I/O-Optimal Algorithms for Symmetric Linear Algebra Kernels

Olivier Beaumont, Lionel Eyraud-Dubois, Mathieu Vérité, Julien Langou

I/O-Optimal Algorithms for Symmetric Linear Algebra Kernels

Olivier Beaumont
Lionel Eyraud-Dubois
Mathieu Vérité
olivier.beaumont@inria.fr
lionel.eyraud-dubois@inria.fr
mathieu.verite@inria.fr
Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800
Talence, France

Julien Langou
University of Colorado Denver
Denver, Colorado, USA
julien.langou@ucdenver.edu

ABSTRACT

In this paper, we consider two fundamental symmetric kernels in linear algebra: the Cholesky factorization and the symmetric rank-k update (SYRK), with the classical three nested loops algorithms for these kernels. In addition, we consider a machine model with a fast memory of size S and an unbounded slow memory. In this model, all computations must be performed on operands in fast memory, and the goal is to minimize the amount of communication between slow and fast memories. As the set of computations is fixed by the choice of the algorithm, only the ordering of the computations (the schedule) directly influences the volume of communications.

We prove lower bounds of $\frac{1}{3\sqrt{2}}\frac{N^3}{\sqrt{S}}$ for the communication volume of the Cholesky factorization of an $N \times N$ symmetric positive definite matrix, and of $\frac{1}{\sqrt{2}}\frac{N^2M}{\sqrt{S}}$ for the SYRK computation of $\mathbf{A}\cdot\mathbf{A}^\mathsf{T}$, where $\mathbf{A}$ is an $N \times M$ matrix. Both bounds improve the best known lower bounds from the literature by a factor $\sqrt{2}$.

In addition, we present two out-of-core, sequential algorithms with matching communication volume: **TBS** for SYRK, with a volume of $\frac{1}{\sqrt{2}}\frac{N^2M}{\sqrt{S}} + O(NM\log N)$, and **LBC** for Cholesky, with a volume of $\frac{1}{3\sqrt{2}}\frac{N^3}{\sqrt{S}} + O(N^{5/2})$. Both algorithms improve over the best known algorithms from the literature by a factor $\sqrt{2}$, and prove that the leading terms in our lower bounds cannot be improved further. This work shows that the operational intensity of symmetric kernels like SYRK or Cholesky is intrinsically higher (by a factor $\sqrt{2}$) than that of corresponding non-symmetric kernels (GEMM and LU factorization).

CCS CONCEPTS

• **Theory of computation** → *Shared memory algorithms*; **Communication complexity**; • **Mathematics of computing** → *Solvers.*

KEYWORDS

communication-avoiding algorithms, linear algebra, symmetric kernels, syrk, cholesky

SPAA'22, July 11–14, 2022, Philadephia, PA

ACM Reference Format:
Olivier Beaumont, Lionel Eyraud-Dubois, Mathieu Vérité, and Julien Langou. 2022. I/O-Optimal Algorithms for Symmetric Linear Algebra Kernels. In *Proceedings of SPAA '22: ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'22).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnn

1 INTRODUCTION

Dense matrix factorizations play a significant role in scientific computing. In particular, symmetric positive definite matrices appear in many applications; the Cholesky factorization is a dedicated factorization algorithm for such matrices. With the increase of both the scale of the platforms and the problems to solve, minimizing communications for scientific computing, and in particular for these factorization kernels, is crucial to reach the peak performance from modern hardware. In addition to its effect on execution time, the volume of data movement has a major impact on energy consumption during a computation, so that reducing the volume of data movement will result in a reduction of the energy required by a computation.

To evaluate data movements, we consider in this paper an elementary model from the literature, with a computation unit associated to a fast memory of size S where the operands of computations must be located, and a slow memory. In this context, the objective is to perform a set of computations, given by the algorithm, while minimizing the data movements between the fast memory and the slow memory. Different orderings (schedules) for a given set of computations (algorithm) may induce different volume of communication between slow and fast memories. We are interested both in establishing the minimum volume of communication needed for a given computation and in finding optimal schedules.

The *operational intensity* (OI), defined as the ratio of the number of arithmetic operations to the volume of data movement to/from memory, is a critical metric for comparing the efficiency of various algorithms and their schedules. Since the number of operations for our computation is known and fixed, studying the operational intensity is similar to studying the volume of data movement. We want to increase OI, we do so by reducing the volume of communication.

