



HAL
open science

On fast multiplication of a matrix by its transpose

Jean-Guillaume Dumas, Clément Pernet, Alexandre Sedoglavic

► **To cite this version:**

Jean-Guillaume Dumas, Clément Pernet, Alexandre Sedoglavic. On fast multiplication of a matrix by its transpose. 2020. hal-02432390v3

HAL Id: hal-02432390

<https://hal.science/hal-02432390v3>

Preprint submitted on 4 May 2020 (v3), last revised 9 Jun 2020 (v4)

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

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

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 2×2 -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 2×2 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 $L \cdot D \cdot L^T$ factorization.

1 INTRODUCTION

Strassen's algorithm [21], with 7 recursive multiplications and 18 additions, was the first sub-cubic time algorithm for matrix product, with a complexity bound 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 $\text{MM}_\omega(n)$ (see [18] for the best theoretical value of ω known to date).

We propose here a new algorithm for the computation of the product of a 2×2 -block-matrix by its transpose, $A \cdot A^T$, using only 5 block multiplications over some base field, instead of 6 for the natural divide & conquer algorithm. For such a product, the best previously known complexity bound was dominated by $\frac{2}{2^\omega-4} \text{MM}_\omega(n)$ over any base field (see e.g. [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 $\frac{2}{2^\omega-3} \text{MM}_\omega(n)$ 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 2×2 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 ω . By exploiting the symmetry of the problem, it requires about half of the arithmetic cost of general matrix multiplication when ω 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 2×2 matrices, the matrix product $C = A \cdot B$ could be computed using Strassen algorithm by performing the following

computations (see [21]):

$$\begin{aligned} \rho_1 &\leftarrow a_{11}(b_{12} - b_{22}), \\ \rho_2 &\leftarrow (a_{11} + a_{12})b_{22}, \quad \rho_4 \leftarrow (a_{12} - a_{22})(b_{21} + b_{22}), \\ \rho_3 &\leftarrow (a_{21} + a_{22})b_{11}, \quad \rho_5 \leftarrow (a_{11} + a_{22})(b_{11} + b_{22}), \\ \rho_6 &\leftarrow a_{22}(b_{21} - b_{11}), \quad \rho_7 \leftarrow (a_{21} - a_{11})(b_{11} + b_{12}), \end{aligned} \quad (1)$$

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} \rho_5 + \rho_4 - \rho_2 + \rho_6 & \rho_6 + \rho_3 \\ \rho_2 + \rho_1 & \rho_5 + \rho_7 + \rho_1 - \rho_3 \end{pmatrix}.$$

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

$$\begin{aligned} \mathbb{K}^{m \times n} \times \mathbb{K}^{n \times p} &\rightarrow \mathbb{K}^{m \times p}, \\ (X, Y) &\rightarrow X \cdot Y, \end{aligned} \quad (2)$$

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

$$\begin{aligned} \mathbb{K}^{m \times n} \times \mathbb{K}^{n \times p} \times (\mathbb{K}^{m \times p})^* &\rightarrow \mathbb{K}, \\ (X, Y, Z^T) &\rightarrow \langle Z, X \cdot Y \rangle. \end{aligned} \quad (3)$$

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:

$$\begin{aligned} \sum_{i=1}^7 S_{i1} \otimes S_{i2} \otimes S_{i3} &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} + \\ &\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix} + \\ &\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \\ &\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned} \quad (4)$$

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

$$\begin{aligned} ((\mathbb{K}^{m \times n})^* \otimes (\mathbb{K}^{n \times p})^* \otimes \mathbb{K}^{m \times p}) \otimes (\mathbb{K}^{m \times n} \otimes \mathbb{K}^{n \times p}) &\rightarrow \mathbb{K}^{m \times p}, \\ \mathcal{S} \otimes (A \otimes B) &\rightarrow \sum_{i=1}^7 \langle S_{i1}, A \rangle \langle S_{i2}, B \rangle S_{i3}, \end{aligned} \quad (5)$$

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

The tensor formulation of matrix multiplication algorithm gives explicitly its symmetries (a.k.a. *isotropies*). As this formulation is associated to the trilinear form $\text{Trace}(A \cdot B \cdot C)$, given three invertible matrices U, V, W of suitable sizes and the classical properties

of the trace, one can remark that $\text{Trace}(A \cdot B \cdot C)$ is equal to:

$$\begin{aligned} \text{Trace}((A \cdot B \cdot C)^\top) &= \text{Trace}(C \cdot A \cdot B) = \text{Trace}(B \cdot C \cdot A), \\ &= \text{Trace}(U^{-1} \cdot A \cdot V \cdot V^{-1} \cdot B \cdot W \cdot W^{-1} \cdot C \cdot U). \end{aligned} \quad (6)$$

These relations illustrate the following theorem:

THEOREM 2.1 ([8, § 2.8]). *The isotropy group of the $n \times n$ matrix multiplication tensor is $\text{PSL}^\pm(\mathbb{K}^n)^{\times 3} \rtimes \mathfrak{S}_3$, where PSL stands for the group of matrices of determinant ± 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 $g = (U \times V \times W)$ in $\text{PSL}^\pm(\mathbb{K}^n)^{\times 3}$, its action $g \diamond \mathcal{S}$ on a tensor \mathcal{S} is given by $\sum_{i=1}^7 g \diamond (S_{i1} \otimes S_{i2} \otimes S_{i3})$ where the term $g \diamond (S_{i1} \otimes S_{i2} \otimes S_{i3})$ is equal to:*

$$(U^{-\top} \cdot S_{i1} \cdot V^\top) \otimes (V^{-\top} \cdot S_{i2} \cdot W^\top) \otimes (W^{-\top} \cdot S_{i3} \cdot U^\top). \quad (7)$$

REMARK 2.1. *In $\text{PSL}^\pm(\mathbb{K}^n)^{\times 3}$, 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 $\{g_2, g_1^\top \diamond A \otimes B \otimes C\}$.*

The following theorem shows that all 2×2 -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 $\text{PSL}^\pm(\mathbb{K}^n)^{\times 3}$ acts transitively on the variety of optimal algorithms for the computation of 2×2 -matrix multiplication.*

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

2.1 Design of a specific 2×2 -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:*

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

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

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

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 complexity 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 [23].

