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 block-matrix by its transpose over C or any finite field using 5 recursive products. 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 space and time efficient schedules that enable us to provide fast practical implementations for higher-dimensional matrix products.

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 of $O(n^{2.81})$. This leads to fast generic matrix multiplication algorithm with complexity $\mathrm{MM}_\omega(n) = O(n^\omega)$ for $n \times n$ matrices (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 block-matrix by its transpose $A \cdot A^\mathsf{T}$ using only 5 multiplications over some base field, instead of 6 for the natural divide & conquer algorithm. For such a product with the best previously known complexity was dominated by $\frac{2}{2^\omega-4}\mathrm{MM}_\omega(n)$ over any base field (see e.g. [11, § 6.3.1]). Here, we etablish the following result:

Theorem 1.1. *There exists an algorithm that computes the product of a block-matrix by its transpose with a complexity dominated by $\frac{2}{2^\omega-3}MM_\omega(n)$ over a base field for which there exists either a square root of -1 or a skew-orthogonal matrix.*

Symmetric rank k update. We focus on the computation of the product of an $n \times k$ block-matrix by its transpose and possibly accumulating the result to another matrix. Although this product is sometimes called the Gram matrix, we prefer to avoid this terminology as, over the complex numbers, it refers the product with the Hermitian conjugate. Instead we will follow terminology introduced in the BLAS3 standard [10] and refer to the operation as a symmetric rank k update (SYRK).

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]):

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

In order to consider above 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 $\langle M, N\rangle = \mathrm{Trace}(M^\mathsf{T} \cdot N)$. Hence, this inner product establishes an isomorphism between $\mathbb{K}^{a\times b}$ and its dual space $(\mathbb{K}^{a\times b})^\star$ allowing for example to associate matrix multiplication and the trilinear form $\mathrm{Trace}(X \cdot Y \cdot Z)$:

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

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:

$$\begin{array}{c} \sum_{i=1}^7 S_{i1}\otimes S_{i2}\otimes S_{i3} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\otimes\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}\otimes\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}+ \\ \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}\otimes\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\otimes\begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}+\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}\otimes\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\otimes\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}+ \\ \begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}\otimes\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}\otimes\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}+\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\otimes\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\otimes\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}+ \\ \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\otimes\begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}\otimes\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}+\begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}\otimes\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}\otimes\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \tag{4}$$

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 2×2-matrices, one can explicitly retrieve from tensor $\mathcal{S}$ the Strassen matrix multiplication algorithm computing $A \cdot B$ by the *partial* contraction $\{\mathcal{S}, A \otimes B\}$:

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

while the *complete* contraction $\{\mathcal{S}, A \otimes B \otimes C\}$ 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:

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

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 ±1 and $\mathfrak{S}_3$ for the symmetric group on 3 elements.*

Hence, given any matrices U, V and W, 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:*

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

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 2×2-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 2×2-matrix multiplication.*

Thus, action of an isotropy on Strassen tensor maybe defines matrix product algorithm with interesting computational properties.

2.1 Design of a specific 2×2-matrix product

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

Strategy 2.1. *By applying a undetermined isotropy:*

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

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

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

Then, we could impose further conditions on these algorithms and check by a Gröbner basis computation if such an algorithm exist.

If so, there is a subsequent work to do in order to choose a point on this variety; this choice can be motivated by the additive complexity 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) don't increase the additive complexity 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 denotes 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 of first and second components are equal ([6, eq. (11)]) and at least $7-3$. 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

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

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

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

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 original Strassen algorithm requires 18 additions, only 15 additions are necessary in the Winograd Algorithm 1.

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

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

1: 8 additions:

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

2: 7 recursive multiplications:

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

3: 7 final additions:

$$c_1\leftarrow p_1+p_5,\quad c_2\leftarrow c_1+p_4,\quad c_3\leftarrow p_1+p_2,\quad c_4\leftarrow c_2+p_3,$$
$$c_5\leftarrow c_2+p_7,\quad c_6\leftarrow c_1+p_7,\quad c_7\leftarrow c_6+p_6.$$

4: **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 $\{1,\dots,7\}$ and imposing $T_{i1}=T_{i2}$ as constraints with i in J (see [3, eq 4]) save $|J|$ operations and thus reduce the computational complexity.*

The definition (9) of $\mathcal{T}$, these constraints and the fact that U, V, W's determinant is 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:

$$\mathsf{g} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{12}$$

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 2×2 matrix product $A \cdot A^\intercal$.