In this paper, we study the OI of the Cholesky factorization and of the symmetric rank-k update (SYRK): we derive lower bounds of the volume of data movement that any schedule has to perform for each of these algorithms. Such bounds are valid both for parallel algorithms and out-of-core sequential algorithms, and we provide

out-of-core sequential algorithms whose communication complexities asymptotically match our lower bounds.

It is known [2] that performing a Cholesky factorization of an $N \times N$ matrix with a memory of size S requires $\Omega\left(\frac{N^3}{\sqrt{S}}\right)$ data transfers. Regarding the optimal constant, on the one hand, recent work on automated lower bound derivation [10] show that the constant is at least $\frac{1}{6}$; and, on the other hand, an algorithm by Béreux [4] proves that the constant is at most $\frac{1}{3}$. This represents a factor of 2 between the largest known lower bound and the smallest known upper bound. In this paper, we close this gap by increasing the lower bound by a factor $\sqrt{2}$ and decreasing the upper bound by the same factor therefore reaching optimality.

This also shows that the algorithm of Béreux [4] is not optimal which is a surprising result: a rule of thumb, until this present paper, has been that the minimum volume of data transfer, in dense linear operations, is "number of operations divided by $\sqrt{S}$". For the Cholesky factorization, this would give the constant to be $\frac{1}{3}$ and make Béreux's algorithm [4] optimal.

Similar results exist for the SYRK kernel, which computes the symmetric matrix $\mathbf{C} = \mathbf{A} \cdot \mathbf{A}^\mathsf{T}$ where $\mathbf{A}$ is a $N \times M$ matrix. With a memory of size S, $\Theta\left(\frac{N^2M}{\sqrt{S}}\right)$ transfers are required. The **OOC_SYRK** algorithm by Béreux [4] achieves a constant of 1, and a recent work [10] provides a lower bound with a constant of $\frac{1}{2}$. Again, it was believed that the optimal value was 1. Again, in this paper, we prove that the best known lower bound [10] can be increased by a factor of $\sqrt{2}$ and the I/O volume of the best known algorithm [4] decreased by the same factor, hence reaching optimality.

Previous works on the SYRK kernel take the symmetry of the operation $\mathbf{C} = \mathbf{A} \cdot \mathbf{A}^\mathsf{T}$ into account by computing the lower half of the matrix $\mathbf{C}$. However the fact that the matrix $\mathbf{A}$ appears twice on the right-hand side has not been exploited: if $A_{j,i}$ is required while $A_{i,j}$ is in the fast memory, the schedule loads $A_{j,i}$ anyway. Indeed it is akin to computing the lower half of $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$, where $\mathbf{A}$ is a $N \times M$ matrix and $\mathbf{B}$ is a $M \times N$ matrix. Similarly, a lower bound for Cholesky is derived in [8] under the constraint that it is forbidden to use $A_{i,j}$ when $A_{j,i}$ is available. The lower bound obtained with such a constraint is potentially too large: there may exist algorithms which perform fewer data transfers by making a better use of symmetry.

In this paper, we precisely show how to exploit the symmetry of input to evaluate the volume of I/O actually required to perform SYRK and Cholesky and derive lower bounds (thus potentially lower than bounds that do not take it into account). We also present algorithms which make explicit use of the symmetry to reduce data movement.

After a more detailed presentation of the related works in Section 2 and a presentation of the methodology in Section 3, we thus present our 4 contributions:

- an improvement by a factor $\sqrt{2}$ of the best known lower bound for the communication requirements of the SYRK kernel (from $\frac{1}{2}$ to $\frac{1}{\sqrt{2}}$, Section 4.1);
- an application of this result to the Cholesky factorization, improving over the best known lower bound by a factor $\sqrt{2}$ (from $\frac{1}{6}$ to $\frac{1}{3\sqrt{2}}$, Section 4.2);
- a **TBS** algorithm for the SYRK kernel which makes use of the symmetry of the computations to reduce the communication volume by a factor $\sqrt{2}$ over previous approaches (from 1 to $\frac{1}{\sqrt{2}}$, Section 5.1);
- a **LBC** algorithm for the Cholesky factorization, which uses **TBS** and a large-block, right-looking approach to obtain a communication-optimal schedule (Section 5.2), providing a $\sqrt{2}$ improvement over the best known algorithm (from $\frac{1}{3}$ to $\frac{1}{3\sqrt{2}}$).

