Parallel Memory-Independent Communication Bounds for SYRK

Hussam Al Daas
Rutherford Appleton Laboratory
Didcot, Oxfordshire, UK
hussam.al-daas@stfc.ac.uk

Grey Ballard
Wake Forest University
Winston-Salem, NC, USA
ballard@wfu.edu

Laura Grigori
Inria Paris
Paris, France
laura.grigori@inria.fr

Suraj Kumar
Inria Lyon
Lyon, France
suraj.kumar@inria.fr

Kathryn Rouse
Inmar Intelligence
Winston-Salem, NC, USA
kathryn.rouse@inmar.com

ABSTRACT

In this paper, we focus on the parallel communication cost of multiplying a matrix with its transpose, known as a symmetric rank-k update (SYRK). SYRK requires half the computation of general matrix multiplication because of the symmetry of the output matrix. Recent work (Beaumont et al., SPAA '22) has demonstrated that the sequential I/O complexity of SYRK is also a constant factor smaller than that of general matrix multiplication. Inspired by this progress, we establish memory-independent parallel communication lower bounds for SYRK with smaller constants than general matrix multiplication, and we show that these constants are tight by presenting communication-optimal algorithms. The crux of the lower bound proof relies on extending a key geometric inequality to symmetric computations and analytically solving a constrained nonlinear optimization problem. The optimal algorithms use a triangular blocking scheme for parallel distribution of the symmetric output matrix and corresponding computation.

CCS CONCEPTS

• **Theory of computation → Parallel algorithms**.

KEYWORDS

Symmetric matrices, Communication costs, Convex optimization

ACM Reference Format:
Hussam Al Daas, Grey Ballard, Laura Grigori, Suraj Kumar, and Kathryn Rouse. 2023. Parallel Memory-Independent Communication Bounds for SYRK. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '23), June 17–19, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3558481.3591072

1 INTRODUCTION

The symmetric rank-k update computation, known by the acronym SYRK, takes a single matrix $\mathbf{A}$ as input and outputs the multiplication of $\mathbf{A}$ with its transpose, $\mathbf{C} = \mathbf{A}\mathbf{A}^T$, which is a symmetric matrix. The computation gets its name from its use as a subroutine within algorithms for computing the Cholesky decomposition of a symmetric positive-definite matrix, and it is one of the level-3 Basic Linear Algebra Subroutines (BLAS). SYRK is a useful operation for many other applications: the result $\mathbf{C}$ is known as the covariance matrix in the context of statistical data, and it is also called the Gram matrix in the context of numerical linear algebra. The SYRK computation is often the computational bottleneck for solving linear least squares problems via the normal equations, computing a QR factorization using the Cholesky QR algorithm, or computing singular values and vectors via the Gram SVD algorithm. Because of its fundamental importance and inclusion in the BLAS, SYRK is a well-optimized kernel on the full range of sequential and parallel computational platforms.

Mathematically, SYRK is equivalent to general matrix multiplication (GEMM), but because the output matrix is symmetric, algorithms can save half the computation compared to GEMM by computing only the lower or upper triangle. In the context of Cholesky decomposition algorithms, $\mathbf{A}$ is typically tall and skinny, so that the output $\mathbf{C}$ is a much larger matrix. For applications like the normal equations, $\mathbf{A}$ is short and wide, so the output $\mathbf{C}$ is a smaller matrix.

In this work, we focus on the communication costs of parallel algorithms for SYRK. The main contributions of our work are to

(1) establish new communication lower bounds that apply for all ranges of input matrix dimensions and numbers of processors,
(2) present new parallel algorithms for SYRK with parameters that can be tuned to minimize communication cost, and
(3) prove that lower bounds and algorithms are optimal in all ranges, with matching constants in the leading order terms.

The I/O complexity and communication bounds for SYRK have been well studied. A simple reduction argument shows that the communication cost of SYRK with square input is at least that of GEMM (up to a constant factor): to multiply matrices $\mathbf{A}$ and $\mathbf{B}$, compute SYRK for the matrix $\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{0} & \mathbf{B}^T \end{bmatrix}$. Thus, classical results for matrix multiplication [14, 15] extend to SYRK. These results imply that for SYRK with square input of dimension n has sequential I/O complexity of $\Omega(n^3/\sqrt{M})$, where M is the size of the fast memory, and parallel communication cost of $\Omega(n^2/P^{2/3})$, where P is the number of processors. Ballard et al. [3–5] have given direct proofs for a set of computations that include SYRK, but the leading order constants are not tight. In the case of GEMM, Smith et al. [21] tightened the constant for sequential I/O complexity,

SPAA '23, June 17–19, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9545-8/23/06…$15.00
https://doi.org/10.1145/3558481.3591072

and Al Daas et al. [2] tightened the constants for parallel memory-independent communication lower bounds. Beaumont et al. [7] establish the I/O complexity of SYRK with input dimensions $n_1 \times n_2$ as $(1/\sqrt{2}) \cdot n_1^2 n_2/\sqrt{M}$, which is a factor of $2^{3/2}$ smaller than that of GEMM, and they show that the leading constant is tight by providing an optimal sequential algorithm. Sequential communication lower bounds can be extended to memory-dependent parallel bounds in a straightforward way (interpreting the fast memory size M as a local memory size), but the parallel bounds are typically not attainable when $\mathbf{A}$ is sufficiently rectangular or either P or M is sufficiently large. The parallel lower bounds we establish for SYRK in this work are attainable, given sufficient memory, and each is a factor of 2 smaller than the corresponding bound for GEMM.

Our work builds primarily on two of the aforementioned results. Al Daas et al. [2] use a constrained optimization approach involving projections of a processor's assigned computations onto the matrix data to prove communication lower bounds for rectangular matrix multiplication. The constraints on the projections are derived from the Loomis-Whitney inequality [17] as well as individual bounds on the projections, and these constraints yield three lower bounds that apply across different ranges of relative matrix dimensions and number of processors. Al Daas et al. present three types of algorithms, called 1D, 2D, and 3D corresponding to the number of dimensions of the computational space that are partitioned, that obtain the lower bounds in each of the three cases. In this work, we use the same general approach to prove the lower bound for SYRK in § 4, and we use the same classification scheme to describe our optimal algorithms. The key difference is that the Loomis-Whitney inequality does not provide a constraint sufficient to obtain optimal constants for SYRK. We develop a new result (Lemma 3), based on Loomis-Whitney, that is specialized to the structure of SYRK.

Beaumont et al. [7] prove sequential lower bounds for SYRK, and they also propose a novel cache-blocking scheme in order to obtain an optimal sequential algorithm. Their scheme is based on triangle blocks, such as those that occur naturally on the diagonal of a standard cache-blocking scheme for symmetric matrices. They illustrate that for SYRK, triangle blocks exhibit a higher operational intensity (computation-to-data ratio) than standard cache blocks because of the symmetry of the data. Further, they show how a symmetric matrix can be partitioned into triangle blocks so that the entire sequential computation can achieve the higher operational intensity and attain the constant of the lower bound. As we describe in § 5, we use the triangle block partitioning scheme to define a parallel distribution for 2D and 3D algorithms that attain the constants of the parallel lower bounds. The key difference is that our algorithms are parallel, and the triangle block partitioning is used to define a parallel data distribution rather than sequential cache blocks.

Algorithms for SYRK have long exploited the symmetry to reduce the computational cost by a factor of 2 compared to GEMM. As mentioned earlier, Beaumont et al. [7] demonstrate that in the sequential context, the communication cost of SYRK is a factor of $2^{3/2}$ smaller than matrix multiplication, at least in theory. The main conclusion from our work is that in the parallel case, the communication cost of SYRK is a factor of 2 smaller than GEMM. State-of-the-art library implementations of SYRK such as ScaLAPACK [8] or Elemental [20] do not achieve this reduction; they halve the computation but communicate the same amount of data as GEMM along the critical path. While our results are currently only theoretical and involve some assumptions on the number of processors, we believe future work can demonstrate that they will outperform GEMM in practice by close to that theoretical factor, whether the time is computation or communication bound.

2 RELATED WORK

The first communication lower bounds were presented by Hong and Kung [14] who used the red-blue pebble game to develop bounds for many sequential computations including matrix multiplication. Aggarwal et al. [1] extended their matrix multiplication results to the parallel LPRAM model and provided the first memory independent communication lower bounds. Irony et al. [15] reproduced previous matrix multiplication bounds by applying the Loomis-Whitney inequality [17]. Ballard et al. [3] extended the use of the Loomis-Whitney inequality to derive bounds for any computation that can be written as three nested loops. Dongarra et al. [13] improved the constant for sequential lower bounds for GEMM, which was further improved by Smith et al. [21]. Demmel et al. [12] derived three parallel memory-independent tight communication lower bounds for rectangular matrix multiplication based on aspect ratios of matrices. Al Daas et al. [2] applied constrained optimization techniques to improve the constants for all three parallel memory independent matrix multiplication lower bounds.

Olivry et al. [18] presented a method to automatically derive sequential communication lower bounds for affine computations (including SYRK) which was extended to automatically derive algorithms that asymptotically achieve the lower bounds in [19]. Kwasniewski et al. [16] presented a general method to automatically derive communication lower bounds for Disjoint Array Access Programs, which they then applied to determine lower bounds for parallel Cholesky and LU factorizations.

The automated methods just mentioned require assumptions that prevent algorithms from taking advantage of symmetry of inputs or output. Beaumont et al. [7] improved the constants of communication lower bounds for both sequential SYRK and Cholesky factorization by a factor of $\sqrt{2}$ by taking advantage of symmetry. They also provided novel algorithms that take advantage of symmetry and reduce the I/O complexity of the previous best algorithms by the same factor and prove that the leading terms in their new lower bounds are tight. In separate work, Beaumont et al. [6] introduced a new tiling scheme for Cholesky factorization algorithms that takes advantage of symmetry to reduce communication.