2.2 Two-dimensional symmetric matrix product with 5 multiplications

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

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

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

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

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

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

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

The constraints of type J allow to save preliminary additions when applying to matrices $B = A^\intercal$: since then operations on A and $A^\intercal$ will be the same. The constraints of type H allow also to save multiplications especially when dealing with 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 cardinal of H is at most 2 and then the cardinal 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 $z = v_{11} - v_{21}$:

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

Remark 2.3. *As $\left((z+v_{21})^2 + {v_{21}}^2\right)^2 + 1 = 0$ occurs in this paramaterization, 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 product and use our algorithm directly in any finite field.*

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.*

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 the following constraints to the system for a subset K of $\{1,\dots,7\}$:

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

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:

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

3 FAST BLOCK RECURSIVE SYRK

3.1 Higher-dimensional algorithm

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 because these coefficients represent now matrices and 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 $Y^{\mathsf{T}} = -Y^{-1}$ holds.*

If Y is skew-orthogonal, then of the 7 recursive matrix products involved in Algorithm 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.

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 $C = A \cdot A^{\mathsf{T}} = \begin{pmatrix} C_{11} & C_{21}{}^{\mathsf{T}} \\ C_{21} & C_{22} \end{pmatrix}$.

1: 4 additions and 2 multiplications by Y:

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

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

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

3: 2 symmetric additions (half additions);

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

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

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

5: 1 half addition ($U_5 = U_1 + P_4 + P_4{}^{\mathsf{T}}$ is symmetric):

$$\mathrm{Low}(U_5) \leftarrow \mathrm{Low}(U_2) + \mathrm{Low}(P_4{}^{\mathsf{T}}). \quad (29)$$

6: **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.*

3.2 Minimality of the number of additions

From Lemma 2.1, we know that 11 additions are minimal to compute $A \cdot A^{\mathsf{T}}$ from 7 multiplications in generic 2×2 matrices. Here we are considering matrices over a field, therefore $C = A \cdot A^{\mathsf{T}}$ 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.*

Remark 3.1. *Symmetry of the blocks C_{11} and C_{22} gives a candidate minimal number of extra additions to be performed by a 5 multiplications algorithm: apart from those involved in the block multiplications, $7.5n^2 + 1.5n$ additions are sufficient to multiply a $2n \times 2n$ block matrix over a field by its transpose with a bilinear algorithm that uses 5 multiplications. Indeed, following the line of proof of Proposition 3.2, in a $(2, 2, 2, 1)$ configuration, C_{11} and C_{22} must require 1 block addition for one and 2 block additions for the other. The symmetry of each of these blocks gives that only, say, the lower part needs to be computed. This is $1.5(n+1)n$ instead of $3n^2$ operations, for a total of $4n^2 + 2n^2 + 1.5n^2 + 1.5n = 7.5n^2 + 1.5n$, 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.

3.3 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.

3.3.1 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. Now, even though over the complex the product of a matrix by its *conjugate* transpose (ZHERK) is more widely used, ZSYRK has some applications, see for instance [1].

3.3.2 Minus one is a square. Now, over some finite fields, square roots of -1 can be elements of the base field, denoted i in $\mathbb{F}$ again. There, Algorithm 2 will only require some pre-multiplications by this square root (with also $Y = i\,\mathrm{I}_n \in \mathbb{F}^{n\times n}$), but *within the field*. The following Proposition 3.3 characterize these finite fields.

Proposition 3.3 (Appendix A.3). *Finite fields with even characteristic, or with an odd characteristic $p \equiv 1 \bmod 4$, or that are an even extension, contain a square root of -1.*

3.3.3 Any finite field. Finally, we show that Algorithm 2 can also be run without any field extension, even when -1 is not a square. There, we can form the skew-orthogonal matrices constructed in Proposition 3.4 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 $A \cdot A^{\mathsf{T}} = \begin{pmatrix} A & 0 \end{pmatrix} \cdot \begin{pmatrix} A^{\mathsf{T}} \\ 0 \end{pmatrix}$.