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 complexity bound of this algorithm. Hence, in order to reduce the number of addition 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's denote by $e_{(i,j)}$ the matrix whose (l, k) entry is 1 when (i, j) is (l, k) and 0 otherwise. A 2×2 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 2×2 -matrix product is at least 15 ([6, Theorem 1]).*

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

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

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

$$w = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & -1 \\ 0 & -1 \end{pmatrix} \times \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}. \quad (11)$$

The variant of the Winograd tensor [23] presented with a renumbering as Algorithm 1 is obtained by the action of 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 : $C = W(A, B)$

Require: $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ and $B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$;

Ensure: $C = A \cdot B$

1: 8 additions:

$$\begin{aligned} s_1 &\leftarrow a_{11} - a_{21}, & s_2 &\leftarrow a_{21} + a_{22}, & s_3 &\leftarrow s_2 - a_{11}, & s_4 &\leftarrow a_{12} - s_3, \\ t_1 &\leftarrow b_{22} - b_{12}, & t_2 &\leftarrow b_{12} - b_{11}, & t_3 &\leftarrow b_{11} + t_1, & t_4 &\leftarrow b_{21} - t_3. \end{aligned}$$

2: 7 recursive multiplications:

$$\begin{aligned} p_1 &\leftarrow a_{11} \cdot b_{11}, & p_2 &\leftarrow a_{12} \cdot b_{21}, & p_3 &\leftarrow a_{22} \cdot t_4, & p_4 &\leftarrow s_1 \cdot t_1, \\ p_5 &\leftarrow s_3 \cdot t_3, & p_6 &\leftarrow s_4 \cdot b_{22}, & p_7 &\leftarrow s_2 \cdot t_2. \end{aligned}$$

3: 7 final additions:

$$\begin{aligned} c_1 &\leftarrow p_1 + p_5, & c_2 &\leftarrow c_1 + p_4, & c_3 &\leftarrow p_1 + p_2, & c_4 &\leftarrow c_2 + p_3, \\ c_5 &\leftarrow c_2 + p_7, & c_6 &\leftarrow c_1 + p_7, & c_7 &\leftarrow c_6 + p_6. \end{aligned}$$

4: **return** $C = \begin{pmatrix} c_3 & c_7 \\ c_4 & c_5 \end{pmatrix}$.

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 $\{1, \dots, 7\}$ 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 12 unknowns and $3 + 4|J|$ equations whose solutions define matrix squaring algorithms.

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

$$\mathbf{g} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \quad (12)$$

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 2×2 matrix product $A \cdot A^\top$.

2.2 2×2 -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 $A \otimes A^\top$ computing $A \cdot A^\top$.

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

$$\begin{aligned} \langle T_{i2}, A^\top \rangle &= \text{Trace}(T_{i2}^\top \cdot A^\top) = \text{Trace}((A \cdot T_{i2})^\top), \\ &= \text{Trace}(A \cdot T_{i2}) = \text{Trace}(T_{i2} \cdot A) = \langle T_{i2}^\top, A \rangle. \end{aligned} \quad (13)$$

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

$$\sum_{i=1}^7 \langle T_{i1}, A \rangle \langle T_{i2}, A^\top \rangle T_{i3} = \sum_{i=1}^7 \langle T_{i1}, A \rangle \langle T_{i2}^\top, A \rangle T_{i3}. \quad (14)$$

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 $\{1, \dots, 7\}$ and H of $\{(i, j) \in \{2, \dots, 7\}^2 \mid i \neq j, i \notin J, j \notin J\}$ in order to express the following constraints:

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

The constraints of type J allow one to save preliminary additions when applying the method to matrices $B = A^\top$: since then operations on A and A^\top 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 one 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 $z = v_{11} - v_{21}$:

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

REMARK 2.3. As $((z + v_{21})^2 + v_{21}^2)^2 + 1 = 0$ 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 2×2 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, $A \cdot A^\top$ 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^\top . 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 $\{1, \dots, 7\}$:

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

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:

$$\mathbf{a} = \begin{pmatrix} z^2 & 0 \\ 0 & z^2 \end{pmatrix} \times \begin{pmatrix} z & -z \\ 0 & z^3 \end{pmatrix} \times \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix}, \quad z^4 = -1. \quad (18)$$

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

$$\mathbf{d} = D_1 \otimes D_2 \otimes D_3 = \begin{pmatrix} z^2 & 0 \\ 0 & z^2 \end{pmatrix} \times \begin{pmatrix} z & 0 \\ 0 & -z^3 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad z^4 = -1. \quad (19)$$

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

$$\{(\mathbf{d} \circ \mathbf{w}) \circ \mathcal{S}, (A \otimes A^\top \otimes C)\} = \{\mathbf{w} \circ \mathcal{S}, \mathbf{d}^\top \circ (A \otimes A^\top \otimes C)\} \quad (20)$$

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

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

$$\left(D_1^{-1} \cdot A \cdot D_2\right) \otimes \left(D_2^{-1} \cdot A^\top \cdot D_3\right) \otimes \left(D_3^{-1} \cdot C \cdot D_1\right). \quad (21)$$

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:

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

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

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:

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

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

$$C = W\left(A \cdot \frac{D_2}{z}, \left(\frac{D_2}{z}\right)^{-1} \cdot A^\top\right) = W\left(A \cdot Q, \left(A \cdot (Q^{-1})^\top\right)^\top\right). \quad (24)$$

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

3 FAST 2×2 -BLOCK RECURSIVE SYRK

The algorithm presented in the previous section is noncommutative 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): 2 can be avoided completely because they are just transposition of other products, 3 are recursive calls to SYRK and 2 are generic matrix products. This results in Algorithm 2.

PROPOSITION 3.1 (APPENDIX A.1). *Algorithm 2 is correct for any skew-orthogonal matrix Y .*

3.1 Minimality of the number of additions

From Lemma 2.1, we know that 11 additions are minimal to compute $A \cdot A^\top$ from 7 multiplications in generic 2×2 matrices. Here we are considering matrices over a field, therefore $C = A \cdot A^\top$ is a symmetric matrix and the lower left block of the result is exactly the transpose of the upper right one. Therefore, we can save the additions computing one of those blocks, as stated by Proposition 3.2.

PROPOSITION 3.2 (APPENDIX A.2). *9 block additions are necessary and sufficient to multiply a 2×2 block matrix over a field by its transpose with a bilinear algorithm that uses 5 multiplications.*

Algorithm 2 : Matrix-parameterized Fast Symmetric product

Require: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$;

Require: A skew-orthogonal matrix Y .

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

1: 4 additions and 2 multiplications by Y :

$$\begin{aligned} S_1 &\leftarrow (A_{21} - A_{11}) \cdot Y, & S_2 &\leftarrow A_{22} - A_{21} \cdot Y, \\ S_3 &\leftarrow S_1 - A_{22}, & S_4 &\leftarrow S_3 + A_{12}. \end{aligned} \quad (25)$$

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

$$\begin{aligned} P_1 &\leftarrow A_{11} \cdot A_{11}^\top, & P_2 &\leftarrow A_{12} \cdot A_{12}^\top, & P_3 &\leftarrow A_{22} \cdot S_4^\top, \\ P_4 &\leftarrow S_1 \cdot S_2^\top, & P_5 &\leftarrow S_3 \cdot S_3^\top. \end{aligned} \quad (26)$$

3: 2 symmetric additions (half additions);

$$\begin{aligned} \text{Low}(U_1) &\leftarrow \text{Low}(P_1) + \text{Low}(P_5), & \{U_1, P_1, P_5 \text{ are symm.}\} \\ \text{Low}(U_3) &\leftarrow \text{Low}(P_1) + \text{Low}(P_2), & \{U_3, P_1, P_2 \text{ are symm.}\} \end{aligned} \quad (27)$$