3 PRELIMINARIES

3.1 SYRK

The SYRK computation is denoted as $\mathbf{C} = \mathbf{A} \cdot \mathbf{A}^T$, where $\mathbf{A}$ is an $n_1 \times n_2$ matrix and $\mathbf{C}$ is an $n_1 \times n_1$ symmetric matrix. Here $C_{ij} = \sum_{k=0}^{n_2-1} \mathbf{A}_{ik}\mathbf{A}_{jk}$ for $0 \leq j \leq i < n_1$. Figure 1 depicts the iteration space of SYRK along with projections of the data dependence. A scalar multiplication is performed in each iteration point.

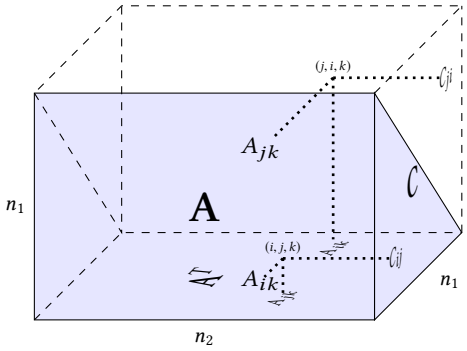

Figure 1: Iteration space of SYRK with a total of $n_1 n_2(n_1 + 1)/2$ iteration points. Iteration (i, j, k) and its projections are shown along with its symmetric partner (j, i, k) as described in Lemma 3.

3.2 Parallel Computation Model

We consider the MPI or $\alpha-\beta-\gamma$ model [11, 22] of parallel computation. In this model, a computation is distributed over P processors, each of which have their own local memory and are connected to all other processors via a fully connected network with bidirectional links. A processor may share data with another processor by sending (receiving) a message to (from) the processor over the network which we refer to as communication. A processor can only perform computations using data in its local memory, thus any input data for the computation that does not reside in the processor's memory must be received as a message from another processor. The cost of communication depends on two parameters α and β, where α is the per-message latency cost, and β is the per-word bandwidth cost. γ denotes the per-operation arithmetic cost. An algorithm's total communication is determined by both the size of the messages, the bandwidth cost, and the number of messages, the latency cost, required for each processor to have a copy of its required data in local memory. We assume that a processor can send or receive at most one message at a time, but multiple messages between disjoint pairs of processors may occur simultaneously as the fully connected network assures that there is no contention on the network. As bandwidth cost dominates overall communication when messages are large, we seek to minimize the bandwidth cost along the critical path which we will refer to as communication cost.

In this work, we assume that each processor has sufficient local memory to execute each algorithm. We discuss extensions of the lower bounds and algorithms to limited-memory regimes in §6.

Our algorithms in § 5 perform communication using All-to-All and Reduce-Scatter collectives. We assume that pairwise exchange algorithms are used for both collectives. These algorithms have optimal bandwidth costs. The latency and bandwidth costs of both collectives on P processors are $P-1$ and $\left(1-\frac{1}{P}\right)w$, respectively, where w is the number of words on each processor before and after All-to-All or before Reduce-Scatter [10, 11, 22]. Each processor also performs $\left(1-\frac{1}{P}\right)w$ computations for Reduce-Scatter. We discuss the use of latency-efficient collectives in §6.

3.3 Fundamental Existing Results

In this section we cover some existing definitions and results which we will use in §4 to derive our communication lower bounds.

The first result is geometric and allows us to relate the volume of a computation to the area of data required by it in the input and output arrays. The Loomis-Whitney inequality [17] has seen frequent use in the derivation of communication lower bounds for many linear algebra computations [2, 3, 12, 15, 21].

Lemma 1 (Loomis-Whitney [17]). *Let V be a finite set of points in $\mathbb{Z}^3$. Let $\phi_i(V)$ be the projection of V in the i-direction, i.e., all points (j,k) such that there exists an i so that $(i,j,k) \in V$. Define $\phi_j(V)$ and $\phi_k(V)$ similarly. Then*

$$|V| \leq |\phi_i(V)|^{1/2} \cdot |\phi_j(V)|^{1/2} \cdot |\phi_k(V)|^{1/2}.$$

The Loomis-Whitney inequality yields the intuition that to maximize volume given the sizes of the projections, the set V should be a cube, with projections that are squares. Applying this intuition to computations with cubical iteration spaces has led to tight lower bounds. In the case of rectangular matrix multiplication, the Loomis-Whitney inequality alone does not give a tight lower bound when the iteration space is not cubical, and in that scenario additional optimization is required [2]. In the case of SYRK, the iteration space is a triangular prism. While the Loomis-Whitney inequality applies in this situation, it does not yield tight constants, which leads to the gap between the automatically derived bounds of Olivry et al. [18] and the tight bounds of Beaumont et al. [7] for the sequential case. We present an extension of the Loomis-Whitney inequality in §4.1 allowing us to generate tight bounds for symmetric computations.

The following definitions and results enable us to solve the constrained optimization problem of Lemma 6 that is the crux of our lower bound argument. For completeness of the exposition, we follow Al Daas et al. [2] and begin by reminding the reader of the definitions of differentiable convex and quasiconvex functions, and the KKT conditions. We also state the lemma on the sufficiency of KKT-conditions for optimality under certain conditions.

Definition 1 ([9, eq. (3.2)]). *A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is* convex *if its domain is a convex set and for all $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{dom}\ f$,*

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle.$$

Definition 2 ([9, eq. (3.20)]). *A differentiable function $g : \mathbb{R}^d \to \mathbb{R}$ is* quasiconvex *if its domain is a convex set and for all $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{dom}\ g$,*

$$g(\boldsymbol{y}) \leq g(\boldsymbol{x}) \text{ implies that } \langle \nabla g(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \leq 0.$$

Definition 3 ([9, eq. (5.49)]). *Consider an optimization problem of the form*

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ and $\boldsymbol{g} : \mathbb{R}^d \to \mathbb{R}^c$ are both differentiable. Define the dual variables $\boldsymbol{\mu} \in \mathbb{R}^c$, and let $\boldsymbol{J_g}$ be the Jacobian of $\boldsymbol{g}$. The Karush-Kuhn-Tucker (KKT) *conditions of $(\boldsymbol{x}, \boldsymbol{\mu})$ are as follows:*

- Primal feasibility*: $\boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}$;*
- Dual feasibility*: $\boldsymbol{\mu} \geq 0$;*
- Stationarity*: $\nabla f(\boldsymbol{x}) + \boldsymbol{\mu} \cdot \boldsymbol{J_g}(\boldsymbol{x}) = \boldsymbol{0}$;*
- Complementary slackness*: $\mu_i g_i(\boldsymbol{x}) = 0$ for all $i \in \{1, \ldots, c\}$.*

Lemma 2 ([2, Lemma 3]). *Consider an optimization problem of the form given in eq.* (1). *If f is a convex function and each g_i is a quasiconvex function, then the KKT conditions are sufficient for optimality.*

4 MAIN LOWER BOUND RESULT

In this section, we present the communication lower bound for the SYRK computation. We focus on computing the strict lower triangle of C, yielding a valid lower bound on the full computation. We begin by proving an extension of the Loomis-Whitney inequality that yields a tight constraint to relate the volume of computation to the amount of data accessed for SYRK. We demonstrate some additional constraints on the sizes of projections onto arrays, and use the constraints to formulate an abstract optimization problem which we solve using the KKT-conditions. Finally, we demonstrate that the abstract minimization problem corresponds to minimizing the communication for SYRK to prove the lower bound.

4.1 Fundamental New Results

We present an extension of the Loomis-Whitney inequality in the following lemma that we use in the proof of Theorem 1 to obtain a tight constraint. The iteration space of SYRK forms a triangular prism, as shown in Fig. 1. The idea of the proof of the following lemma is, given a subset of iteration points corresponding to SYRK multiplications, to copy the points across the plane corresponding to the diagonal of C, and then relate the volume of the points (along with their symmetric partners) to the areas of the projections in the 3 directions using the Loomis-Whitney inequality. The symmetric partner of a single point is illustrated in Fig. 1.