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

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \otimes \mathrm{I}_n = \begin{pmatrix} a\,\mathrm{I}_n & b\,\mathrm{I}_n \\ -b\,\mathrm{I}_n & a\,\mathrm{I}_n \end{pmatrix} \text{ in } \mathbb{F}^{2n\times 2n} \tag{30}$$

is skew-orthogonal.

Proposition 3.4 shows that skew-orthogonal matrices do exist for any finite field. For Algorithm 2, we need to build them mostly for $p \equiv 3 \bmod 4$ (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 $r \equiv 1 \bmod 4$ and $r \equiv -1 \bmod p$ hold;
(2) Thus, results of [5] allow 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 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 only if the ERH is true.

Algorithm 3 : Sum of squares modulo prime

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

```
if (k/p) == 1 then                                  {k is a square mod p}
    return (ModSquareRoot(k, p), 0).
else
    s = 2; while (s/p) == 1 do  {Lowest quadratic non-residue}
        s = s + 1;
    end while
end if
c = ModSquareRoot(s − 1, p);                        {s − 1 must be a square}
r ≡ k s^{-1} mod p;
a = ModSquareRoot(r, p);  {Now k ≡ a^2 s ≡ a^2(1 + c^2) mod p}
return (a, ac mod p).
```

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

Remark 3.2. *When computing t sum of squares modulo the same prime, one can of course compute the lowest quadratic non-residue (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 3.3. *Another possibility is to use randomization: instead of using the LQNR, randomly select a non-residue s, and then decrement it until $s - 1$ is a quadratic residue (as 1 is a square, this will terminate)*[1].

Remark 3.4. *Except in characteristic 2, 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 $\mathsf{FFSoS}_{\mathbb{F}_q}(k)$ this algorithm decomposing k as a sum of squares within any finite field $\mathbb{F}_q$.*

[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 (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 a finite field, Algorithm 2 can require less than $\frac{2C_\omega}{2^\omega - 3} n^\omega + o(n^\omega)$ arithmetic operations.*

Remark 4.1. *Up 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 complexity improvment in different cases.*

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	$\frac{2}{3}\,MM_{\log_2(7)}(n)$	$\frac{2}{2^\omega-4}\,MM_\omega(n)$
	Alg. 2	$0.8n^3$	$\frac{1}{2}\,MM_{\log_2(7)}(n)$	$\frac{2}{2^\omega-3}\,MM_\omega(n)$

Table 1: Complexity bounds leading term over finite fields.

Remark 4.2. *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 first $G = (A + B) \cdot (A^\mathsf{T} - B^\mathsf{T})$ and second, $H = A \cdot B^\mathsf{T}$, then $(A + iB) \cdot (A^\mathsf{T} + iB^\mathsf{T}) = (G - H^\mathsf{T} + H) + i(H + H^\mathsf{T})$. This symmetric 2M 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. That is $\frac{6}{2^\omega-4}RR_\omega + o(n^\omega)$, using the 3M method for the generic complex floating point products.

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.	$O(n^3)$	$O\left(n^{\log_2(7)}\right)$	$O(n^\omega)$
$A \cdot B \in \mathbb{C}^{n\times n}$	naive	$8n^3$	$4\,\mathrm{RR}_{\log_2(7)}(n)$	$4\,\mathrm{RR}_\omega(n)$
	3M	$6n^3$	$3\,\mathrm{RR}_{\log_2(7)}(n)$	$3\,\mathrm{RR}_\omega(n)$
$A \cdot A^\mathsf{T} \in \mathbb{C}^{n\times n}$	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: 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. We propose in Table 3 and Figure 1 a schedule for the operation $C \leftarrow A \cdot A^\mathsf{T}$ using no more extra storage than the unused upper triangular part

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

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

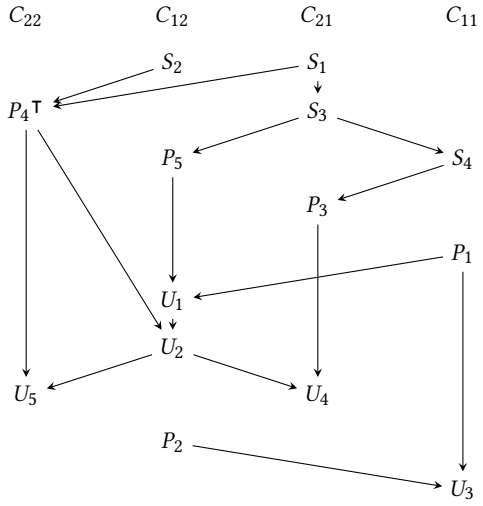

Figure 1: DAG of the tasks and their memory location for the computation of $C \leftarrow A \cdot A^\mathsf{T}$ presented in Table 3.