4: 2 complete additions (P_4 and P_3 are not symmetric):

$$\text{Up}(U_1) \leftarrow \text{Low}(U_1)^\top, \quad U_2 \leftarrow U_1 + P_4, \quad U_4 \leftarrow U_2 + P_3. \quad (28)$$

5: 1 half addition ($U_5 = U_1 + P_4 + P_4^\top$ is symmetric):

$$\text{Low}(U_5) \leftarrow \text{Low}(U_2) + \text{Low}(P_4^\top). \quad (29)$$

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

REMARK 3.1. *The symmetry of blocks C_{11} and C_{22} gives a candidate minimal number of extra additions for a 5 multiplications algorithm. Indeed, suppose that C_{11} (resp. and C_{22}) require 1 (resp. 2) block addition like in Algorithm 2 (from the proof of Proposition 3.2, this is the case in all variants with minimal number of block additions).*

The symmetry of each of these blocks gives that only, say, their lower part needs to be computed. This is $1.5(n+1)n$ instead of $3n^2$ additions, for a total of $4n^2 + 2n^2 + 1.5(n+1)n = 7.5n^2 + 1.5n$ additions, as in Algorithm 2.

To further reduce the number of additions, a promising approach is that undertaken in [2, 17]. This is however not clear to us how to adapt our strategy to their recursive transformation of basis.

3.2 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 .

3.2.1 Over the complex field. Therefore Algorithm 2 is directly usable over $\mathbb{C}^{n \times n}$ with $Y = iI_n \in \mathbb{C}^{n \times n}$. Further, usually, complex numbers are emulated by a pair of floats so then the multiplications by $Y = iI_n$ 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].

3.2.2 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 $Y = iI_n \in \mathbb{F}^{n \times n}$), but

within the field. Proposition 3.3 thereafter characterizes these fields.

PROPOSITION 3.3 (APPENDIX A.3). *Fields with characteristic two, or with an odd characteristic $p \equiv 1 \pmod{4}$, or finite fields that are an even extension, contain a square root of -1 .*

3.2.3 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 3.4, 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 $A \cdot A^\top = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} A^\top \\ 0 \end{pmatrix}$.

PROPOSITION 3.4 (APPENDIX A.4). *Let \mathbb{F} be a field of characteristic p , there exists (a, b) in \mathbb{F}^2 such that the matrix:*

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \otimes I_n = \begin{pmatrix} aI_n & bI_n \\ -bI_n & aI_n \end{pmatrix} \text{ in } \mathbb{F}^{2n \times 2n} \quad (30)$$

is skew-orthogonal.

Proposition 3.4 shows that skew-orthogonal matrices do exist for any field with prime characteristic. For Algorithm 2, we need to build them mostly for $p \equiv 3 \pmod{4}$ (otherwise use Proposition 3.3).

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 $r \equiv 1 \pmod{4}$ and $r \equiv -1 \pmod{p}$ 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 $a^2 + b^2 \equiv -1 \pmod{p}$.

By the prime number theorem the first step is polynomial in $\log(p)$, as is the second step (square root modulo a prime, denoted ModSquareRoot, 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 : Sum of squares modulo prime

Require: $p \in \mathbb{P} \setminus \{2\}, k \in \mathbb{Z}$.

Ensure: $(a, b) \in \mathbb{Z}^2$, s.t. $a^2 + b^2 \equiv k \pmod{p}$.

```

1: if  $\left(\frac{k}{p}\right) == 1$  then                                {k is a square mod p}
2:   return (ModSquareRoot(k, p), 0).
3: else
4:    $s = 2$ ; while  $\left(\frac{s}{p}\right) == 1$  do {Lowest quadratic non-residue}
5:      $s = s + 1$ ;
6:   end while
7:   end if
8:    $c = \text{ModSquareRoot}(s - 1, p)$ ;                    {s - 1 must be a square}
9:    $r \equiv ks^{-1} \pmod{p}$ ;
10:   $a = \text{ModSquareRoot}(r, p)$ ; {Now  $k \equiv a^2s \equiv a^2(1 + c^2) \pmod{p}$ }
11: return  $(a, ac \pmod{p})$ .

```

PROPOSITION 3.5 (APPENDIX A.5). *Algorithm 3 is correct and, under the ERH, runs in expected time $\tilde{O}(\log^3(p))$.*

REMARK 3.2. *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 s -o- s modulo the same prime, one can compute the LQNR only once to get all the sum of squares with an expected cost bounded by $\tilde{O}(\log^3(p) + t \log^2(p))$.*

REMARK 3.3. *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 $\text{FFSoS}_{\mathbb{F}_q}(k)$ this algorithm decomposing k as a sum of squares within any finite field \mathbb{F}_q . This is not true over infinite fields, but there Algorithm 3 still works anyway for the special case $k = -1$: just run it in the prime subfield.*

3.3 Conjugate transpose

Note that Algorithm 2 remains valid if transposition is replaced by conjugate transposition, provided that there exists a matrix Y such that $Y \cdot Y^\top = -I$. 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 $Y = \sqrt{-1} \cdot I_n$ still works; otherwise there exists a square root i of -1 in \mathbb{F}_{q^2} , from Proposition 3.3. In the latter case, thus build (a, b) , both in \mathbb{F}_q , such that $a^2 + b^2 = -1$, and $Y = (a + ib) \cdot I_n$ in $\mathbb{F}_{q^2}^{n \times n}$ is appropriate: indeed, since $q \equiv 3 \pmod{4}$, we have that $\overline{a + ib} = (a + ib)^q = a - ib$.

4 ANALYSIS AND IMPLEMENTATION

4.1 Complexity bounds

THEOREM 4.1 (APPENDIX A.6). *Given ω such that $3 > \omega > 2$, if a generic matrix product algorithm requires less than $C_\omega n^\omega + o(n^\omega)$ operations then, over \mathbb{C} or over a prime characteristic field, Algorithm 2 can require less than $\frac{2}{2^\omega - 3} C_\omega n^\omega + o(n^\omega)$ arithmetic operations.*

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.	$O(n^3)$	$O(n^{\log_2(7)})$	$O(n^\omega)$
$A \cdot A^\top \in \mathbb{F}^{n \times n}$	[11]	n^3	$\frac{2}{3} \text{MM}_{\log_2(7)}(n)$	$\frac{2}{2^\omega - 4} \text{MM}_\omega(n)$
	Alg. 2	$0.8n^3$	$\frac{1}{2} \text{MN}_{\log_2(7)}(n)$	$\frac{2}{2^\omega - 3} \text{MM}_\omega(n)$

Table 1: Arithmetic complexity bounds leading terms.