Lemma 3. *Let V be a finite set of points contained in $\{(i,j,k) \in \mathbb{Z}^3 \mid j < i\}$. Let $\phi_i(V)$ be the projection of V in the i-direction, $\phi_i(V) = \{(j,k) \in \mathbb{Z}^2 \mid$ there exists $i \in \mathbb{Z}$ such that $(i,j,k) \in V\}$. Define $\phi_j(V)$ and $\phi_k(V)$ similarly. Then*

$$2|V| \leq |\phi_i(V) \cup \phi_j(V)| \cdot (2|\phi_k(V)|)^{1/2}.$$

Proof. Consider the set

$$\tilde{V} = \{(i,j,k) \in \mathbb{Z}^3 \mid (i,j,k) \in V \text{ or } (j,i,k) \in V\}$$

and its projections. If (i,j,k) is an element of $\tilde{V}$ then either (i,j,k) is an element of V or (j,i,k) is an element of V, and so $|\tilde{V}| \leq 2|V|$. If (i,j,k) is an element of V, then (j,i,k) can not be an element of V, but both (i,j,k) and (j,i,k) are elements of $\tilde{V}$ and so $|\tilde{V}| \geq 2|V|$. Thus $|\tilde{V}| = 2|V|$. Now, we will show that $\phi_i(\tilde{V}) = \phi_j(\tilde{V}) = \phi_i(V) \cup \phi_j(V)$. If $(j,k) \in \phi_i(\tilde{V})$, then there exists an i such that $(i,j,k) \in V$ or $(j,i,k) \in V$. In the first case $(j,k) \in \phi_i(V)$, in the second case $(j,k) \in \phi_j(V)$, and thus $\phi_i(\tilde{V}) \subseteq \phi_i(V) \cup \phi_j(V)$. If $(j,k) \in \phi_i(V) \cup \phi_j(V)$, then there exists an i such that $(i,j,k) \in V$ or $(j,i,k) \in V$ thus $(i,j,k) \in \tilde{V}$ and so $(j,k) \in \phi_i(\tilde{V})$. Hence $\phi_i(V) \cup \phi_j(V) \subseteq \phi_i(\tilde{V})$ and so $\phi_i(\tilde{V}) = \phi_i(V) \cup \phi_j(V)$. The same argument shows that $\phi_j(\tilde{V}) = \phi_i(V) \cup \phi_j(V)$. Finally, we show that $|\phi_k(\tilde{V})| = 2|\phi_k(V)|$. If $(i,j) \in \phi_k(V)$, then there exists a k such that $(i,j,k) \in V$. By the definition of V, $(i,j,k) \in V$ implies that $j < i$ and thus (j,i,k) can not be an element of V so (j,i) can not be an element in $\phi_k(V)$. However, $(i,j,k) \in V$ implies that $(i,j,k) \in \tilde{V}$ and $(j,i,k) \in \tilde{V}$ so for each element $(i,j) \in \phi_k(V)$, (i,j) and (j,i) are different elements in $\phi_k(\tilde{V})$. Thus $|\phi_k(\tilde{V})| \geq 2|\phi_k(V)|$. Suppose $(i,j) \in \phi_K(\tilde{V})$, then there exists a k such that $(i,j,k) \in \tilde{V}$ so either $(i,j,k) \in V$ and $(i,j) \in \phi_k(V)$ or $(j,i,k) \in V$ and $(j,i) \in \phi_k(V)$. Hence for each element $(i,j) \in \phi_k(\tilde{V})$, there exists an element in $\phi_k(V)$ that is either equal to (i,j) or has the two coordinates reversed. Thus $|\phi_k(\tilde{V})| \leq 2|\phi_k(V)|$ and hence $|\phi_k(\tilde{V})| = 2|\phi_k(V)|$. By the Loomis-Whitney inequality (Lemma 1), we know that

$$|\tilde{V}| \leq |\phi_i(\tilde{V})|^{1/2} |\phi_j(\tilde{V})|^{1/2} |\phi_k(\tilde{V})|^{1/2}.$$

Applying the results just shown and simplifying yields

$$2|V| \leq |\phi_i(V) \cup \phi_j(V)| (2|\phi_k(V)|)^{1/2}.$$

□

We also prove that the nonlinear function appearing in one of the constraints in our main optimization problem is quasiconvex.

Lemma 4. *The function $g_0(\boldsymbol{x}) = L - x_1^2 x_2$, for some constant L, is quasiconvex in the positive quadrant.*

Proof. Let $\boldsymbol{x}, \boldsymbol{y}$ be points in the positive quadrant with $g_0(\boldsymbol{y}) \leq g_0(\boldsymbol{x})$. Then $y_1^2 y_2 \geq x_1^2 x_2$. Applying the inequality of arithmetic and geometric means to the values y_1/x_1, y_2/x_2, we have

$$\frac{1}{3}\left(2\frac{y_1}{x_1} + \frac{y_2}{x_2}\right) \geq \left(\frac{y_1^2 y_2}{x_1^2 x_2}\right)^{1/3} \geq 1 \tag{2}$$

as $\frac{y_1}{x_1}$ and $\frac{y_2}{x_2}$ are positive. Then $\nabla g_0(\boldsymbol{x}) = \begin{bmatrix} -2x_1 x_2 & -x_1^2 \end{bmatrix}$, and

$$\begin{aligned} \langle \nabla g_0(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle &= -2x_1 x_2 y_1 + 2x_1^2 x_2 - x_1^2 y_2 + x_1^2 x_2 \\ &= 3x_1^2 x_2 \left(1 - \frac{1}{3}\left(2\frac{y_1}{x_1} + \frac{y_2}{x_2}\right)\right) \leq 0 \end{aligned}$$

where the last inequality follows from eq. (2). Thus by Def. 2, g_0 is quasiconvex in the positive quadrant. □

4.2 Lower Bounds on Individual Array Access

In the following result, we determine the minimum number of elements from each array that must be accessed by a processor performing the SYRK computation based on the number of arithmetic operations that this processor is performing. These lower bounds will provide additional constraints to our optimization problem which allows us to determine all applicable lower bounds with a unified argument.

Lemma 5. *Given a parallel algorithm that computes* SYRK *by multiplying an $n_1 \times n_2$ matrix* A *by its transpose using P processors, any processor that performs at least $1/P$th of the scalar multiplications associated with elements in the strict lower triangle of* C *must access at least $n_1 n_2/2P$ elements of* A *and also compute contributions to at least $n_1(n_1-1)/2P$ elements of the strict lower triangle of* C.

Proof. The total number of scalar multiplications that must be performed to compute the elements in the strict lower triangle of C is $n_1(n_1-1)n_2/2$. Consider a processor that computes at least $1/P$th of these computations. Each element of A is involved in $n_1 - 1$ scalar multiplications. If the processor accesses fewer than $n_1 n_2/2P$ elements of A, then it would perform fewer than $(n_1-1) \cdot n_1 n_2/2P$ scalar multiplications, which is a contradiction. Finally, each element of C is the sum of n_2 scalar multiplications. If

the processor computes contributions to fewer than $n_1(n_1-1)/2P$ elements of $\mathbf{C}$, then it would perform fewer than $n_2 \cdot n_1(n_1-1)/2P$ scalar multiplications, which is again a contradiction. □

4.3 Key Optimization Problem

The following lemma, which we state abstractly, is the key to our lower bound argument. In the abstract statement, x_1 corresponds to the number of elements of $\mathbf{A}$ that the processor needs to access, and x_2 corresponds to the number of elements of $\mathbf{C}$ that the processor contributes to. The first constraint comes from Lemma 3, while the constraints on x_2 come from Lemma 5 and the maximum number of computed elements of $\mathbf{C}$. We do not use the constraint on x_1 that can be derived from Lemma 5 as it is not tight at any optimal solutions. Instead, we use only the constraint that it must be positive to simplify the optimization.

Lemma 6. *Consider the following optimization problem:*

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} x_1 + x_2$$

such that

$$\left(\frac{n_1(n_1-1)n_2}{\sqrt{2}P}\right)^2 \le x_1^2 x_2,$$

$$0 \le x_1,$$

$$\frac{n_1(n_1-1)}{2P} \le x_2 \le \frac{n_1(n_1-1)}{2},$$

where $P \ge 1$, and n_1, n_2 are positive integers. The optimal solution $\boldsymbol{x}^*$ depends on the relative values of the constraints, yielding three cases:

(1) if $n_1 \le n_2$ and $P \le \frac{n_2}{\sqrt{n_1(n_1-1)}}$, then $x_1^* = \frac{n_2\sqrt{n_1(n_1-1)}}{P}$, $x_2^* = \frac{n_1(n_1-1)}{2}$;

(2) if $n_1 > n_2$ and $P \le \frac{n_1(n_1-1)}{n_2^2}$, then $x_1^* = n_2\sqrt{\frac{n_1(n_1-1)}{P}}$, $x_2^* = \frac{n_1(n_1-1)}{2P}$;

(3) if $n_1 \le n_2$ and $P > \frac{n_2}{\sqrt{n_1(n_1-1)}}$, or if $n_1 > n_2$ and $P > \frac{n_1(n_1-1)}{n_2^2}$, then $x_1^* = \left(\frac{n_1(n_1-1)n_2}{P}\right)^{2/3}$, $x_2^* = \frac{1}{2}\left(\frac{n_1(n_1-1)n_2}{P}\right)^{2/3}$.

Proof. By Lemma 2, we can establish the optimality of the solution for each case by verifying that there exist dual variables such that the KKT conditions specified in Def. 3 are satisfied. This optimization problem fits the assumptions of Lemma 2 because the objective function and all but the first constraint are affine functions, which are convex and quasiconvex, and the first constraint is quasiconvex in the positive quadrant by Lemma 4.

To begin, we rewrite our objective function and constraints in standard notation that match Lemma 2. In particular, we rewrite the constraints as a vector where each term corresponds to a constraint written as a function which must take values less than or equal to 0. Let $f(\boldsymbol{x}) = x_1 + x_2$ and

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} \left(\frac{n_1(n_1-1)n_2}{\sqrt{2}P}\right)^2 - x_1^2 x_2 \\ -x_1 \\ \frac{n_1(n_1-1)}{2P} - x_2 \\ x_2 - \frac{n_1(n_1-1)}{2} \end{bmatrix}.$$

Thus the gradient of the objective function is $\nabla f(\boldsymbol{x}) = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and the Jacobian of the constraint function is

$$\boldsymbol{J_g}(\boldsymbol{x}) = \begin{bmatrix} -2x_1x_2 & -x_1^2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix}.$$

Case 1 ($n_1 \le n_2$ and $P \le \frac{n_2}{\sqrt{n_1(n_1-1)}}$). We let

$$\boldsymbol{x}^* = \begin{bmatrix} \frac{n_2\sqrt{n_1(n_1-1)}}{P} & \frac{n_1(n_1-1)}{2} \end{bmatrix}$$

and

$$\boldsymbol{\mu}^* = \begin{bmatrix} \frac{P}{(n_1(n_1-1))^{3/2}n_2} & 0 & 0 & \frac{n_2}{(n_1(n_1-1))^{1/2}P} - 1 \end{bmatrix}$$

and verify the KKT conditions. As x_2^* is set to the boundary of one of its constraints and $x_1^* \ge 0$, it is clear that primal feasibility holds. For dual feasibility, it is immediate that $\mu_1^* > 0$ and $P \le \frac{n_2}{\sqrt{n_1(n_1-1)}}$ implies $\mu_4^* \ge 0$. Stationarity may be directly verified. Complementary slackness is satisfied as the 1st and 4th constraints are tight for $\boldsymbol{x}^*$ and the dual variables μ_2^* and μ_3^* are zero.