Our results provide a proof that the maximal operational intensity for the multiplication operations in SYRK and Cholesky is $\sqrt{\frac{S}{2}}$, or equivalently $\sqrt{2S}$ when also counting the addition operations. This is to be compared to the equivalent results for matrix-matrix multiplication (GEMM) and LU factorization, for which the maximal operational intensity is $\sqrt{S}$ (see Table 1 in [10]). Our work shows that symmetric operations have intrinsically higher operational intensity, and our algorithms provide insight about how to take advantage of that. At this stage, we do not claim that our algorithms can be used in a practical setting, but their existence shows that the lower bounds that we obtain are the best possible.

2 RELATED WORKS

Research work regarding the estimation of data transfers required to perform linear algebra kernels heavily rely on two simplified machine models:

(1) Two-levels memory model: the machine features one fast and limited memory of size S, one "slow" and unlimited memory. Input required for any computation must reside in fast memory to be performed, while the data initially resides in slow memory.
(2) Parallel model: P nodes, each with a memory of size S, can communicate through a network.

Both models are highly related: the two-level model can be used to study the volume of communication of a single node in a parallel machine, since the set of all other nodes can be viewed as a single "slow" memory with which data transfers occur. Thus, most lower bounds are actually obtained in the two-level model, and then transferred to the parallel model. For the purpose of comparison, only bandwidth communication cost estimations are of interest to us: we do not consider latency issues or bound the number of messages, we focus on the number of data elements transferred.

2.1 Two-level sequential model

The paper from Hong and Kung [7] can be considered as the founding piece of subsequent development on the topic. In this work, the authors consider a two-level memory machine. A set of rules referred as the *pebbling game* models the required data transfers between the two types of memory. Based on the analysis of the *computational DAG*, the authors derive asymptotic lower bounds for the number of data transfers required between the two levels of memory. For classical $N \times N$ matrix multiplication (*i.e.* requiring $O(N^3)$ operation), Hong and Kung prove that $\Omega\left(\frac{N^3}{\sqrt{S}}\right)$ data transfers are required.

Irony *et al.* improve this result in [6] through the expression of a "memory-communication tradeoff" in the context of two-levels memory model. For $N \times N$ matrix multiplication, the total number of data transfers between slow and fast memory actually is $\Theta(\frac{N^3}{\sqrt{S}})$. This however assumes a limited fast memory size: $S \leqslant \frac{N^2}{\sqrt[3]{32}}$.

In [2], Ballard *et al.* extend Hong and Kung's result to Cholesky factorization using a reduction technique: by observing that a $\frac{N}{3} \times \frac{N}{3}$ matrix multiplication can be carried out through a $N \times N$ Cholesky factorization, they prove that the communication costs of the latter method are only a constant factor of the former. Therefore, the asymptotic bounds established in [7] hold: performing a $N \times N$ Cholesky factorization requires at least $\Omega(\frac{N^3}{\sqrt{S}})$ data transfers. This result has been later generalized to a broader variety of kernels [1, 3]

This line of work enables to establish very general bounds, for a broad range of kernels, including sparse computations, and provides algorithms with matching communication complexity. However, these algorithms are only asymptotically optimal: they achieve a communication volume $O(\frac{N^3}{\sqrt{S}})$ for Cholesky for example, but the respective hidden constant factors of the lower bounds and the algorithms can be significantly different.

Very recently, automatic *cDAG* analysis techniques have led to refinement of lower bounds for several kernels at once, meaning that the constant factor of the dominant term is explicitly provided. In particular, Olivry *et al.* [9, 10] derive lower bounds on data transfers (in the context of the out-of-core model) for any kernel expressed as an affine program. Among other results, they establish that Cholesky factorization requires at least $\frac{N^3}{6\sqrt{S}} + O(N^2)$ I/O operations, and that SYRK requires at least $\frac{1}{2}\frac{N^2M}{\sqrt{S}} + O(NM)$ I/O operations. This work also presents a tool which computes an efficient tiling scheme according to the available memory size. The tool is however limited to rectangular tilings.