Differently, as complex numbers are usually emulated by a pair of floats, one can use the 3M method (Karatsuba) for generic matrix multiplication over the complex field to get one complex multiplication in only 3 floating point multiplications [15]. If we denote by RR_ω the complexity bound on floating point matrix multiplication then the generic 3M method requires $3RR_\omega + o(n^\omega)$ floating point operations.

Now a 2M symmetric method would use 2 floating point multiplications: compute $G = (A + B) \cdot (A^\top - B^\top)$ and $H = A \cdot B^\top$, then the

¹In practice, the running time seems very close to that of Algorithm 3 anyway, see, e.g. the implementation in Givaro rev. 7bdef6, <https://github.com/linbox-team/givaro>.

relation $(A + iB) \cdot (A^\top + iB^\top) = (G - H^\top + H) + i(H + H^\top)$ holds. This method uses therefore $2RR_\omega + o(n^\omega)$ operations.

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 generic complex floating point products, this algorithm uses $\frac{6}{2^{\omega-4}}RR_\omega + o(n^\omega)$ operations.

Finally, our Algorithm 2 would use only $\frac{2}{2^{\omega-3}}$ complex floating point multiplications for a leading term bounded by $\frac{6}{2^{\omega-3}}RR_\omega$, better than 2 for $\omega > \log_2(6) \approx 2.585$.

This is summarized in Table 2, replacing ω by 3 or $\log_2(7)$.

Problem	Alg.	$O(n^3)$	$O(n^{\log_2(7)})$	$O(n^\omega)$
$A \cdot B \in \mathbb{C}^{n \times n}$	naive	$8n^3$	$4RR_{\log_2(7)}(n)$	$4RR_\omega(n)$
	3M	$6n^3$	$3RR_{\log_2(7)}(n)$	$3RR_\omega(n)$
$A \cdot A^\top \in \mathbb{C}^{n \times n}$	2M	$4n^3$	$2RR_{\log_2(7)}(n)$	$2RR_\omega(n)$
	[11]	$3n^3$	$2RR_{\log_2(7)}(n)$	$\frac{6}{2^{\omega-4}}RR_\omega(n)$
	Alg. 2	$2.4n^3$	$\frac{3}{2}RR_{\log_2(7)}(n)$	$\frac{6}{2^{\omega-3}}RR_\omega(n)$

Table 2: Complexity bounds leading term over the complex field, emulated with separate real and imaginary parts.

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 $C \leftarrow A \cdot A^\top$ 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}		$\text{Up}(U_1) = \text{Low}(U_1)^\top$	C_{12}
3	$P_4^\top = S_2 \cdot S_1^\top$	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	$P_5 = S_3 \cdot S_3^\top$	C_{12}	12	$U_5 = U_2 + P_4^\top$	C_{22}
6	$S_4 = S_3 + A_{12}$	C_{11}	13	$P_2 = A_{12} \cdot A_{12}^\top$	C_{12}
7	$P_3 = A_{22} \cdot S_4^\top$	C_{21}	14	$U_3 = P_1 + P_2$	C_{11}
8	$P_1 = A_{11} \cdot A_{11}^\top$	C_{11}			

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

For the more general operation $C \leftarrow \alpha A \cdot A^\top + \beta C$, 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 [14, commit 0a91d61e] for dense linear algebra over a finite field. The library is linked with OpenBLAS [24, v0.3.6] and compiled with `gcc-9.2` on an Intel skylake i7-6700 running a Debian GNU/Linux system (v5.2.17).

Figure 3 compares the computation speed in effective Gfops (defined as $n^3 / (10^9 \times \text{time})$) of this implementation over $\mathbb{Z}/131071\mathbb{Z}$ with that of the double precision BLAS routines `dsyrk`, the classic cubic-time routine over a finite field (calling `dsyrk` and performing modular reductions on the result), and the classic divide and

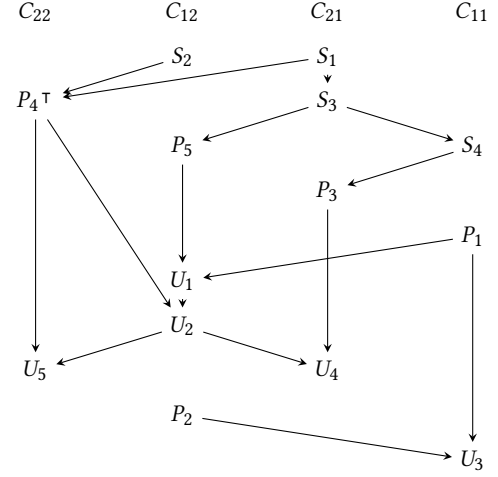


Figure 1: DAG of the tasks and their memory location for the computation of $C \leftarrow A \cdot A^\top$ presented in Table 3.

operation	loc.	operation	loc.
$S_1 = (A_{21} - A_{11}) \cdot Y$	tmp	$P_1 = \alpha A_{11} \cdot A_{11}^\top$	tmp
$S_2 = A_{22} - A_{21} \cdot Y$	C_{12}	$U_1 = P_1 + P_5$	C_{12}
$\text{Up}(C_{11}) = \text{Low}(C_{22})^\top$	C_{11}	$\text{Up}(U_1) = \text{Low}(U_1)^\top$	C_{12}
$P_4^\top = \alpha S_2 \cdot S_1^\top$	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}
$P_5 = \alpha S_3 \cdot S_3^\top$	C_{12}	$U_5 = U_2 + P_4^\top + \beta \text{Up}(C_{11})^\top$	C_{22}
$S_4 = S_3 + A_{12}$	C_{11}	$P_2 = \alpha A_{12} \cdot A_{12}^\top + \beta C_{11}$	C_{11}
$P_3 = \alpha A_{22} \cdot S_4^\top + \beta C_{21}$	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 $C \leftarrow \alpha A \cdot A^\top + \beta C$ when $k \leq n$. The block C_{12} of the output matrix as well as an $n/2 \times n/2$ block tmp are used as temporary storages.

conquer algorithm [11, § 6.3.1]. The slight overhead of performing the modular reductions is quickly compensated by the speed-up of the sub-cubic algorithm (the threshold for a first recursive call is near $n = 2000$). The classic divide and conquer approach also speeds up the classic algorithm, but starting from a larger threshold, and hence at a slower pace. Lastly, we also show that the speed is merely identical over the field $\mathbb{Z}/131041\mathbb{Z}$, having square roots of -1 , 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: $C \leftarrow C - A \cdot D \cdot A^\top$.

Over finite fields, though, extending the factorization of quadratic non-residues of Algorithm 3, one can actually factor the

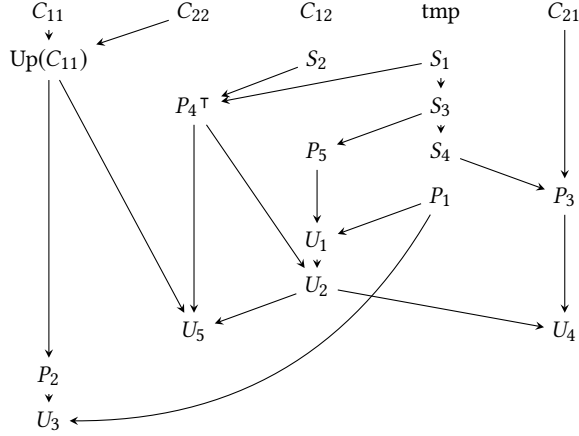


Figure 2: DAG of the tasks and their memory location for the computation of $C \leftarrow \alpha A \cdot A^T + \beta C$ presented in Table 4.