Case 2 ($n_1 > n_2$ and $P \le \frac{n_1(n_1-1)}{n_2^2}$). We let

$$\boldsymbol{x}^* = \begin{bmatrix} n_2\sqrt{\frac{n_1(n_1-1)}{P}} & \frac{n_1(n_1-1)}{2P} \end{bmatrix}$$

and

$$\boldsymbol{\mu}^* = \begin{bmatrix} \frac{P^{3/2}}{(n_1(n_1-1))^{3/2}n_2} & 0 & 1 - n_2\sqrt{\frac{P}{n_1(n_1-1)}} & 0 \end{bmatrix}$$

and verify the KKT conditions. As before, $x_1^* > 0$, and x_2^* is set to the boundary of one of its constraints so it is clear that primal feasibility holds. To see that the dual feasibility condition holds note that $\mu_1^* > 0$ and $P \le n_1(n_1-1)/n_2^2$ implies $\mu_3^* \ge 0$. Stationarity may be directly verified. Complementary slackness is satisfied as the 1st and 3rd constraints are tight for $\boldsymbol{x}^*$ and the dual variables μ_2^* and μ_4^* are zero.

Case 3 (($n_1 \le n_2$ and $P > \frac{n_2}{\sqrt{n_1(n_1-1)}}$) or ($n_1 > n_2$ and $P > \frac{n_1(n_1-1)}{n_2^2}$)). We let

$$\boldsymbol{x}^* = \begin{bmatrix} \left(\frac{n_1(n_1-1)n_2}{P}\right)^{2/3} & \frac{1}{2}\left(\frac{n_1(n_1-1)n_2}{P}\right)^{2/3} \end{bmatrix}$$

and

$$\boldsymbol{\mu}^* = \begin{bmatrix} \left(\frac{P}{n_1(n_1-1)n_2)}\right)^{4/3} & 0 & 0 & 0 \end{bmatrix}$$

and verify the KKT conditions. To verify primal feasibility we note that the feasibility of x_2^* may be directly verified for both sets of conditions, and $x_1^* \ge 0$. Dual feasibility is immediate as $\mu_1^* > 0$. Stationarity may be directly verified. Complementary slackness is satisfied because the 1st constraint is tight for $\boldsymbol{x}^*$ and the dual variables μ_2^*, μ_3^* and μ_4^* are zero.

Note that the optimal solutions coincide at boundary points between cases so that the values are continuous as P varies. □

4.4 Communication Lower Bound

THEOREM 1. *Consider the* SYRK *computation,* $\mathbf{C} = \mathbf{A} \cdot \mathbf{A}^T$*, where* $\mathbf{A}$ *has dimensions* $n_1 \times n_2$*. Any parallel algorithm using* P *processors that begins with one copy of the input matrix* $\mathbf{A}$ *and ends with one copy of the strict lower triangle of* $\mathbf{C}$ *and load balances either the computation of the entries below the diagonal of* $\mathbf{C}$ *or the data required to compute the entries below the diagonal of* $\mathbf{C}$ *must communicate at least*

$$W - \frac{n_1(n_1-1)/2 + n_1 n_2}{P}$$

words of data where

$$W = \begin{cases} \frac{n_1 n_2}{P} + \frac{n_1(n_1-1)}{2} & \text{if} \quad n_1 \le n_2 \text{ and } P \le \frac{n_2}{\sqrt{n_1(n_1-1)}} \\ \frac{n_1 n_2}{\sqrt{P}} + \frac{n_1(n_1-1)}{2P} & \text{if} \quad n_2 < n_1 \text{ and } P \le \frac{n_1(n_1-1)}{n_2^2} \\ \frac{3}{2}\left(\frac{n_1(n_1-1)n_2}{P}\right)^{2/3} & \text{if} \quad \left(n_1 \le n_2 \text{ and } P > \frac{n_2}{\sqrt{n_1(n_1-1)}}\right) \\ & \quad \text{or } \left(n_2 < n_1 \text{ and } P > \frac{n_1(n_1-1)}{n_2^2}\right). \end{cases}$$

PROOF. To establish the lower bound, we focus on the multiplications performed by a single processor and the amount of data that processor requires to perform those multiplications. We restrict our focus to the multiplications required to compute the entries below the diagonal of $\mathbf{C}$. We first verify that there exists a processor that performs at least $1/P$th of the multiplications and starts and ends the computation with at most $1/P$th of the total data.

If the algorithm load balances the computation of the subdiagonal portion of $\mathbf{C}$, then every processor performs $n_1(n_1-1)n_2/2P$ multiplications and there exists at least one processor whose elements of $\mathbf{A}$ at the start of the computation plus entries below the diagonal of $\mathbf{C}$ at the end of the computation must be at most $(n_1(n_1-1)/2 + n_1 n_2)/P$. If no such processor existed, then the algorithm would need to start or end with more than one copy of either $\mathbf{A}$ or the entries below the diagonal of $\mathbf{C}$. If instead, the algorithm load balances the data required in the computation of the entries below the diagonal of $\mathbf{C}$, then every processor starts and ends the computation with a total $(n_1(n_1-1)/2 + n_1 n_2)/P$ words of data, and there must exist at least one processor that performs at least $n_1(n_1-1)n_2/2P$ multiplications. If no such processor existed, then the algorithm would not be able to perform all $n_1(n_1-1)n_2/2$ multiplications required to compute the entries below the diagonal of $\mathbf{C}$. In either case, there is at least one processor that starts and ends the computation with at most $(n_1(n_1-1)/2 + n_1 n_2)/P$ words of data and performs at least $n_1(n_1-1)n_2/2P$ multiplications.

We consider the amount of data that this processor needs to perform its multiplications. Let F be the set of indices (i, j, k) associated with the multiplications assigned to this processor. Each element of F corresponds to the multiplication of the elements $\mathbf{A}(i,k)$ by $\mathbf{A}^T(k,j)$ which contributes to $\mathbf{C}(i,j)$. Then for every element of F, $j < i$, and $|F| \ge n_1(n_1-1)n_2/2P$. We consider the projections of F and note that $\phi_j(F)$ and $\phi_i(F)$ give the indices of the elements of $\mathbf{A}$ accessed to perform the multiplications. As these sets may overlap, the unique indices of elements of $\mathbf{A}$ required by the multiplications in F are given by $\phi_i(F) \cup \phi_j(F)$. Similarly, $\phi_k(F)$ yields the indices of the elements of $\mathbf{C}$ contributed to by the multiplications indexed by elements of F. By Lemma 3 we have

$$|\phi_i(F) \cup \phi_j(F)|(2|\phi_k(F)|)^{1/2} \ge 2|F| \ge \frac{n_1(n_1-1)n_2}{P},$$

which we can simplify to

$$|\phi_i(F) \cup \phi_j(F)|^2 |\phi_k(F)| \ge \left(\frac{n_1(n_1-1)n_2}{\sqrt{2}P}\right)^2.$$

Clearly a projection onto an array cannot be larger than the array itself, thus we have an upper bound constraint on the size of projection. Additionally by Lemma 5 we have a lower bound constraint on $|\phi_k(F)|$: $\frac{n_1(n_1-1)}{2P} \le |\phi_k(F)| \le \frac{n_1(n_1-1)}{2}$.

Finally as the size of any projection or union of projections onto an array must be non-negative, we have the constraint $0 \le |\phi_i(F) \cup \phi_j(F)|$. Thus, we want to minimize $|\phi_i(F) \cup \phi_j(F)| + |\phi_k(F)|$ subject to the constraints above and the result follows by Lemma 6. □

5 OPTIMAL PARALLEL ALGORITHMS

We present in this section three algorithms that attain the lower bounds of Theorem 1 in each of its three cases, matching the leading order constants exactly. We classify the algorithms as 1D, 2D, and 3D algorithms, following the terminology of Al Daas et al. [2]. If we consider the 3D iteration space of SYRK (a triangular prism), a 1D algorithm partitions the computation in only 1 of the dimensions, a 2D algorithm partitions the computation in exactly 2 dimensions, and a 3D algorithm partitions all 3 dimensions. Our 1D algorithm (§ 5.1) partitions only the n_2 dimension, and our 2D algorithm (§ 5.2) partitions the two n_1 dimensions. The 1D and 2D algorithms both utilize a 1D processor grid, though their indexing schemes differ, as the 2D algorithm uses triangle blocking (see § 5.2.1) for the distribution of $\mathbf{C}$. The 3D algorithm (§ 5.3) uses a 2D processor grid and also uses triangle blocking.

The choice of algorithm to attain optimality depends on the relative sizes of n_1, n_2, and P. First, consider the case $n_1 \le n_2$. This implies that $\mathbf{A}$ is short and wide and that $\mathbf{C}$ is a smaller matrix than $\mathbf{A}$. If P is relatively small, then the 1D algorithm is optimal. When P is sufficiently large, we switch to the 3D algorithm to maintain optimality. In the case that $n_1 > n_2$, $\mathbf{A}$ is tall and skinny and $\mathbf{C}$ is a bigger matrix than $\mathbf{A}$. For small P, we use the 2D algorithm. When P is sufficiently large, we again switch to the 3D algorithm to maintain optimality. In § 5.4, we detail the values of P for switching from 1D to 3D or from 2D to 3D and show that by choosing appropriate processor grid dimensions, we attain optimality in all cases.

