*,j+1} \leftarrow \beta \cdot \bar{A}_{*,j+1}$
    $(\bar{A}_{*,\ell}\ \bar{A}_{*,j}\ \bar{A}_{*,j+1}) \leftarrow (\bar{A}_{*,\ell}\ \bar{A}_{*,j}\ \bar{A}_{*,j+1}) \cdot \left(\begin{smallmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right)$
    $\delta \leftarrow 1$
**return** $\mathtt{syrkd}(\bar{A}, \bar{D})$ ▷ $\bar{A} \cdot \bar{D} \cdot \bar{A}^{\mathsf{T}}$ *using Algorithm 5*

## REFERENCES


[1] M. Baboulin, L. Giraud, and S. Gratton. A parallel distributed solver for large dense symmetric systems: Applications to geodesy and electromagnetism problems. *Int. J. of HPC Applications*, 19(4):353–363, 2005. doi:10.1177/1094342005056134.

[2] G. Beniamini and O. Schwartz. Faster matrix multiplication via sparse decomposition. In *Proc. SPAA'19*, pages 11–22, 2019. doi:10.1145/3323165.3323188.

[3] M. Bodrato. A Strassen-like matrix multiplication suited for squaring and higher power computation. In *Proc. ISSAC'10*, pages 273–280. ACM, 2010. doi:10.1145/1837934.1837987.

[4] R. P. Brent. Algorithms for matrix multiplication. Technical Report STAN-CS-70-157, C.S. Dpt. Standford University, Mar. 1970.

[5] J. Brillhart. Note on representing a prime as a sum of two squares. *Math. of Computation*, 26(120):1011–1013, 1972. doi:10.1090/S0025-5718-1972-0314745-6.

[6] N. H. Bshouty. On the additive complexity of $2 \times 2$ matrix multiplication. *Inf. Processing Letters*, 56(6):329–335, Dec. 1995. doi:10.1016/0020-0190(95)00176-X.

[7] Ph. Chatelin. On transformations of algorithms to multiply $2 \times 2$ matrices. *Inf. processing letters*, 22(1):1–5, Jan. 1986. doi:10.1016/0020-0190(86)90033-5.

[8] H. F. de Groot. On varieties of optimal algorithms for the computation of bilinear mappings I. The isotropy group of a bilinear mapping. *Theoretical Computer Science*, 7(2):1–24, 1978. doi:10.1016/0304-3975(78)90038-5.

[9] H. F. de Groot. On varieties of optimal algorithms for the computation of bilinear mappings II. Optimal algorithms for $2 \times 2$-matrix multiplication. *Theoretical Computer Science*, 7(2):127–148, 1978. doi:10.1016/0304-3975(78)90045-2.

[10] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. on Math. Soft.*, 16(1):1–17, Mar. 1990. doi:10.1145/77626.79170.

[11] J.-G. Dumas, P. Giorgi, and C. Pernet. Dense linear algebra over prime fields. *ACM Trans. on Math. Soft.*, 35(3):1–42, Nov. 2008. doi:10.1145/1391989.1391992.

[12] J.-G. Dumas and C. Pernet. Symmetric indefinite elimination revealing the rank profile matrix. In *Proc. ISSAC'18*, pages 151–158. ACM, 2018. doi:10.1145/3208976.3209019.

[13] J.-C. Faugère. FGb: A Library for Computing Gröbner Bases. In *Proc ICMS'10*, LNCS, 6327, pages 84–87, 2010. doi:10.1007/978-3-642-15582-6_17.

[14] The FFLAS-FFPACK group. *FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package*, 2019. v2.4.1. URL: http://github.com/linbox-team/fflas-ffpack.

[15] N. J. Higham. Stability of a method for multiplying complex matrices with three real matrix multiplications. *SIMAX*, 13(3):681–687, 1992. doi:10.1137/0613043.

[16] E. Karstadt and O. Schwartz. Matrix multiplication, a little faster. In *Proc. SPAA'17*, pages 101–110. ACM, 2017. doi:10.1145/3087556.3087579.

[17] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proc ISSAC'14*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

[18] A. Lempel. Matrix factorization over $GF(2)$ and trace-orthogonal bases of $GF(2^n)$. *SIAM J. on Computing*, 4(2):175–186, 1975. doi:10.1137/0204014.

[19] G. Seroussi and A. Lempel. Factorization of symmetric matrices and trace-orthogonal bases in finite fields. *SIAM J. on Computing*, 9(4):758–767, 1980. doi:10.1137/0209059.

[20] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969. doi:10.1007/BF02165411.

[21] S. Wedeniwski. Primality tests on commutator curves. PhD U. Tübingen, 2001.

[22] S. Winograd. La complexité des calculs numériques. *La Recherche*, 8:956–963, 1977.