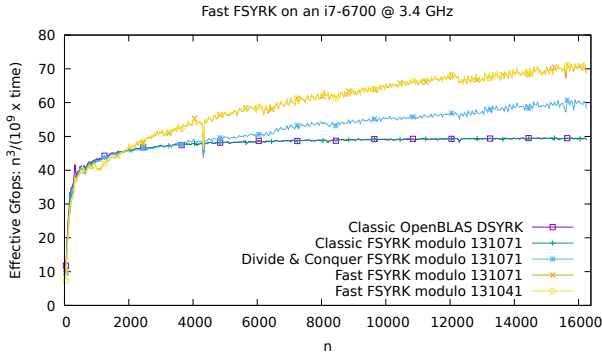


Figure 3: Speed of an implementation of Algorithm 2

diagonal matrix D into $D = \Delta \cdot \Delta^T$, *without needing any field extension*, and then compute instead $(A \cdot \Delta) \cdot (A \cdot \Delta)^T$. This is what we propose in this section. Indeed, this is better to deal with potential non-squares and 2×2 blocks before launching a recursive algorithm. For instance, a 2×2 diagonal blocks might have to be cut by a recursive cut of dimensions. 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 virtual zero column to deal with these cases: this is potentially $O(\log_2(n))$ extra columns.

Differently, with Algorithm 6, thereafter, one has to deal with non-squares and 2×2 blocks only beforehand, with a maximum of only 2 additional zero columns overall.

For this algorithm, we then need to ensure the following:

- only perform recursive calls on blocks with even dimensions;
- avoid resorting to field extensions, use instead Section 5.1 to factor a diagonal matrix in the base field;
- deal with antidiagonal or antitriangular 2×2 blocks, depending on the characteristic, as shown in Sections 5.2.1 and 5.2.2.

5.1 Factoring non-squares within a finite field

Algorithm 4 : Symmetric factorization of a pair of non-residues

Require: $(\alpha, \beta) \in \mathbb{F}_q^2$, both being quadratic non-residues.

Ensure: $Y \in \mathbb{F}_q^{2 \times 2}$, s.t. $Y \cdot Y^T = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}$.

- 1: $(a, b) \leftarrow \text{FFSoS}_{\mathbb{F}_q}(\alpha)$; $\{\alpha = a^2 + b^2\}$
- 2: $d \leftarrow a \text{FFSqrt}_{\mathbb{F}_q}(\beta \alpha^{-1})$; $\{d^2 = a^2 \beta \alpha^{-1}\}$
- 3: $c \leftarrow -b d a^{-1}$; $\{ac + bd = 0\}$
- 4: **return** $Y = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

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 $A \cdot D \cdot A^T$ with D a diagonal matrix: factor D with a tridiagonal matrix $D = \Delta \cdot \Delta^T$, 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 : $A \cdot D \cdot A^T$: SYRK with a diagonal over a finite field

Require: $A \in \mathbb{F}_q^{m \times n}$ and $(d_1, \dots, d_n) \in \mathbb{F}_q^n$.

Ensure: $A \cdot \text{DiagonalMatrix}(d_1, \dots, d_n) \cdot A^T$ in $\mathbb{F}_q^{n \times n}$.

- 1: **if** number of quadratic non-residues in D is odd **then**
- 2: Let d_ℓ be one of the quadratic non-residues;
- 3: Form $\bar{D} = \text{DiagonalMatrix}(d_1, \dots, d_n, d_\ell) \in \mathbb{F}_q^{(n+1) \times (n+1)}$
- 4: $\bar{A} = (A \ 0) \in \mathbb{F}_q^{m \times (n+1)}$ {Augment A with a zero column}
- 5: **else**
- 6: $\bar{D} = D = \text{DiagonalMatrix}(d_1, \dots, d_n) \in \mathbb{F}_q^{n \times n}$;
- 7: $\bar{A} = A \in \mathbb{F}_q^{m \times n}$
- 8: **end if**
- 9: **for all** quadratic residues d_j in \bar{D} **do**
- 10: $\bar{A}_{*,j} \leftarrow \text{FFSqrt}_{\mathbb{F}_q}(d_j) \cdot \bar{A}_{*,j}$ {Scale column j of \bar{A} by a square root of d_j }
- 11: **end for**
- 12: **for all** distinct pairs of quadratic non-residues (d_i, d_j) in \bar{D} **do**
- 13: Let $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ be the symmetric factorization of $\begin{pmatrix} d_i & 0 \\ 0 & d_j \end{pmatrix}$ {Alg. 4}
- 14: $(\bar{A}_{*,i} \ \bar{A}_{*,j}) \leftarrow (\bar{A}_{*,i} \ \bar{A}_{*,j}) \begin{pmatrix} a & b \\ c & d \end{pmatrix}$;
- 15: **end for**
- 16: **return** $\bar{A} \cdot \bar{A}^T$. {Alg. 2}

5.2 Antidiagonal and antitriangular blocks

In a generic $L \cdot D \cdot L^T$ factorization, antitriangular or antidiagonal blocks can appear in D [12]. In order to use a fast symmetric multiplication as the main subroutine to this factorization, it is more efficient to preprocess these blocks in order to deal only with a diagonal matrix. The next sections are devoted to handling this point.

5.2.1 Antidiagonal blocks in odd characteristic. In odd characteristic, the 2-dimensional blocks in a symmetric factorization are only symmetric antidiagonal blocks i.e. scalings of the antidiagonal

identity: $\begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix}$. For those blocks, it is possible to factor them symmetrically using Equation (31), and therefore reduce to the diagonal case (note the requirement that 2 is invertible).

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} \frac{1}{2}\beta & 0 \\ 0 & -\frac{1}{2}\beta \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^\top = \begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix}. \quad (31)$$

5.2.2 Antitriangular blocks in even characteristic. In even characteristic, some 2×2 symmetric blocks might not be only antidiagonal anymore, but also antitriangular of the form: $\begin{pmatrix} 0 & \beta \\ \beta & \gamma \end{pmatrix}$, with γ nonzero.

In even characteristic every element is a square, therefore those antitriangular blocks can be factored as shown in Equation (32):

$$\begin{pmatrix} 0 & \beta \\ \beta & \gamma \end{pmatrix} = \begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & \frac{\beta}{\sqrt{\gamma}} \\ \frac{\beta}{\sqrt{\gamma}} & 0 \end{pmatrix} \begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & \frac{\beta}{\sqrt{\gamma}} \\ \frac{\beta}{\sqrt{\gamma}} & 0 \end{pmatrix}^\top = \begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & 0 \\ 0 & \sqrt{\gamma} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & 0 \\ 0 & \sqrt{\gamma} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^\top. \quad (32)$$

Therefore the antitriangular blocks also resume to the diagonal case after adding and swapping two rows, i.e. preprocessing by $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$.