Throughout this section, we use Π to denote the set of all processors, with $|\Pi| = P$. For the 2D processor grid used by the 3D algorithm, we will let $P = p_1 p_2$ and index processors by the pair $(k, \ell) \in \Pi$ with $0 \le k < p_1$ and $0 \le \ell < p_2$. In the case of the 1D algorithm, we consider $p_1 = 1$ and $p_2 = P$, and in that case processors will be indexed only by $0 \le \ell < P$. In the case of the 2D algorithm, we consider $p_1 = P$ and $p_2 = 1$, and in that case processors will be indexed only by $0 \le k < P$. When $p_1 > 1$, we will assume that $p_1 = c(c+1)$ for some prime number c. This assumption is a sufficient but not necessary condition for the triangle block distribution we use for the 2D and 3D algorithms.

5.1 1D Algorithm

In the case that $n_1 \leq n_2$ and P is not too large, a 1D algorithm is optimal. This algorithm, presented as Alg. 1, partitions the computation along only the dimension of length n_2. In this way, no communication of $\mathbf{A}$ is required, and the only communication will involve reducing contributions to the $\mathbf{C}$ matrix.

5.1.1 Data Distribution and Algorithm. The 1D algorithm requires that $\mathbf{A}$ is distributed in a 1D block column fashion so that each processor owns $1/P$th of the columns of $\mathbf{A}$. We use the index $\ell \in \Pi$ to specify a processor rank so that $0 \leq \ell < P$. Then assuming P divides n_2 evenly, $\mathbf{A}_\ell$ is a column block of dimension $n_1 \times (n_2/P)$. It proceeds by performing a local SYRK on each processor with its column block, producing a symmetric $n_1 \times n_1$ contribution to $\mathbf{C}$, followed by a reduction to compute the final output. Algorithm 1 uses a Reduce-Scatter collective in order to obtain an output that is evenly distributed across processors. The actual distribution can be arbitrary as long as it is even, so we leave it unspecified and denote the part of the result owned by processor ℓ at the end of the Reduce-Scatter as $\mathbf{C}^{(\ell)}$.

Algorithm 1 1D SYRK

Require: $\mathbf{A}$ is evenly column distributed across processor set Π with $\mathbf{A}_\ell$ the column block of $\mathbf{A}$ owned by processor $\ell \in \Pi$
Ensure: $\mathbf{C} = \mathbf{A}\mathbf{A}^T$ is evenly distributed across Π
1: **function** $\mathbf{C}$ =1D-SYRK($\mathbf{A}, \Pi$)
2: ℓ = MyRank(Π)
3: $\bar{\mathbf{C}}$ = Local-SYRK($\mathbf{A}_\ell$)
4: $\mathbf{C}^{(\ell)}$ = Reduce-Scatter($\bar{\mathbf{C}}, \Pi$)
5: **end function**

5.1.2 Cost Analysis. We recall that α, β and γ denote the latency cost, the per-word bandwidth cost and the per-operation arithmetic cost respectively. The computation is performed in line 3. The dimension of $\mathbf{A}_\ell$ is $n_1 \times (n_2/P)$, so the cost is dominated by $\gamma \cdot n_1^2 n_2/P$.

The communication is performed in line 4, and the number of entries of $\bar{\mathbf{C}}$ is $n_1(n_1+1)/2$. Thus, the communication cost of the Reduce-Scatter (see §3.2) is

$$\alpha \cdot (P-1) + \beta \cdot \frac{n_1(n_1+1)}{2}\frac{P-1}{P}, \tag{3}$$

and we ignore the cost of the computation of the Reduce-Scatter because it is dominated by that of the local SYRK.

5.2 2D Algorithm

In the case that $n_1 > n_2$ and P is not too large, a 2D algorithm is optimal. Our 2D algorithm, presented as Alg. 2, partitions the computation associated with both dimensions of length n_1. In this way, no communication of $\mathbf{C}$ is required, and the only communication will involve parts of $\mathbf{A}$. The 2D algorithm will assume a 1D processor grid, so that processors are indexed using $0 \leq k < P$. Note that we use a different index than in the 1D case to indicate a different partition of the computation. We will assume throughout this section that $P = c(c+1)$ for a prime c.

5.2.1 Triangle Block Distribution. The key to the optimality of our 2D algorithm is the observation that the lower triangle of a symmetric matrix can be partitioned into triangle blocks [7]. A triangle block is the strict lower triangle of a Cartesian product of a set of indices with itself. For example, given a set of indices $\{1,2,3\}$, the corresponding triangle block is the set of pairs $\{(2,1),(3,1),(3,2)\}$. We use this idea to distribute $\mathbf{C}$ as follows. For $P = c(c+1)$, we partition $\mathbf{C}$ into square blocks of dimension n_1/c^2, which results in $c^2(c^2-1)/2$ off-diagonal blocks and c^2 symmetric diagonal blocks. Next, as described in more detail below, we distribute blocks to processors so that each processor owns $c(c-1)/2$ off-diagonal blocks that form a triangle block of blocks. Finally, we distribute one symmetric diagonal block to each of c^2 processors (c processors own no diagonal block).

The advantage of a triangle block is that a processor can compute the output values of all the $\mathbf{C}$ blocks in its triangle block (a set with $O(c^2)$ blocks) using only the c row blocks of $\mathbf{A}$ with indices corresponding to the triangle block. However, partitioning the lower triangle of $\mathbf{C}$ blocks into triangle blocks is nontrivial. One simple condition that guarantees a valid partitioning is that c is a prime number, though other schemes are possible [7]. Once $\mathbf{C}$ is partitioned, we let processors follow an owner-compute rule, so that each processor performs all the computations associated with $\mathbf{C}$ blocks it owns, and processors must gather the elements of $\mathbf{A}$ they need to perform the computation (we define a conformal distribution of $\mathbf{A}$ below).

We use the notation $\mathbf{C}_{ij}$ with $0 \leq i,j < c^2$ to specify a block of $\mathbf{C}$ with dimension $(n_1/c^2) \times (n_1/c^2)$, and $\mathbf{A}_i$ denotes a $(n_1/c^2) \times n_2$ block of $\mathbf{A}$. To specify a Triangle Block Distribution, we define three types of sets: R_k is a c-element set of row block indices that define the triangle block of $\mathbf{C}$ blocks assigned to processor k; $D_k \subset R_k$ is a one-element (or empty) set containing the index of the diagonal block of $\mathbf{C}$ that is assigned to processor k; and Q_i is a $(c+1)$-element set of processor ranks k such that $i \in R_k$. We specify these sets for any prime c below. An example of Triangle Block Distribution is detailed in Fig. 2 and Tab. 1.

Our specification follows the "cyclic (c,k)-indexing family" [7] with $k = c$. The idea is to partition $\mathbf{C}$ hierarchically: we first divide $\mathbf{C}$ into $(n_1/c) \times (n_1/c)$ *zones*, so that each off-diagonal zone is a $c \times c$ grid of blocks. Each of the first c^2 processors are assigned exactly one block of each off-diagonal zone, and the set of blocks forms a triangle block of blocks. Each of the last c processors are assigned all of the off-diagonal blocks within a single diagonal zone. In this way, every processor is assigned exactly $c(c-1)/2$ off-diagonal blocks forming a triangle block of blocks.

We now define a helper function f_k that we use in the specifications of both R_k and D_k. Letting $0 \leq u < c$ and let $0 \leq k < P$, we define the function $f_k : \{0,\ldots,c-1\} \to \{0,\ldots,c^2-1\}$ as

$$f_k(u) = (\lfloor k/c \rfloor (u-1) + k) \bmod c + cu. \tag{4}$$

We can interpret $f_k(u)$ as the row index of the block assigned to processor k in the uth zone of the first zone column. Next we specify the set of row block indices that defines the triangle block for a particular processor k:

$$R_k = \begin{cases} \{\lfloor k/c \rfloor\} \cup \{f_k(u) : 0 < u < c\} & \text{if } 0 \leq k < c^2 \\ \{(k-c^2)c + u : 0 \leq u < c\} & \text{if } c^2 \leq k < c^2 + c. \end{cases} \tag{5}$$

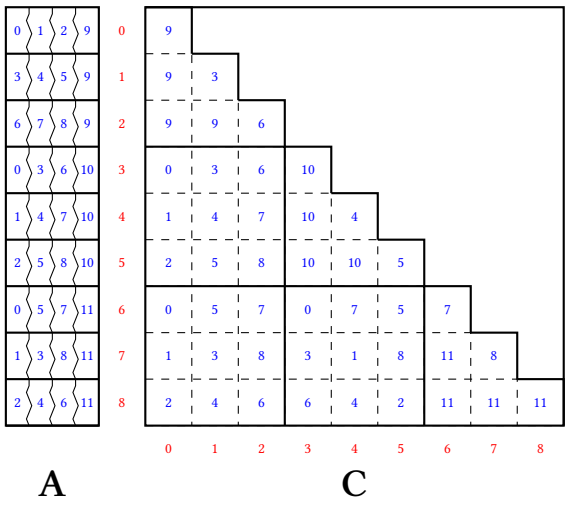

Figure 2: 2D Triangle Block Distribution of A and C with $c = 3$, $P = 12$. C is divided into $\frac{c(c-1)}{2}$ off-diagonal and c diagonal zones (the coarse partitioning). Each off-diagonal zone has $c \times c$ blocks. Processor ranks $0 \le k < P$ are shown in blue to indicate ownership of a block and block indices $0 \le i < c^2$ are shown in red. Distribution of each row block of A among its $c+1$ processors is arbitrary as long as it is even. For example, row block 6 of A is evenly distributed among processors $\{0, 5, 7, 11\}$.