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

of the result C. For the more general operation $C \leftarrow \alpha A \cdot A^\mathsf{T} + \beta C$, Table 4 and Figure 2 propose a schedule requiring only an additional $n/2 \times n/2$ temporary storage. These algorithms have been

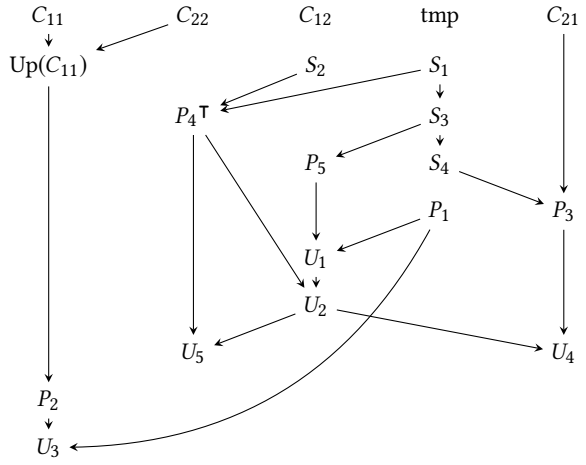

Figure 2: DAG of the tasks and their memory location for the computation of $C \leftarrow \alpha A \cdot A^\mathsf{T} + \beta C$ presented in Table 4.

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-5.2.17 GNU/Linux system.

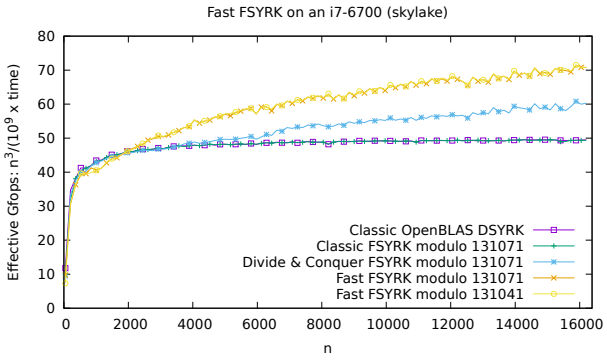

Figure 3: Speed of an implementation of Algorithm 2

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 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 $C - A \cdot D \cdot A^{\mathsf{T}}$ IN THE BASE FIELD

The main source of speed in the $L \cdot D \cdot L^{\mathsf{T}}$ Symmetric indefinite triangular factorization is the internal computation of symmetric matrix products with a symmetric block diagonal matrix D inside (formed by 1-dimensional or 2-dimensional blocks) [12]. The main routine is then the in-place update of a symmetric matrix C via $C \leftarrow C - A \cdot D \cdot A^{\mathsf{T}}$.

Extending the factorization of non quadratic residues of Algorithm 3, one can actually factor D into $D = \Delta \cdot \Delta^{\mathsf{T}}$, *without needing any field extension*, and then compute instead $(A \cdot \Delta) \cdot (A \cdot \Delta)^{\mathsf{T}}$. This is what we propose in this section. Indeed, this is better to deal beforehand 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 the following:

- Be careful to perform recursive calls only on blocks with even dimensions;
- In odd characteristic, using field extensions could be expensive, so Section 5.1 shows a way to factor a diagonal matrix in the base field;
- Sections 5.2.1 and 5.2.2 show how to deal with antidiagonal or antitriangular 2×2 blocks, depending on the characteristic.

5.1 Factoring non squares within the 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. $Y \cdot Y^{\mathsf{T}} = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}$.

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

Proposition 5.1 (Appendix A.7). *Algorithm 4 is correct.*

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 $A \cdot D \cdot A^{\mathsf{T}}$ with D a diagonal matrix: factor D with a tridiagonal matrix $D = \Delta \cdot \Delta^{\mathsf{T}}$, 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 from matrix multiplication is only $O(n^2)$ (that is $O(n)$ square roots and $O(n)$ column scalings).