5.2.3 Antidiagonal blocks in even characteristic. In the antidiagonal case this is more complicated: the symmetric factorization might require an extra row or column [19]. This is shown in Equation (33):

$$\begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}^\top = \begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix} \pmod{2}. \quad (33)$$

One could thus add one row and one column to A at each antidiagonal block and then replace it with the 2×3 symmetric factorization of Equation (33). It is however more efficient to combine a diagonal element, say x , and an antidiagonal block with Equation (34) instead. For this, consider the diagonal matrix $D_{\sqrt{x},1,\beta}$ with coefficients \sqrt{x} , 1 , β , the transformation $M = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ and compute $D_{\sqrt{x},1,\beta} \cdot M \cdot M^\top \cdot D_{\sqrt{x},1,\beta}^\top$ that is:

$$\begin{pmatrix} \sqrt{x} & \sqrt{x} & \sqrt{x} \\ 1 & 0 & 1 \\ 0 & \beta & \beta \end{pmatrix} \begin{pmatrix} \sqrt{x} & \sqrt{x} & \sqrt{x} \\ 1 & 0 & 1 \\ 0 & \beta & \beta \end{pmatrix}^\top = \begin{pmatrix} x & 0 & 0 \\ 0 & 0 & \beta \\ 0 & \beta & 0 \end{pmatrix} \pmod{2}. \quad (34)$$

With Equation (34), we thus can just combine any antidiagonal block with any 1×1 block in order to factor them.

There remains the case when there are no 1×1 blocks. Then, one needs to use Equation (33) once, on the first antidiagonal block, and add a single row and 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 afterwards as x in any further combination with the next antidiagonal blocks.

Algorithm 6 summarizes the use of Equations (31) to (34).

REFERENCES

- [1] M. Baboulin, L. Giraud, and S. Gratton. A parallel distributed solver for large dense symmetric systems: Applications to geodesy and electromagnetism problems. *The International Journal of High Performance Computing Applications*, 19(4):353–363, 2005. doi:10.1177/1094342005056134.
- [2] G. Beniamini and O. Schwartz. Faster matrix multiplication via sparse decomposition. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'19*, pages 11–22. ACM, 2019. doi:10.1145/3323165.3323188.
- [3] M. Bodrato. A Strassen-like matrix multiplication suited for squaring and higher power computation. In W. Koepf, editor, *ISSAC'2010, Munich, Germany*, pages 273–280, July 2010. doi:10.1145/1837934.1837987.
- [4] R. P. Brent. Algorithms for matrix multiplication. Technical Report STAN-CS-70-157, Computer Science Department, Stanford university, Mar. 1970. URL: <http://i.stanford.edu/pub/cstr/reports/cs/tr/70/157/CS-TR-70-157.pdf>.

Algorithm 6 : $A \cdot B \cdot A^\top$ with block-diagonal scaling

Require: $A \in \mathbb{F}_q^{m \times n}$;

Require: A block diagonal matrix B , formed by 1-dimensional scalar blocks or 2-dimensional symmetric antitriangular blocks.

Ensure: $A \cdot B \cdot A^\top$ in $\mathbb{F}_q^{n \times n}$.

```

1: Form  $\bar{A} = A \in \mathbb{F}_q^{m \times n}$  and let  $\bar{D} = \text{IdentityMatrix}(n)$ ;
2: for all scalar blocks in  $B$  at position  $j$  do  $\bar{D}_j \leftarrow B_{j,j}$  end for
3: if  $q$  is odd then {Use Eq. (31)}
4:   for all symmetric antidiagonal blocks in  $B$  at  $(j, j+1)$  do
5:     Let  $\beta = B_{j,j+1} = B_{j+1,j}$ ;
6:      $\bar{D}_j \leftarrow \frac{1}{2}\beta$  and  $\bar{D}_{j+1} \leftarrow -\frac{1}{2}\beta$ ;
7:      $(\bar{A}_{*,i} \bar{A}_{*,j}) \leftarrow (\bar{A}_{*,i} \bar{A}_{*,j}) \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ ;
8:   end for
9: else
10:  for all antitriangular blocks in  $B$  at position  $(j, j+1)$  do
11:    Let  $\beta = B_{j,j+1} = B_{j+1,j}$  and let  $\delta = \text{FFSqrt}_{\mathbb{F}_q}(B_{j+1,j+1})$ ;
12:     $\bar{A}_{*,j} \leftarrow \beta \delta^{-1} \cdot \bar{A}_{*,j}$  {Scale column  $j$  of  $\bar{A}$ }
13:     $\bar{A}_{*,j+1} \leftarrow \delta \cdot \bar{A}_{*,j}$  {Scale column  $j+1$  of  $\bar{A}$ }
14:     $\bar{A}_{*,j+1} \leftarrow \bar{A}_{*,j+1} + \bar{A}_{*,j}$ ; {Use Eq. (32)}
15:    Swap columns  $j$  and  $j+1$  of  $\bar{A}$ ;
16:  end for
17:  if there are  $n/2$  antidiagonal blocks in  $B$  then {Use Eq. (33)}
18:    Let  $\beta = B_{1,2} = B_{2,1}$ ;
19:     $\bar{A}_{*,2} \leftarrow \beta \cdot \bar{A}_{*,2}$  and  $\bar{A} \leftarrow (\bar{A} \bar{A}_{*,1} + \bar{A}_{*,2}) \in \mathbb{F}_q^{m \times (n+1)}$ 
20:    Let  $\ell = 1$  and let  $\delta = 1$ ;
21:  else
22:    Let  $\ell$  be the index of a non antidiagonal block in  $B$ ;
23:    Let  $\delta = \text{FFSqrt}_{\mathbb{F}_q}(\bar{D}_\ell, \ell)$ ;
24:  end if
25:  for all remaining antidiagonal blocks in  $B$  at  $(j, j+1)$  do
26:    Let  $\beta = B_{j,j+1} = B_{j+1,j}$ ; {Use Eq. (34)}
27:     $\bar{A}_{*,\ell} \leftarrow \delta \cdot \bar{A}_{*,\ell}$  and  $\bar{A}_{*,j+1} \leftarrow \beta \cdot \bar{A}_{*,j+1}$ 
28:     $(\bar{A}_{*,\ell} \bar{A}_{*,j} \bar{A}_{*,j+1}) \leftarrow (\bar{A}_{*,\ell} \bar{A}_{*,j} \bar{A}_{*,j+1}) M$ ; and  $\delta \leftarrow 1$ ;
29:  end for
30: end if
31: return  $\bar{A} \cdot \bar{D} \cdot \bar{A}^\top$ . {Alg. 5}