[23] Z. Xianyi, M. Kroeker, et al. *OpenBLAS, an Optimized BLAS library*, 2019. http://www.openblas.net/.

# A APPENDIX

## A.1 Proof of Proposition 3.1

Proposition 3.1 (Appendix A.1). *Algorithm 2 is correct for any skew-orthogonal matrix $Y$.*

Proof. If $Y$ is skew-orthogonal, then $Y \cdot Y^\intercal = -\mathrm{I}$. First,

$$U_3 = P_1 + P_2 = A_{11} \cdot A_{11}{}^\intercal + A_{12} \cdot A_{12}{}^\intercal = C_{11}. \tag{33}$$

Denote by $R_1$ the product:

$$\begin{aligned} R_1 &= A_{11} \cdot Y \cdot S_2{}^\intercal = A_{11} \cdot Y \cdot (A_{22}{}^\intercal - Y^\intercal \cdot A_{21}{}^\intercal) \\ &= A_{11} \cdot (Y \cdot A_{22}{}^\intercal + A_{21}{}^\intercal). \end{aligned} \tag{34}$$

Thus, as $S_3 = S_1 - A_{22} = (A_{21} - A_{11}) \cdot Y - A_{22} = -S_2 - A_{11} \cdot Y$:

$$\begin{aligned} U_1 = P_1 + P_5 &= A_{11} \cdot A_{11}{}^\intercal + S_3 \cdot S_3{}^\intercal \\ &= A_{11} \cdot A_{11}{}^\intercal + (S_2 + A_{11} \cdot Y) \cdot (S_2{}^\intercal + Y^\intercal \cdot A_{11}{}^\intercal) \\ &= S_2 \cdot S_2{}^\intercal + R_1{}^\intercal + R_1. \end{aligned} \tag{35}$$

And denote $R_2 = A_{21} \cdot Y \cdot A_{22}{}^\intercal$, so that:

$$\begin{aligned} S_2 \cdot S_2{}^\intercal &= (A_{22} - A_{21} \cdot Y) \cdot (A_{22}{}^\intercal - Y^\intercal \cdot A_{21}{}^\intercal) \\ &= A_{22} \cdot A_{22}{}^\intercal - A_{21} \cdot A_{21}{}^\intercal - R_2 - R_2{}^\intercal. \end{aligned} \tag{36}$$

Furthermore, from Equation (34):

$$\begin{aligned} R_1 + P_4 &= R_1 + S_1 \cdot S_2{}^\intercal \\ &= R_1 + (A_{21} - A_{11}) \cdot Y \cdot (A_{22}{}^\intercal - Y^\intercal \cdot A_{21}{}^\intercal) \\ &= A_{11} \cdot (Y \cdot A_{22}{}^\intercal + A_{21}{}^\intercal) + S_1 \cdot S_2{}^\intercal \\ &= A_{21} \cdot Y \cdot A_{22}{}^\intercal + A_{21} \cdot A_{21}{}^\intercal = R_2 + A_{21} \cdot A_{21}{}^\intercal. \end{aligned} \tag{37}$$

Therefore, from Equations (35), (36) and (37):

$$\begin{aligned} U_5 = U_1 + P_4 + P_4{}^\intercal &= S_2 \cdot S_2{}^\intercal + R_1 + R_1{}^\intercal + P_4 + P_4{}^\intercal \\ &= A_{22} \cdot A_{22}{}^\intercal + (-1+2) A_{21} \cdot A_{21}{}^\intercal = C_{22}. \end{aligned} \tag{38}$$

And the last coefficient $U_4$ of the result is obtained from Equations (37) and (38):

$$\begin{aligned} U_4 = U_2 + P_3 &= U_5 - P_4{}^\intercal + P_3 \\ &= U_2 + A_{22} \cdot (A_{12}{}^\intercal + Y^\intercal \cdot A_{21}{}^\intercal - Y^\intercal \cdot A_{11}{}^\intercal - A_{22}{}^\intercal) \\ &= A_{21} \cdot A_{21}{}^\intercal - P_4{}^\intercal + A_{22} \cdot (A_{12}{}^\intercal + Y^\intercal \cdot A_{21}{}^\intercal - Y^\intercal \cdot A_{11}{}^\intercal) \\ &= R_1{}^\intercal - R_2{}^\intercal + A_{22} \cdot (A_{12}{}^\intercal + Y^\intercal \cdot A_{21}{}^\intercal - Y^\intercal \cdot A_{11}{}^\intercal) \\ &= R_1{}^\intercal + A_{22} \cdot (A_{12}{}^\intercal - Y^\intercal \cdot A_{11}{}^\intercal) \\ &= A_{21} \cdot A_{11}{}^\intercal + A_{22} \cdot A_{12}{}^\intercal = C_{21}. \end{aligned} \tag{39}$$