Algorithm 5 : $A \cdot D \cdot A^{\mathsf{T}}$: SYRK with a diagonal over a finite field

Require: $A \in {\mathbb{F}_q}^{m\times n}$ and $(d_1, \ldots, d_n) \in {\mathbb{F}_q}^n$.
Ensure: $A \cdot \mathtt{DiagonalMatrix}(d_1, \ldots, d_n) \cdot A^{\mathsf{T}}$ 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 $\bar{D} = \mathtt{DiagonalMatrix}(d_1, \ldots, d_n, d_\ell) \in {\mathbb{F}_q}^{(n+1)\times(n+1)}$
4: $\bar{A} = (\,A\ 0\,) \in {\mathbb{F}_q}^{m\times(n+1)}$ {Augment A with a zero column}
5: **else**
6: $\bar{D} = D = \mathtt{DiagonalMatrix}(d_1, \ldots, d_n) \in {\mathbb{F}_q}^{n\times n}$;
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: $\bar{A}_{*,j} \leftarrow \mathtt{FFSqrt}_{\mathbb{F}_q}(d_j) \cdot \bar{A}_{*,j}$ {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** $\bar{A} \cdot \bar{A}^{\mathsf{T}}$. {Alg. 2}

5.2 Antidiagonal and antitriangular blocks

In a generic $L \cdot D \cdot L^{\mathsf{T}}$ factorization, antitriangular or antidiagonal blocks can appear in D. 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 handle 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 resume to the diagonal case (note the requirement that 2 is invertible).

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} \frac{1}{2}\beta & 0 \\ 0 & -\frac{1}{2}\beta \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{\mathsf{T}} = \begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix}. \tag{31}$$

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):

$$\begin{pmatrix} 0 & \beta \\ \beta & \gamma \end{pmatrix} = \begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & \frac{\beta}{\sqrt{\gamma}} \\ \sqrt{\gamma} & 0 \end{pmatrix}\begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & \frac{\beta}{\sqrt{\gamma}} \\ \sqrt{\gamma} & 0 \end{pmatrix}^{\mathsf{T}} = \left(\begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & 0 \\ 0 & \sqrt{\gamma} \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\right)\left(\begin{pmatrix} \frac{\beta}{\sqrt{\gamma}} & 0 \\ 0 & \sqrt{\gamma} \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\right)^{\mathsf{T}}. \tag{32}$$

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

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):

$$\begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \left(\begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \right)^{\mathsf{T}} = \begin{pmatrix} 0 & \beta \\ \beta & 0 \end{pmatrix} \bmod 2. \tag{33}$$

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 $\sqrt{x}, 1, \beta$, the transformation $M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ and compute $D_{\sqrt{x},1,\beta} \cdot M \cdot M^{\mathsf{T}} \cdot D_{\sqrt{x},1,\beta}{}^{\mathsf{T}}$ that is:

$$\begin{pmatrix} \sqrt{x} & \sqrt{x} & \sqrt{x} \\ 1 & 0 & 1 \\ 0 & \beta & \beta \end{pmatrix} \begin{pmatrix} \sqrt{x} & \sqrt{x} & \sqrt{x} \\ 1 & 0 & 1 \\ 0 & \beta & \beta \end{pmatrix}^{\mathsf{T}} = \begin{pmatrix} x & 0 & 0 \\ 0 & 0 & \beta \\ 0 & \beta & 0 \end{pmatrix} \bmod 2. \tag{34}$$

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 block. 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, whose any of the 3 ones can be used afterwards as x in any further combination with the next antidiagonal blocks.

Algorithm 6 summarizes the use of Equations (31), (32) and (33).

Algorithm 6 : $A \cdot B \cdot A^{\mathsf{T}}$ with block-diagonal scaling

Require: $A \in \mathbb{F}_q{}^{m \times\times n}$;
Require: A block diagonal matrix B, formed by 1-dimensional scalar blocks or 2-dimensional symmetric antitriangular blocks.
Ensure: $A \cdot B \cdot A^{\mathsf{T}}$ in $\mathbb{F}_q{}^{n \times n}$.