```

- [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 2×2 matrix multiplication. *Inf. Processing Letters*, 56(6):329–335, Dec. 1995. doi:10.1016/0020-0190(95)00176-X.
- [7] P. Chatelin. On transformations of algorithms to multiply 2×2 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 2×2 -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 C. Arreche, editor, *ISSAC'2018, New York, USA*, pages 151–158, July 2018. doi:10.1145/3208976.3209019.
- [13] J.-C. Faugère. Fgb: A Library for Computing Gröbner Bases. In K. Fukuda, J. Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 84–87. Springer Berlin / Heidelberg, Sept. 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. *SIAM Journal on Matrix Analysis and Applications*, 13(3):681–687, 1992. doi:10.1137/0613043.
- [16] M. Kaminski, D. G. Kirkpatrick, and N. H. Bshouty. Addition requirements for matrix and transposed matrix products. *Journal of Algorithms*, 9(3):354–364, 1988. doi:10.1016/0196-6774(88)90026-0.
- [17] E. Karstadt and O. Schwartz. Matrix multiplication, a little faster. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'17*, pages 101–110. ACM, 2017. doi:10.1145/3087556.3087579.
- [18] F. Le Gall. Powers of tensors and fast matrix multiplication. In K. Nabeshima, editor, *ISSAC'2014, Kobe, Japan*, pages 296–303, July 2014. doi:10.1145/2608628.2608664.
- [19] 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.
- [20] G. Seroussi and A. Lempel. Factorization of symmetric matrices and trace-orthogonal bases in finite fields. *SIAM Journal on Computing*, 9(4):758–767, 1980. doi:10.1137/0209059.
- [21] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969. doi:10.1007/BF02165411.
- [22] S. Wedeniwski. Primality tests on commutator curves. PhD U. Tübingen, 2001.
- [23] S. Winograd. La complexité des calculs numériques. *La Recherche*, 8:956–963, 1977.
- [24] 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 $Y \cdot Y^\top = -I$. First,

$$U_3 = P_1 + P_2 = A_{11} \cdot A_{11}^\top + A_{12} \cdot A_{12}^\top = C_{11}. \quad (35)$$

Denote by R_1 the product:

$$\begin{aligned} R_1 &= A_{11} \cdot Y \cdot S_2^\top = A_{11} \cdot Y \cdot (A_{22}^\top - Y^\top \cdot A_{21}^\top) \\ &= A_{11} \cdot (Y \cdot A_{22}^\top + A_{21}^\top). \end{aligned} \quad (36)$$

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

$$\begin{aligned} U_1 &= P_1 + P_5 = A_{11} \cdot A_{11}^\top + S_3 \cdot S_3^\top \\ &= A_{11} \cdot A_{11}^\top + (S_2 + A_{11} \cdot Y) \cdot (S_2^\top + Y^\top \cdot A_{11}^\top) \\ &= S_2 \cdot S_2^\top + R_1^\top + R_1. \end{aligned} \quad (37)$$

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

$$\begin{aligned} S_2 \cdot S_2^\top &= (A_{22} - A_{21} \cdot Y) \cdot (A_{22}^\top - Y^\top \cdot A_{21}^\top) \\ &= A_{22} \cdot A_{22}^\top - A_{21} \cdot A_{21}^\top - R_2 - R_2^\top. \end{aligned} \quad (38)$$

Furthermore, from Equation (36):

$$\begin{aligned} R_1 + P_4 &= R_1 + S_1 \cdot S_2^\top \\ &= R_1 + (A_{21} - A_{11}) \cdot Y \cdot (A_{22}^\top - Y^\top \cdot A_{21}^\top) \\ &= A_{11} \cdot (Y \cdot A_{22}^\top + A_{21}^\top) + S_1 \cdot S_2^\top \\ &= A_{21} \cdot Y \cdot A_{22}^\top + A_{21} \cdot A_{21}^\top = R_2 + A_{21} \cdot A_{21}^\top. \end{aligned} \quad (39)$$

Therefore, from Equations (37), (38) and (39):

$$\begin{aligned} U_5 &= U_1 + P_4 + P_4^\top = S_2 \cdot S_2^\top + R_1 + R_1^\top + P_4 + P_4^\top \\ &= A_{22} \cdot A_{22}^\top + (-1 + 2)A_{21} \cdot A_{21}^\top = C_{22}. \end{aligned} \quad (40)$$

And the last coefficient U_4 of the result is obtained from Equations (39) and (40):

$$\begin{aligned} U_4 &= U_2 + P_3 = U_5 - P_4^\top + P_3 \\ &= U_2 + A_{22} \cdot (A_{12}^\top + Y^\top \cdot A_{21}^\top - Y^\top \cdot A_{11}^\top - A_{22}^\top) \\ &= A_{21} \cdot A_{21}^\top - P_4^\top + A_{22} \cdot (A_{12}^\top + Y^\top \cdot A_{21}^\top - Y^\top \cdot A_{11}^\top) \\ &= R_1^\top - R_2^\top + A_{22} \cdot (A_{12}^\top + Y^\top \cdot A_{21}^\top - Y^\top \cdot A_{11}^\top) \\ &= R_1^\top + A_{22} \cdot (A_{12}^\top - Y^\top \cdot A_{11}^\top) \\ &= A_{21} \cdot A_{11}^\top + A_{22} \cdot A_{12}^\top = C_{21}. \end{aligned} \quad (41)$$

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

A.2 Proof of Proposition 3.2

To prove Proposition 3.2 we need the result of the following Lemma, stating that in dimension larger than 2 it is impossible to compute any coefficient of the result with a single multiplication.

LEMMA A.1. *The dot-product of two vectors of dimension larger than 2 over a field cannot be computed by a bilinear algorithm with a single multiplication.*

PROOF. Let $(a_i)_{i=1\dots n}$ and $(b_j)_{j=1\dots n}$ be the given two vectors. Consider any linear combinations $\sum \lambda_i a_i$ and $\sum \mu_j a_j$ with the λ_i and μ_j as indeterminates. Suppose their product produces the dot-product $\sum_{k=1\dots n} a_k b_k$. By monomial identification, it would then mean that the following system is satisfied:

$$\lambda_k \mu_k = 1, \forall k = 1 \dots n, \quad \lambda_i \mu_j = 0, \forall i \neq j. \quad (42)$$

Over a field, the first set of equations implies that none of the λ_k and μ_k can be zero, while the second set of equations requires that some of them are. The equations are thus mutually incompatible and the lemma is proven. \square

PROPOSITION 3.2 (APPENDIX A.2). *9 block additions are necessary and sufficient to multiply a 2×2 block matrix over a field by its transpose with a bilinear algorithm that uses 5 multiplications.*

PROOF. From [16, Theorem 2], the minimal number of post-additions to get C is 7. So the respective number of block post-additions to get C_{11}, C_{12}, C_{21} and C_{22} are among the sets of four non-negative integers (i, j, k, ℓ) which sum to 7. Further, from Lemma A.1, there exists no combination that can provide either of the blocks of C without addition of multiplicative terms, so the four integers are actually strictly positive. This leaves $(4, 1, 1, 1), (3, 2, 1, 1)$ and $(2, 2, 2, 1)$'s permutations as candidates.