Finally, $P_1 = A_{11} \cdot A_{11}{}^\intercal$, $P_2 = A_{12} \cdot A_{12}{}^\intercal$, and $P_5 = S_3 \cdot S_3{}^\intercal$ are symmetric by construction. So are therefore, $U_1 = P_1 + P_5$, $U_3 = P_1 + P_2$ and $U_5 = U_1 + (P_4 + P_4{}^\intercal)$. □

## A.2 Threshold in the theoretical number of operations for dimensions that are a power of two

Here, we look for a theoretical threshold where our fast symmetric algorithm performs less arithmetic operations than the classical one. Below that threshold any recursive call should call a classical algorithm for $A \cdot A^\intercal$. But, depending whether padding or static/dynamic peeling is used, this threshold varies. For powers of two, however, no padding nor peeling occurs and we thus have a look in this section of the thresholds in this case.

| n | | | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|
| SYRK | | | 70 | 540 | 4216 | 33264 | 264160 | 2105280 |
| | Rec. | SW | | | | | | |
| Syrk-i | | | **70** | **540** | **4216** | 33264 | 264160 | 2105280 |
| G0-i | 1 | 0 | 81 | 554 | **4020** | 30440 | 236496 | 1863584 |
| G1-i | | | 89 | 586 | **4148** | 30952 | 238544 | 1871776 |
| G2-i | | | 97 | 618 | 4276 | 31464 | 240592 | 1879968 |
| G3-i | | | 105 | 650 | 4404 | **31976** | 242640 | 1888160 |
| Syrk-i | | | 90 | 604 | 4344 | **32752** | 253920 | 1998784 |
| G0-i | 2 | 1 | | 651 | 4190 | **29340** | 217784 | 1674096 |
| G1-i | | | | 707 | 4414 | **30236** | 221368 | 1688432 |
| G2-i | | | | 763 | 4638 | **31132** | 224952 | 1702768 |
| G3-i | | | | 819 | 4862 | 32028 | 228536 | 1717104 |
| Syrk-i | | | | 824 | 5048 | 34160 | **248288** | 1886144 |
| G0-i | 3 | 2 | | | 4929 | 30746 | **210900** | 1546280 |
| G1-i | | | | | 5225 | 31930 | **215636** | 1565224 |
| G2-i | | | | | 5521 | 33114 | **220372** | 1584168 |
| G3-i | | | | | 5817 | 34298 | **225108** | 1603112 |
| Syrk-i | | | | | 6908 | 40112 | 260192 | **1838528** |
| G0-i | 4 | 3 | | | | 36099 | 221390 | **1500540** |
| G1-i | | | | | | 37499 | 226990 | **1522940** |
| G2-i | | | | | | 38899 | 232590 | **1545340** |
| G3-i | | | | | | 40299 | 238190 | **1567740** |

**Table 5: Number of arithmetic operations in the multiplication an $n \times n$ matrix by its transpose: blue when Syrk-i (using Strassen-Winograd with $i-1$ recursive levels) is better than other Syrk; orange/red/violet/green when ours (using Strassen-Winograd with $i-1$ recursive levels, and G0-i for $\mathbb{C}$ / G1-i if $-1$ is a square / G2-i or G3-i otherwise, depending whether $-2$ is a square or not) is better than others.**

First, from Section 3.1, over $\mathbb{C}$, we can choose $Y = i\,\mathrm{I}_n$. Then multiplications by $i$ are just exchanging the real and imaginary parts. In Equation (27) this is an extra cost of $y = 0$ arithmetic operations in usual machine representations of complex numbers. Overall, for $y = 0$ (complex case), $y = 1$ ($-1$ a square in the finite field) or $y = 3$ (any other finite field), the dominant term of the complexity is anyway unchanged, but there is a small effect on the threshold. In the following, we denote by $G0$, $G1$ and $G3$ these three variants.

More precisely, we denote by SYRK the classical multiplication of a matrix by its transpose. Then we denote by Syrk-i the algorithm making four recursive calls and two calls to a generic matrix multiplication via Strassen-Winograd's algorithm, the latter with $i-1$ recursive calls before calling the classical matrix multiplication. Finally G1-i (resp. G3-i) is our Algorithm 2 when $-1$ is a square (resp. not a square), with three recursive calls and two calls to Strassen-Winograd's algorithm, the latter with $i-1$ recursive calls.

Now, we can see in Table 5 in which range the thresholds live. For instance, over a field where $-1$ is a square, Algorithm 2 is better for $n \geq 16$ with 1 recursive level (and thus 0 recursive levels for Strassen-Winograd), for $n \geq 32$ with 2 recursive levels, etc. Over a field where $-1$ is not a square, Algorithm 2 is better for $n \geq 32$ with 1 recursive level, for $n \geq 64$ with 3 recursive levels, etc.