Independently, using explicit enumeration of data reuse, Kwasniewski *et al.* [8] obtain a corresponding lower bound for LU factorization: their proof is in a parallel context, but their arguments show that the minimum number of data transfers is lower bounded by $\frac{2}{3}\frac{N^3}{\sqrt{S}}$. They also propose a generalization to Cholesky factorization and obtain an improved $\frac{1}{3}\frac{N^3}{\sqrt{S}}$ lower bound, which however makes the implicit assumption that there is no data reuse related to the symmetry of the matrix as discussed in Section 1.

In 2009, Béreux [4] proposes a sequential out-of-core Cholesky algorithm with "narrow blocks" that performs $\frac{N^3}{3\sqrt{S}} + O(N^2)$ I/O operations, without making use of the symmetry of the matrix. This matches the lower bound from Kwasniewski *et al.*, showing that this is the best possible bound in the context of this implicit assumption. The same paper also mentions an out-of-core SYRK algorithm, based on similar ideas, which performs $\frac{MN^2}{\sqrt{S}} + O(MN)$ I/O operations.

2.2 Parallel model

In [6] Irony *et al.* apply their "memory-communication tradeoff" for matrix multiplication in the context of parallel execution. It states that using P nodes to perform the multiplication of $M \times N$ and $N \times R$ matrices, at least one node must send or receive at least $\frac{MNR}{2\sqrt{2}P\sqrt{S}} - S$ data. The authors also present $2D$ and $3D$ task distributions for matrix multiplication as parallel implementation in the two limit cases for memory size: $S = O\left(\frac{N^2}{P}\right)$ for the first case, $S = O\left(\frac{N^2}{P^{\frac{2}{3}}}\right)$ for the second. They prove that those algorithms are asymptotically optimal regarding communications since they match the lower bound derived from the "memory-communication tradeoff". More recent work by Solomonik *et al.* [11] presents an original way of modeling dependencies of any *cDAG* as a lattice-hypergraph which enables the authors to extend the "memory-communication tradeoff" to take into account synchronization and express bounds about the communication on the critical path.

In 2011, Solomonik and Demmel [12] introduced the $2.5D$ algorithms for matrix multiplication and LU factorization, bringing a continuum between $2D$ and $3D$ algorithms.Experimental results show the superiority of $2.5D$ algorithms over conventional $2D$ algorithms.

Regarding Cholesky, Ballard *et al.* [2] reviewed existing parallel distributed algorithms and, based on their lower bound on communication, proved that LAPACK and other block recursive implementations are asymptotically optimal for a carefully selected block size. The work on lower bounds by Kwasniewski*et al.* [8] leads to the design of parallel distributed $2.5D$ LU (COnfLUX) and Cholesky (COnfCHOX) algorithms. These algorithms perform a volume of communication per node of $\frac{N^3}{P\sqrt{S}} + O(N^2)$.

3 ASSUMPTIONS AND METHODOLOGY

We consider a computational platform with a slow memory of unbounded size, and a fast memory of bounded size S. We fix a given computation described with a computational directed acyclic graph cDAG $G = (V, E)$, where each vertex in V represents a computation operation and each edge in E represents a data dependency between operations. An operation can only be performed if the corresponding input data is in fast memory. We assume that the algorithms explicitly control which data is loaded and removed from the fast memory. The operations in V can be performed in different orders, and we are interested in finding orderings which induce the minimum amount of transfers between slow and fast memory, also called *I/O operations*.

3.1 Lower bound methodology

The lower bounds of this paper are based on a careful application of Lemma 1 in [8], which states:

Lemma 3.1. *Fix a constant $X > S$ and assume that any subcomputation H of a cDAG $G = (V, E)$ which reads at most X elements and writes at most X elements performs a number of operations $|H|$ bounded by $|H| \leqslant H_{\max}$.*

Consider any execution of G with memory S. Its operational intensity ρ is bounded by $\rho \leqslant \frac{H_{\max}}{X-S}$, and its number of I/O operations Q is bounded by

$$Q \geqslant \frac{|V|}{\rho} \geqslant \frac{|V|\,(X-S)}{H_{\max}}.$$

Algorithm 1: Pseudo-code of SYRK, where only the lower triangular part of **C** is referenced and computed.

Input: **A** of size $N \times M$, **C** symmetric of size $N \times N$
Output: C += $\mathbf{A} \cdot \mathbf{A}^\mathsf{T}$
for $i = 1$ to N **do**
  for $j = 1$ to i **do**
    for $k = 1$ to M **do**
      $\mathbf{C}_{i,j}$ += $\mathbf{A}_{i,k} \cdot \mathbf{A}_{j,k}$

Algorithm 2: Pseudo-code of Cholesky, where only the lower triangular parts of **A** and **L** are referenced.