By symmetry, C_{12} is C_{21}^\top , thus a minimal realization must have an equal number of additions for C_{12} and C_{21} (otherwise compute one with the smallest and transpose it to obtain the other).

Then the maximal savings from the candidates is when that number of additions is 2. Therefore at least $2 + 2 + 1 = 5$ block post-additions are necessary to compute C from the products. As 4 additions are a minimum for the pre-additions [6, Lemma 9], we get a minimum of 9 block additions.

The sufficient condition is given by our Algorithm 2. \square

A.3 Proof of Proposition 3.3

PROPOSITION 3.3 (APPENDIX A.3). *Fields with characteristic two, or with an odd characteristic $p \equiv 1 \pmod{4}$, or finite fields that are an even extension, contain a square root of -1 .*

- PROOF. • If $p = 2$, then $1 = 1^2 = -1$.
- If $p \equiv 1 \pmod{4}$, 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 $x^{\frac{p^{2k}-1}{2} \frac{p^{k+1}}{2}}$ different from ± 1 and then the square of the latter must be -1 . \square

A.4 Proof of Proposition 3.4

PROPOSITION 3.4 (APPENDIX A.4). *Let \mathbb{F} be a field of characteristic p , there exists (a, b) in \mathbb{F}^2 such that the matrix:*

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \otimes I_n = \begin{pmatrix} a I_n & b I_n \\ -b I_n & a I_n \end{pmatrix} \text{ in } \mathbb{F}^{2n \times 2n} \quad (30)$$

is skew-orthogonal.

PROOF. First, remark that the following relation holds:

$$\begin{pmatrix} a I_n & b I_n \\ -b I_n & a I_n \end{pmatrix} \begin{pmatrix} a I_n & b I_n \\ -b I_n & a I_n \end{pmatrix}^\top = (a^2 + b^2) I_{2n}. \quad (43)$$

Second, if the characteristic is even, then $1^2 + 0^2 = -1$.

Third, if the characteristic is odd, then, in the base prime field, there are $\frac{p+1}{2}$ distinct square elements x_i^2 . 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 $-1 - x_i^2$ is equal to x_j^2 [20, Lemma 6].

Finally, let e.g. a be x_i and b be x_j , then we have the skew-orthogonal matrix: $Y = \begin{pmatrix} x_i & x_j \\ -x_j & x_i \end{pmatrix} \otimes I_n$. \square

A.5 Proof of Proposition 3.5

PROPOSITION 3.5 (APPENDIX A.5). *Algorithm 3 is correct and, under the ERH, runs in expected time $\tilde{O}(\log^3(p))$.*

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 $3 \log^2(p)/2 - 44 \log(p)/5 + 13$ [22, Theorem 6.35]. Thus the expected number of Legendre symbol computations is bounded by $O(\log^2(p))$ and this dominates the modular square root computations. \square

A.6 Proof of Theorem 4.1

THEOREM 4.1 (APPENDIX A.6). *Given ω such that $3 > \omega > 2$, if a generic matrix product algorithm requires less than $C_\omega n^\omega + o(n^\omega)$ operations then, over \mathbb{C} or over a prime characteristic field, Algorithm 2 can require less than $\frac{2}{2^\omega - 3} C_\omega n^\omega + o(n^\omega)$ arithmetic operations.*

PROOF. finite Suppose that a generic matrix multiplication algorithm requires less than $C_\omega n^\omega + o(n^\omega)$ operations with $3 > \omega > 2$. Then, on the one hand, use this algorithm to compute P_4 and P_5 , and on the other hand, recursively compute P_1, P_2 and P_7 .

If the field satisfies the conditions of Proposition 3.3, then with a random square root i of -1 , let Y be $i I_{n/2}$. Multiplication by Y requires n^2 multiplications by i , and let y be 1 .

Over the complex numbers, multiplications by i are just exchanging the real and imaginary parts and flipping one sign, so let $y = 0$.

Otherwise, in characteristic $p \equiv 3 \pmod{4}$, let the couple (a, b) be as in Proposition 3.4 and let Y be $\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \otimes I_{n/2}$. By the law of

quadratic reciprocity, $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$. Therefore, if $p \equiv 3 \pmod{4}$ then $\left(\frac{2}{p}\right) = -1$ if and only if $p \equiv 3 \pmod{8}$. Now, if $p \equiv 3 \pmod{4}$, then $\left(\frac{-1}{p}\right) = -1$. Thus, if $p \equiv 3 \pmod{8}$, then $\left(\frac{-2}{p}\right) = (-1) \cdot (-1) = 1$. We thus have two cases:

- (1) if $p \equiv 7 \pmod{8}$, then multiplication by Y requires $3n^2$ operations, and let y be 3 ;
- (2) if $p \equiv 3 \pmod{8}$, then we have shown that -2 is a square, so let $a = 1$ and $b \equiv \sqrt{-2} \pmod{p}$. Then $a^2 + b^2 = 1 - 2 = -1$

and multiplying by $\begin{pmatrix} 1 & b \\ -b & 1 \end{pmatrix} \otimes I_{n/2}$ requires $2n^2$ operations, and let y be 2.

Combining this with Remark 3.1, we get that overall the arithmetic complexity $T(n)$ of Algorithm 2 with the chosen Y satisfies:

$$T(n) \leq 3T(n/2) + 2C_\omega(n/2)^\omega + (7.5 + 2y)(n/2)^2 + o(n^2) \quad (44)$$

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

$$T(n) \leq \frac{2C_\omega}{2^\omega - 3} n^\omega + o(n^\omega) = \frac{2}{2^\omega - 3} \text{MM}_\omega(n) + o(n^\omega). \quad (45)$$

□

A.7 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 $A \cdot A^T$. 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 3.2, over \mathbb{C} , we can choose $Y = iI_n$. Then multiplications by i are just exchanging the real and imaginary parts. In Equation (44) 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.

A.8 Proof of Algorithm 4

PROOF. Given, α and β quadratic non-residues, (a, b) , such that $\alpha = a^2 + b^2$, are found by the algorithm of Remark 3.3. Second α and β are quadratic non-residues therefore their quotient is a residue since: $(\beta\alpha^{-1})^{\frac{q-1}{2}} = \frac{-1}{-1} = 1$. Third, if $c = -bda^{-1}$, 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 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, α is not a square, therefore it is non-zero and thus one of a or b must be non-zero. Then, $ac + bd = a(-dba^{-1}) + bd = -db + bd = 0$. Finally the matrix product $Y \cdot Y^T$ is $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} a^2+b^2 & ac+bd \\ ac+bd & c^2+d^2 \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}$. □