For one of the first c^2 processors, all but the first index in R_k is given by f_k. For example, in Fig. 2 and Tab. 1, $R_3 = \{1, 3, 7\}$ and processor 3 is assigned blocks $\mathbf{C}_{31}$, $\mathbf{C}_{71}$, $\mathbf{C}_{73}$. For one of the last c processors, the set R_k is a contiguous set of indices. Note that the indices in R_k correspond to the row blocks of **A** required by processor k to perform the computation for its blocks of **C**. Finally, the index of the diagonal block owned by processor k is the set

$$D_k = \begin{cases} \{\} & \text{if } 0 \le k < c \\ \{\lfloor k/c \rfloor\} & \text{if } c \le k < c^2 \text{ and } k \equiv 0 \bmod c \\ \{f_k(\lfloor k/c \rfloor)\} & \text{if } c \le k < c^2 \text{ and } k \not\equiv 0 \bmod c \\ \left\{f_{c(k-c^2)}(k-c^2)\right\} & \text{if } c^2 \le k < c^2 + c. \end{cases} \quad (6)$$

We choose an assignment of diagonal blocks to ensure that $D_k \subset R_k$, and $|D_k| \le 1$, as this implies that no extra block of **A** is required by processor k and no processor has to compute more than one diagonal block (achieving reasonable load balance). The ith diagonal block of the $(0, 0)$ zone is assigned to processor rank ic except for $i = 0$ for which it is assigned to processor rank c^2. The $(i + uc)$th diagonal block with $u > 0$ (this is the ith diagonal block in the zone (u, u)) is assigned to the processor owning the block $(i + uc, u)$ with an exception if that processor rank is equal to uc. In such a case, it is assigned to processor rank $c^2 + u$. For example, in Fig. 2 and Tab. 1, $D_7 = \{6\}$, as the 6th diagonal block falls into the 2nd row zone and the processor of rank 7 owns the block $(6, 2)$.

Given R_k and D_k, we can specify the Triangle Block Distribution of **C**: processor k owns block $\mathbf{C}_{ij}$ if $i, j \in R_k$ and $i > j$, and it owns block $\mathbf{C}_{ii}$ if $i \in D_k$. Given a block $\mathbf{C}_{ij}$ with $i \ne j$, the indices i and j appear together in exactly one set R_k (this is because the triangle block scheme is "valid" [7]), and the block is owned by the corresponding processor.

k	R_k	D_k	i	Q_i
0	{0,3,6}	{}	0	{0,1,2,9}
1	{0,4,7}	{}	1	{3,4,5,9}
2	{0,5,8}	{}	2	{6,7,8,9}
3	{1,3,7}	{1}	3	{0,3,6,10}
4	{1,4,8}	{4}	4	{1,4,7,10}
5	{1,5,6}	{5}	5	{2,5,8,10}
6	{2,3,8}	{2}	6	{0,5,7,11}
7	{2,4,6}	{6}	7	{1,3,8,11}
8	{2,5,7}	{7}	8	{2,4,6,11}
9	{0,1,2}	{0}		
10	{3,4,5}	{3}		
11	{6,7,8}	{8}		

Table 1: Row block sets and processor sets of Triangle Block Distribution for $c = 3$, $P = 12$.

To specify the distribution of **A** that conforms to the Triangle Block Distribution of **C**, we also define a set for the reverse index: given a block index $0 \le i < c^2$, we want to specify the set of processors whose row block sets contain that index. We define the helper function h_i to define the sets Q_i. Let $0 \le q < c$ and $0 \le i < c^2$ and define the function $h_i : \{0, \ldots, c-1\} \to \{0, \ldots, c^2 - 1\}$ as

$$h_i(q) = (i - (\lfloor i/c \rfloor - 1)\, q) \bmod c + cq. \quad (7)$$

We can interpret $h_i(q)$ as specifying the processor assigned block $\mathbf{C}_{iq}$ (which falls in the first zone column).

Using h_i, we now specify Q_i:

$$Q_i = \begin{cases} \{ci + q : 0 \le q < c\} \cup \{c^2\} & \text{if } 0 \le i < c \\ \{h_i(q) : 0 \le q < c\} \cup \{c^2 + \lfloor i/c \rfloor\} & \text{if } c \le i < c^2. \end{cases} \quad (8)$$

Note that the first c blocks are members of a contiguous set of processor ranks along with processor c^2, and the remaining $c^2 - c$ blocks correspond to ranks specified by h_i and one of the last c processors, depending on which zone row the block falls in. For example, in Fig. 2 and Tab. 1, $Q_6 = \{0, 5, 7, 11\}$, as processors 0, 5, 7 own blocks in the 6th block row of the first zone column, and processor 11 owns blocks in the diagonal zone corresponding to the 6th block row.

Given Q_i, we can specify the conformal distribution of **A**. As all $c + 1$ processors with ranks in Q_i will need $\mathbf{A}_i$, we distribute $\mathbf{A}_i$ evenly among them. The specific distribution of $\mathbf{A}_i$ across the processors of Q_i is arbitrary, so we use the notation $\mathbf{A}_i^{(k)}$ to denote the part of $\mathbf{A}_i$ owned by rank $k \in Q_i$.

5.2.2 Algorithm. The 2D algorithm (Alg. 2) partitions the computation according to an owner-computes rule based on the distribution of **C**: if a processor owns block $\mathbf{C}_{ij}$, it computes all of the multiplications associated with the entries in that block. This computation occurs between line 15 and line 20 of the algorithm, where line 16 corresponds to computing a block within the processor's triangle block and line 19 corresponds to computing the diagonal block (if assigned). Note that while the processor grid is 1D, the Triangle Block Distribution of **C** partitions both rows and columns, so the algorithm is classified as 2D.

Algorithm 2 2D SYRK

Require: $|\Pi| = P = c(c+1)$ for prime c
Require: $\mathbf{A}$ is evenly subdivided into c^2 row blocks, and each row block $\mathbf{A}_i$ is evenly divided across a set of $c+1$ processors Q_i
Ensure: $\mathbf{C} = \mathbf{A}\mathbf{A}^T$ is Triangle Block distributed across Π
1: **function** $\mathbf{C}$ =2D-SYRK($\mathbf{A}, \Pi$)
2: k = MyRank(Π)
▷ *Gather c row blocks in row block set*
3: Allocate array $\mathbf{B}$ of P blocks, each of size $\frac{n_1 n_2}{c^2(c+1)}$
4: **for each** $i \in R_k$ **do**
5: **for each** $k' \in Q_i \backslash \{k\}$ **do**
6: $\mathbf{B}_{k'} = \mathbf{A}_i^{(k)}$
7: **end for**
8: **end for**
9: All-to-All($\mathbf{B}, \Pi$)
10: **for each** $i \in R_k$ **do**
11: **for each** $k' \in Q_i \backslash \{k\}$ **do**
12: Accumulate $\mathbf{B}_{k'}$ into $\mathbf{A}_i$
13: **end for**
14: **end for**
▷ *Compute $c(c-1)/2$ off-diagonal blocks*
15: **for each** $(i, j) \in R_k$ with $i > j$ **do**
16: $\mathbf{C}_{ij}$ = Local-GEMM($\mathbf{A}_i, \mathbf{A}_j{}^T$)
17: **end for**
▷ *Compute diagonal block if assigned*
18: **for each** $i \in D_k$ **do**
19: $\mathbf{C}_{ii}$ = Local-SYRK($\mathbf{A}_i$)
20: **end for**
21: **end function**

In order to perform the local computation, each processor k must gather the c row blocks corresponding to its row block set R_k. While the processor starts the computation owning $\mathbf{A}_i^{(k)}$ for each $i \in R_k$, it must communicate with the other processors in Q_i to gather the full block $\mathbf{A}_i$. Because the Triangle Block Distribution is valid, no two block row indices appear together in two R_k sets and no two processor ranks appear in two Q_i sets. Thus, each pair of processors need to exchange data corresponding to at most one row block of $\mathbf{A}$ (and a small subset of pairs of processors do not appear in any Q_i sets). Thus, the required communication pattern is an All-to-All collective, which is illustrated from line 3 to line 14. Most of the pseudocode is devoted to packing and unpacking a temporary data structure $\mathbf{B}$ to organize the data to be exchanged with each processor. Note that each local block $\mathbf{A}_i^{(k)}$ is copied c times into $\mathbf{B}$ to be shared with processors in Q_i.

5.2.3 Cost Analysis. Computation occurs on lines 16 and 19. The dimension of every row block $\mathbf{A}_i$ (and $\mathbf{A}_j$) is $n_1/c^2 \times n_2$. We have $|R_k| = c$ for every k and $|D_k| \in \{0, 1\}$. Thus, the computational cost of Alg. 2 is dominated by

$$\gamma \cdot \left[\frac{c(c-1)}{2} \cdot 2 \left(\frac{n_1}{c^2} \right)^2 n_2 + \left(\frac{n_1}{c^2} \right)^2 n_2 \right] = \gamma \cdot \left[\frac{n_1^2 n_2}{c^2} + O\left(\frac{n_1^2 n_2}{c^3} \right) \right].$$

As $P = c^2 + c$, which implies $c = \sqrt{P + 1/4} - 1/2$, we have that the leading order cost is

$$\gamma \cdot \frac{n_1^2 n_2}{P - c} = \gamma \cdot \left[\frac{n_1^2 n_2}{P} + O\left(\frac{n_1^2 n_2}{P^{3/2}} \right) \right]. \tag{9}$$

Note that the lack of perfect computational load balance is the result of $c \approx \sqrt{P}$ processors not computing a diagonal block of $\mathbf{C}$, but this imbalance does not affect the constant in the leading order term.