Form $\bar{A} = A \in \mathbb{F}_q{}^{m \times n}$ and $\bar{D} =$ `IdentityMatrix`(n);
for all scalar blocks in B at position j **do**
  $\bar{D}_j \leftarrow B_{j,j}$;
end for
if q is odd **then** {Use Eq. (31)}
  for all symmetric antidiagonal blocks in B at $(j, j+1)$ **do**
    Let $\beta = B_{j,j+1} = B_{j+1,j}$;
    $\bar{D}_j \leftarrow \frac{1}{2}\beta$;
    $\bar{D}_{j+1} \leftarrow -\frac{1}{2}\beta$;
    $(\bar{A}_{*,i}\ \bar{A}_{*,j}) \leftarrow (\bar{A}_{*,i}\ \bar{A}_{*,j}) \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$;
  end for
else
  for all antitriangular blocks in B at position $(j, j+1)$ **do**
    Let $\beta = B_{j,j+1} = B_{j+1,j}$ and $\delta = \mathtt{FFSqrt}_{\mathbb{F}_q}(B_{j+1,j+1})$; {Use Eq. (32)}
    $\bar{A}_{*,j} \leftarrow \beta\delta^{-1} \cdot \bar{A}_{*,j}$ {Scale column j of $\bar{A}$}
    $\bar{A}_{*,j+1} \leftarrow \delta \cdot \bar{A}_{*,j}$ {Scale column $j+1$ of $\bar{A}$}
    $\bar{A}_{*,j+1} \leftarrow \bar{A}_{*,j+1} + \bar{A}_{*,j}$;
    Swap columns j and $j+1$ of $\bar{A}$;
  end for
  if there are $n/2$ antidiagonal blocks in B **then** {Use Eq. (33)}
    Let $\beta = B_{1,2} = B_{2,1}$;
    $\bar{A}_{*,2} \leftarrow \beta \cdot \bar{A}_{*,2}$
    $\bar{A} \leftarrow (\bar{A}\ \bar{A}_{*,1}{+}\bar{A}_{*,2}) \in \mathbb{F}_q{}^{m \times (n+1)}$
    Let $\ell = 1$ and $\delta = 1$;
  else
    Let ℓ be the index of a non antidiagonal block in B;
    Let $\delta = \mathtt{FFSqrt}_{\mathbb{F}_q}(\bar{D}_{\ell,\ell})$;
  end if
  for all remaining antidiagonal blocks in B at $(j, j+1)$ **do**
    Let $\beta = B_{j,j+1} = B_{j+1,j}$; {Use Eq. (34)}
    $\bar{A}_{*,\ell} \leftarrow \delta \cdot \bar{A}_{*,\ell}$
    $\bar{A}_{*,j+1} \leftarrow \beta \cdot \bar{A}_{*,j+1}$
    $(\bar{A}_{*,\ell}\ \bar{A}_{*,j}\ \bar{A}_{*,j+1}) \leftarrow (\bar{A}_{*,\ell}\ \bar{A}_{*,j}\ \bar{A}_{*,j+1})\, M$;
    $\delta \leftarrow 1$;
  end for
end if
return $\bar{A} \cdot \bar{D} \cdot \bar{A}^{\mathsf{T}}$. {Alg. 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. *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 Departement. Standford university, Mar. 1970. URL: http://i.stanford.edu/pub/cstr/reports/cs/tr/70/157/CS-TR-70-157.pdf.

[5] J. Brillhart. Note on representing a prime as a sum of two squares. *Mathematics 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 Transactions on Mathematical Software*, 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 Transactions on Mathematical Software*, 35(3):1–42, Nov. 2008. URL: http://hal.archives-ouvertes.fr/hal-00018223, 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. URL: https://hal.archives-ouvertes.fr/hal-01704793, doi:10.1145/3208976.3209019.

[13] J.-C. Faugère. FGb: A Library for Computing GrÃűbner 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] T. F.-F. group. *FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package*, v2.4.1 edition, 2019. 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 Journal 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 thesis, 2001.
[23] S. Winograd. La complexité des calculs numériques. *La Recherche*, 8:956–963, 1977.
[24] Z. Xianyi and M. K. et al. *OpenBLAS, an Optimzed 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,

$$U_3 = P_1 + P_2 = A_{11} \cdot A_{11}^\intercal + A_{12} \cdot A_{12}^\intercal = C_{11}. \tag{35}$$

Denote by R_1 the product:

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

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

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

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

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

Furthermore, from Equation (36):