The communication is performed in line 9. The number of elements in $\mathbf{B}$ is $\frac{n_1 n_2}{c^2(c+1)} P = \frac{n_1 n_2}{c}$ and $|\Pi| = P$. Thus, the communication cost of All-to-All using a pairwise exchange algorithm is

$$\alpha \cdot (P - 1) + \beta \cdot \frac{n_1 n_2}{c} \left(1 - \frac{1}{P} \right). \tag{10}$$

As $c = \sqrt{P + 1/4} - 1/2$, the bandwidth cost of the algorithm is

$$\beta \cdot \frac{n_1 n_2}{\sqrt{P}} \left(\sqrt{1 + \frac{1}{4P}} + \frac{1}{2\sqrt{P}} \right) \left(1 - \frac{1}{P} \right). \tag{11}$$

5.3 3D Algorithm

In the case P is sufficiently large, we use a 3D algorithm regardless of the shape of $\mathbf{A}$. This algorithm, presented as Alg. 3, partitions the computation along all three dimensions but uses a 2D processor grid with dimensions $p_1 \times p_2$. As in the 2D algorithm (Alg. 2), the partitioning of both dimensions of length n_1 assumes a 1D processor grid indexed using $0 \le k < p_1$ and we assume $p_1 = c(c+1)$ for a prime c. The partitioning along the dimension of length n_2 is indexed using $0 \le \ell < p_2$. Because we partition all three dimensions, both $\mathbf{A}$ and $\mathbf{C}$ will require communication. The idea of the 3D algorithm is to perform the 2D algorithm on subsets of the computation using p_1 disjoint sets of processors and then sum the results by reducing across p_2 disjoint sets of processors.

5.3.1 Data Distribution and Algorithm. The 3D algorithm performs the 2D algorithm (based on a Triangle Block Distribution) across each of the p_2 slices of processors. Each slice performs SYRK using n_2/p_2 columns of the input matrix $\mathbf{A}$ so that the final result $\mathbf{C}$ is the sum of the results across the slices. Thus, the data distribution of each slice must match the requirements of the 2D algorithm.

Algorithm 3 3D SYRK

Require: $|\Pi| = p_1 p_2$, with $p_1 = c(c+1)$ for prime c
Require: $\mathbf{A}$ is evenly divided into a $c^2 \times p_2$ grid of blocks, and each block $\mathbf{A}_{i\ell}$ is evenly divided across a set of $c+1$ processors $Q_i \times \{\ell\}$
Ensure: $\mathbf{C} = \mathbf{A}\mathbf{A}^T$ is divided according to Triangle Block distribution across p_1 processors, and each triangle block of blocks $\mathbf{C}_k$ is evenly divided across a set of p_2 processors Π_{k*}
1: **function** $\mathbf{C}$ =3D-SYRK($\mathbf{A}, \Pi$)
2: (k, ℓ) = MyRank(Π)
3: $\overline{\mathbf{C}}$ = 2D-SYRK($\mathbf{A}_{*\ell}, \Pi_{*\ell}$)
4: Let $\overline{\mathbf{C}}_k$ be the local data of $\overline{\mathbf{C}}$
5: $\mathbf{C}_k^{(\ell)}$ = Reduce-Scatter($\overline{\mathbf{C}}_k, \Pi_{k*}$)
6: **end function**

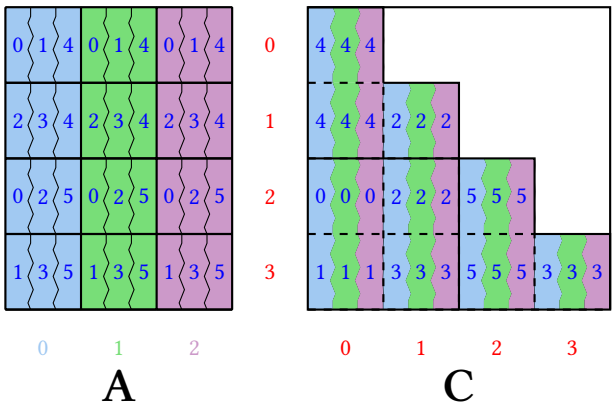

Figure 3: 3D Triangle Block Distribution of A and C with $p_1 = 6$ ($c = 2$), $p_2 = 3$. Processor ranks $0 \le k < p_1$ are shown in blue to indicate (shared) ownership of a block, and ranks $0 \le \ell < p_2$ are indicated by background colors. Row block indices $0 \le i < c^2$ of A are shown in red, and column block indices $0 \le \ell < p_2$ of A are shown in their corresponding colors. Distribution of each block of A among its $c+1$ processors and distribution of each block of C among its p_2 processors are arbitrary as long as they are even.

As show in Fig. 3, we partition $\mathbf{A}$ into a grid of $c^2 \times p_2$ blocks and use the notation $\mathbf{A}_{i\ell}$ to specify a block of $\mathbf{A}$ with dimensions $(n_1/c^2) \times (n_2/p_2)$. Consider processor slice ℓ, which we denote by $\Pi_{*\ell}$. This slice performs SYRK using input matrix $\mathbf{A}_{*\ell}$, which is the ℓth block column of $\mathbf{A}$, as given in line 3 of Alg. 3. In Fig. 3, each processor slice is assigned a unique color for its block column of $\mathbf{A}$. Thus, $\mathbf{A}_{*\ell}$ must be partitioned according to the 2D algorithm across the p_1 processors in $\Pi_{*\ell}$. The result of the SYRK performed by the ℓth slice will be partitioned across the p_1 processors according to the Triangle Block Distribution.

To compute the final result, we sum the results across slices. Because the intermediate results are identically distributed across each set of p_1 processors, we can sum along sets of p_2 processors with the same k index, denoted by Π_{k*}. We let $\overline{\mathbf{C}}$ denote the intermediate result, which is distributed over a processor slice, and we let $\overline{\mathbf{C}}_k$ denote the local data of $\overline{\mathbf{C}}$ owned by processor with rank k. Here $\overline{\mathbf{C}}_k$ consists of a triangle block of blocks, possibly along with a diagonal block. In Fig. 3, we label each block of $\mathbf{C}$ with an index, so that $\mathbf{C}_k$ is the set of blocks labeled with index k. In order to obtain an even distribution of the final output, we perform a Reduce-Scatter of the local $\overline{\mathbf{C}}_k$ across each Π_{k*} in line 5. We use the notation $\mathbf{C}_k^{(\ell)}$ to denote the part of $\mathbf{C}_k$ owned by processor (k, ℓ) at the end of the collective, noting that the distribution over Π_{k*} is arbitrary as long as it is even. This distribution of each $\mathbf{C}_k$ is indicated by the multicolor distribution of blocks of $\mathbf{C}$ in Fig. 3.

5.3.2 Cost Analysis. Computation occurs on lines 3 and 5. The 2D SYRK is performed on matrices of dimension $n_1 \times n_2/p_2$ using p_1 processors, thus the computation cost is the computation cost of the 2D algorithm given in eq. (9), with $P = p_1$ and n_2 divided by p_2: $\gamma \cdot \left[\frac{n_1^2 n_2}{P} + O\left(\frac{n_1^2 n_2}{P p_1^{1/2}}\right)\right]$. The computation cost of the Reduce-Scatter is a lower order term.

Communication of $\mathbf{A}$ occurs in line 3. As the 2D SYRK algorithm is performed on a matrix of size $n_1 \times n_2/p_2$ using p_1 processors, the communication cost is given by that of the 2D algorithm in eq. (10) with $P = p_1$ and n_2 divided by p_2: $\alpha \cdot (p_1 - 1) + \beta \cdot \frac{n_1 n_2}{c p_2}\left(1 - \frac{1}{p_1}\right)$.

Communication of $\mathbf{C}$ occurs in line 5. Each $\overline{\mathbf{C}}_k$ consists of a triangle block of blocks possibly along with a diagonal block for a maximum of $\frac{c(c-1)}{2} \cdot \frac{n_1^2}{c^4} + \frac{1}{2} \cdot \frac{n_1}{c^2}\left(\frac{n_1}{c^2} + 1\right)$ elements, and the collective is performed over p_2 processors. The communication cost of the collective is dominated by $\alpha \cdot (p_2 - 1) + \beta \cdot \frac{1}{2}\frac{n_1^2}{c^2}\left(1 - \frac{1}{p_2}\right)$.

Thus the total communication cost is given by

$$\alpha \cdot O(p_1 + p_2) + \beta \cdot \left(\frac{n_1 n_2}{c p_2}\left(1 - \frac{1}{p_1}\right) + \frac{1}{2}\frac{n_1^2}{c^2}\left(1 - \frac{1}{p_2}\right)\right)$$

As $c = \sqrt{p_1 + 1/4} - \frac{1}{2}$, this cost is approximately equal to

$$\alpha \cdot O\left(p_1 + p_2\right) + \beta \cdot \left(\frac{n_1 n_2}{\sqrt{p_1} p_2}\left(1 - \frac{1}{p_1}\right) + \frac{n_1^2}{2 p_1}\left(1 - \frac{1}{p_2}\right)\right).$$

Thus, the total communication costs (to leading order terms) of Alg. 3 are given by

$$\alpha \cdot O\left(p_1 + p_2\right) + \beta \cdot \left(\frac{n_1 n_2}{\sqrt{p_1} p_2} + \frac{n_1^2}{2 p_1}\right). \tag{12}$$

5.4 Optimal Processor Grid Selection

In order to minimize the communication costs of our algorithms, we propose to select p_1 and p_2 based on the terms of the communication lower bound (Theorem 1). For the sake of simplicity, we assume that p_1 and p_2 are integers. We also assume that the numerator is divisible by the denominator for each division expression. As the lower bound has three cases, we discuss p_1 and p_2 values for each case separately.