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

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

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

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

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

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 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 $(b_j)_{j=1\dots n}$ bet 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:

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

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. □

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 ${C_{21}}^{\mathsf{T}}$, 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. □

A.3 Proof of Proposition 3.3

Proposition 3.3 (Appendix A.3). *Finite fields with even characteristic, or with an odd characteristic $p \equiv 1 \bmod 4$, or 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 cardinal p^{2k}, then, similarly, many elements $x^{\frac{p^k-1}{2}\frac{p^k+1}{2}}$ are different from ± 1 and then the square of the latter must be -1. □

A.4 Proof of Proposition 3.4

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

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \otimes \mathrm{I}_n = \begin{pmatrix} a\,\mathrm{I}_n & b\,\mathrm{I}_n \\ -b\,\mathrm{I}_n & a\,\mathrm{I}_n \end{pmatrix} \text{ in } \mathbb{F}^{2n \times 2n} \tag{30}$$

is skew-orthogonal.

Proof. First, remark that the following relation holds:

$$\begin{pmatrix} a\,\mathrm{I}_n & b\,\mathrm{I}_n \\ -b\,\mathrm{I}_n & a\,\mathrm{I}_n \end{pmatrix} \begin{pmatrix} a\,\mathrm{I}_n & b\,\mathrm{I}_n \\ -b\,\mathrm{I}_n & a\,\mathrm{I}_n \end{pmatrix}^{\mathsf{T}} = (a^2 + b^2)\,\mathrm{I}_{2n}. \tag{43}$$

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

Third, if the characteristic is odd, then, in the base 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: $Y = \begin{pmatrix} x_i & x_j \\ -x_j & x_i \end{pmatrix} \otimes \mathrm{I}_n$. □

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 $\widetilde{O}(\log^3(p))$.*

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 $k = a^2\left(1 + b^2\right) = a^2 + (ab)^2$ 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 $O\left(\log^2(p)\right)$ and this dominates the modular square root computations. □

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 a finite field, Algorithm 2 can require less than $\frac{2C_\omega}{2^\omega - 3} n^\omega + o(n^\omega)$ arithmetic operations.*

Proof. Suppose that a generic matrix multiplication algorithm requires less than $C_\omega n^\omega + o(n^\omega)$ operations with $\log_2(3) > \omega > 2$. 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 finite field satisfy the conditions of Proposition 3.3, then with a random square root i of -1, let Y be $i\,\mathrm{I}_{n/2}$. 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, let the couple (a, b) be as in Proposition 3.4 and let Y be $\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \otimes \mathrm{I}_{n/2}$. Multiplication by Y requires $3n^2$ operations, and let y be 3.

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

$$T(n) \leq 3T(n/2) + 2C_\omega(n/2)^\omega + (7.5 + 2y)(n/2)^2 + o\left(n^2\right) \tag{44}$$

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

$$T(n) \leq \frac{2C_\omega}{2^\omega - 3} n^\omega + o\left(n^\omega\right) = \frac{2}{2^\omega - 3}\mathrm{MM}_\omega(n) + o\left(n^\omega\right). \tag{45}$$

□

A.7 Proof of Proposition 5.1

Proposition 5.1 (Appendix A.7). *Algorithm 4 is correct.*

Proof. First, the equality $\alpha = a^2 + b^2$ holds via the algorithm of Remark 3.4. Second α and β are quadratic non-residues therefore their quotient is a residue since: $\left(\beta\alpha^{-1}\right)^{\frac{q-1}{2}} = \frac{-1}{-1} = 1$. Third, $c^2 + d^2$ 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 $Y \cdot Y^\intercal$ is $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)\left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right) = \left(\begin{smallmatrix} a^2+b^2 & ac+bd \\ ac+bd & c^2+d^2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} \alpha & 0 \\ 0 & \beta \end{smallmatrix}\right)$. □

A.8 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
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
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
G3-i					5817	34298	**225108**	1603112
Syrk-i					6908	40112	260192	**1838528**
G0-i	4	3				36099	221390	**1500540**
G1-i						37499	226990	**1522940**
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/green when ours (using Strassen-Winograd with $i-1$ recursive levels, and G0-i for $\mathbb{C}$ / G1-i if -1 is a square / G3-i otherwise) is better than others.

First, from Section 3.3, 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 (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.