Case 1: $n_1 \le n_2$ and $P \le \frac{n_2}{\sqrt{n_1(n_1-1)}}$. We set $p_1 = 1$ and $p_2 = P$ and use the 1D Algorithm (Alg. 1) for this case. The bandwidth cost of the algorithm given in eq. (3) matches the leading term in the lower bound exactly.

For the remaining two cases, we also assume that there exists a prime integer c such that $c(c+1) = p_1$.

Case 2: $n_1 > n_2$ and $P \le \frac{n_1(n_1-1)}{n_2^2}$. We set $p_1 = P$ and $p_2 = 1$ and use the 2D Algorithm (Alg. 2). The bandwidth cost given in eq. (11) matches the leading term in the lower bound exactly.

Case 3: $n_1 \le n_2$ and $P > \frac{n_2}{\sqrt{n_1(n_1-1)}}$ or $n_1 > n_2$ and $P > \frac{n_1(n_1-1)}{n_2^2}$. We propose to use the 3D algorithm (Alg. 3) for this case with $p_1 = \left(\frac{n_1}{n_2}\right)^{2/3} P^{2/3}$ and $p_2 = \left(\frac{n_2}{n_1}\right)^{2/3} P^{1/3}$. The bandwidth cost of the algorithm (to leading order terms) is given by eq. (12), and with these values of p_1 and p_2, it simplifies to $(3/2)(n_1^2 n_2/P)^{2/3}$, matching the lower bound exactly.

6 CONCLUSION

Theorem 1 in § 4 establishes parallel memory-independent communication lower bounds for parallel SYRK. The lower bound on accessed data is cast as the solution of a nonlinear constrained optimization problem. One of the constraints is derived by manipulating the iteration space so that the Loomis-Whitney theorem gives a constraint suitable for the non-cubic iteration space of SYRK. The analytic solution depends on the relative sizes of the matrix and number of processors giving rise to three cases. We present Algorithms 1 to 3 in § 5 that each achieve a communication cost with leading term matching one of the cases of the lower bound.

In the sequential case, Beaumont et al. [7] demonstrate that SYRK has a higher operational intensity than GEMM, and their I/O-optimal algorithm reduces the memory traffic by a factor of $2^{3/2}$ compared to GEMM. Our algorithms are the first parallel SYRK algorithms that reduce both the computational complexity and bandwidth cost compared to optimal parallel GEMM as presented by Al Daas et al. [2]. In the parallel case, both the computation and communication are reduced by a factor of 2, so the corresponding ratio of computation to communication matches GEMM.

Our computational model assumes sufficient memory, but the 3D algorithm may not be feasible in limited-memory scenarios. In this case, an extension of the memory-dependent sequential bound to the parallel case gives a tighter lower bound. We plan to explore algorithms that attain the memory-dependent lower bound in future work.

In the communication cost analysis, we use bandwidth-optimal algorithms for All-to-All and Reduce-Scatter that have non-optimal latency costs. For Reduce-Scatter, if we consider an adaptation of Bruck's algorithm for concatenation (All-Gather) [10], we can obtain both bandwidth and latency optimality for all processor counts including non-powers-of-two. For All-to-All, if we consider a latency-optimal butterfly algorithm, we can reduce the latency cost to $O(\log P)$ at the expense of a higher bandwidth cost (by a factor of $O(\log P)$). Because we cast our communication in terms of All-to-All, we cannot attain both bandwidth and latency optimality simultaneously, but it is an open question whether a parallel SYRK algorithm exists that attains the optimal leading order bandwidth cost constant and optimal $O(\log P)$ latency.

We believe that our approach of manipulating the iteration space to exploit the symmetry in SYRK can be used to obtain tight lower bounds for other important linear algebra kernels involving a symmetric matrix, such as the symmetric rank-$2k$ update (SYR2K), symmetric matrix multiplication (SYMM), and sparse versions of these kernels such as symmetric sparse matrix times dense matrix and symmetric sampled dense-dense matrix multiplication.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under Grant No. CCF-1942892 and OAC-2106920. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research program under Award Number DE-SC-0023296. This project has also received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant agreement No. 810367).

REFERENCES

[1] A. Aggarwal, A. K. Chandra, and M. Snir. 1990. Communication Complexity of PRAMs. *Theor. Comp. Sci.* 71, 1 (1990). https://doi.org/10.1016/0304-3975(90)90188-N

[2] H. Al Daas, G. Ballard, L. Grigori, S. Kumar, and K. Rouse. 2022. Tight Memory-Independent Parallel Matrix Multiplication Communication Lower Bounds. In *SPAA 2022*. https://doi.org/10.1145/3490148.3538552

[3] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. 2014. Communication Lower Bounds and Optimal Algorithms for Numerical Linear Algebra. *Acta Numerica* 23 (2014). https://doi.org/10.1017/S0962492914000038

[4] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. 2012. Strong Scaling of Matrix Multiplication Algorithms and Memory-Independent Communication Lower Bounds. In *SPAA 2012*. https://doi.org/10.1145/2312005.2312021

[5] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. 2011. Minimizing Communication in Numerical Linear Algebra. *SIAM J. Matrix Anal. Appl.* 32, 3 (2011). https://doi.org/10.1137/090769156

[6] O. Beaumont, P. Duchon, L. Eyraud-Dubois, J. Langou, and M. Vérité. 2022. Symmetric Block-Cyclic Distribution: Fewer Communications Leads to Faster Dense Cholesky Factorization. In *SC 2022*. https://dl.acm.org/doi/abs/10.5555/3571885.3571923

[7] O. Beaumont, L. Eyraud-Dubois, J. Langou, and M. Vérité. 2022. I/O-optimal Algorithms for Symmetric Linear Algebra Kernels. In *SPAA 2022*. https://doi.org/10.1145/3490148.3538587

[8] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. 1997. *ScaLAPACK Users' Guide*. SIAM. Also available from http://www.netlib.org/scalapack/.

[9] S. Boyd and L. Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press. https://web.stanford.edu/~boyd/cvxbook/

[10] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby. 1997. Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems. *IEEE Trans. on Par. and Dist. Sys.* 8, 11 (1997). https://doi.org/10.1109/71.642949

[11] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn. 2007. Collective Communication: Theory, Practice, and Experience. *Conc. and Comp.: Prac. and Exper.* 19, 13 (2007). https://doi.org/10.1002/cpe.1206

[12] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger. 2013. Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication. In *IPDPS 2013*. https://doi.org/10.1109/IPDPS.2013.80

[13] J. Dongarra, J.-F. Pineau, Y. Robert, Z. Shi, and F. Vivien. 2008. Revisiting Matrix Product on Master-Worker Platforms. *Intl. J. Found. of Comp. Sci.* 19, 06 (2008). https://doi.org/10.1142/S0129054108006303

[14] J. W. Hong and H. T. Kung. 1981. I/O complexity: The Red-Blue Pebble Game. In *STOC 1981*. https://doi.org/10.1145/800076.802486

[15] D. Irony, S. Toledo, and A. Tiskin. 2004. Communication Lower Bounds for Distributed-Memory Matrix Multiplication. *J. Par. and Dist. Comp.* 64, 9 (2004). https://doi.org/10.1016/j.jpdc.2004.03.021

[16] G. Kwasniewski, M. Kabic, T. Ben-Nun, A. N. Ziogas, J. E. Saethre, A. Gaillard, T. Schneider, M. Besta, A. Kozhevnikov, J. VandeVondele, and T. Hoefler. 2021. On the Parallel I/O Optimality of Linear Algebra Kernels: Near-Optimal Matrix Factorizations. In *SC 2021*. https://doi.org/10.1145/3458817.3476167

[17] L. H. Loomis and H. Whitney. 1949. An Inequality Related to the Isoperimetric Inequality. *Bull. Amer. Math. Soc.* 55, 10 (1949). https://doi.org/10.1090/S0002-9904-1949-09320-5

[18] A. Olivry, J. Langou, L.-N. Pouchet, P. Sadayappan, and F. Rastello. 2020. Automated Derivation of Parametric Data Movement Lower Bounds for Affine Programs. In *PLDI 2020*. https://doi.org/10.1145/3385412.3385989

[19] G. Olivry, A. Ioos, N. Tollenaere, A. Rountev, P. Sadayappan, and F. Rastello. 2021. IOOpt: Automatic Derivation of I/O Complexity Bounds for Affine Programs. In *PLDI 2021*. https://doi.org/10.1145/3453483

[20] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero. 2013. Elemental: A New Framework for Distributed Memory Dense Matrix Computations. *ACM Trans. on Math. Soft.* 39, 2, Article 13 (2013). https://doi.org/10.1145/2427023.2427030

[21] T. M. Smith, B. Lowery, J. Langou, and R. A. van de Geijn. 2019. *A Tight I/O Lower Bound for Matrix Multiplication*. Technical Report. arXiv. https://doi.org/10.48550/arXiv.1702.02017

[22] R. Thakur, R. Rabenseifner, and W. Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Intl. J. High Perf. Comp. App.* 19, 1 (2005). https://doi.org/10.1177/1094342005051521