Input: **A** symmetric positive definite of size $N \times N$
Output: Replace **A** with **L** such that $\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^\mathsf{T}$
for $k = 1$ to N **do**
  $\mathbf{A}_{k,k} = \sqrt{\mathbf{A}_{k,k}}$
  for $i = k + 1$ to N **do**
    $\mathbf{A}_{i,k} = \mathbf{A}_{i,k}/\mathbf{A}_{k,k}$
    for $j = k + 1$ to i **do**
      $\mathbf{A}_{i,j}$ −= $\mathbf{A}_{i,k} \cdot \mathbf{A}_{j,k}$ *update operations*

In [8], the number of elements read and written by a subset of computations H are expressed in terms of dominator sets and minimum sets in the graph G. In our case however, the graph is quite regular, so we do not need to introduce these notions.

We consider the SYRK and Cholesky kernels, as described in Algorithms 1 and 2. In the following, N and M always denote the sizes of the matrices used in the kernels. In the Cholesky kernel, for the lower bound target we will focus on the *update operations* only. We can thus describe each operation by a triplet of positive integers (i, j, k), and for both cases we will further ignore the diagonal operations where $i = j$. The sets of operations are denoted $\mathcal{S}$ for the SYRK kernel and $\mathcal{C}$ for Cholesky, and are given by:

$$\mathcal{S} = \left\{(i,j,k) \in [|1,N|]^2 \times [|1,M|] \mid i > j\right\}$$
$$\mathcal{C} = \left\{(i,j,k) \in [|1,N|]^3 \mid i > j > k\right\},$$

where $[|a,b|]$ denotes the set of integers between a and b (inclusive).

We can see that for each statement of these algorithms, the set of written variables is included in the set of read variables, so we only focus on the input data of each operation. In the rest of the paper H is used to denote a set of operations, subset of $\mathcal{S}$ or $\mathcal{C}$.

Definition 3.2. Given a set H of operations, $H_{|k}$ is the restriction of H to iteration k:

$$H_{|k} = \left\{(i,j) \in \mathbb{N}^2 \mid (i,j,k) \in H\right\}.$$

Definition 3.3. Given a subset U of $\mathbb{N}^2$, $\tau(U)$ is the *symmetric footprint* of U:

$$\tau(U) = \{i \in \mathbb{N} \mid \exists j, (i,j) \in U \text{ or } (j,i) \in U\}.$$

If $i > j$ for all $(i,j) \in U$, then $|U| \leqslant \frac{|\tau(U)|(|\tau(U)|-1)}{2}$. In particular, this holds for any $H_{|k}$.

With these definitions, we can express the number of data accessed by a set H: using the SYRK kernel as an example, $\bigcup_k H_{|k}$ is the set of elements $C_{i,j}$ accessed by H, and for any k, $\tau\left(H_{|k}\right)$ is the set of $A_{i,k}$ elements accessed by H.

Proposition 3.4. *For any set H of operations, the number of data accessed by H is*

$$D(H) = \left|\cup_k H_{|k}\right| + \sum_k \left|\tau\left(H_{|k}\right)\right|.$$

3.2 Triangle blocks

Many results in this paper are obtained by considering *triangle blocks*, which are generalizations of the diagonal tiles in a tile decomposition of a symmetric matrix. In particular, the SYRK lower bound shows that accessing the result matrix by triangle blocks is the most efficient, and the TBS algorithm describes how to partition the result matrix in disjoint triangle blocks. Figure 1 (page 7) depicts examples of triangle blocks.

Definition 3.5 (Triangle block). Given a set R of integer indices, the *triangle block* $\mathrm{TB}(R)$ is the set of all subdiagonal pairs of elements of R:

$$\mathrm{TB}(R) = \left\{(r,r') \mid r,r' \in R \text{ and } r > r'\right\}$$

It is clear that $|\mathrm{TB}(R)| = \frac{|R|(|R|-1)}{2}$. We say that $\mathrm{TB}(R)$ has side length $|R|$.

For any $m \in \mathbb{N}$, we define $\sigma(m)$ as the smallest possible side length of a triangle block with at least m elements. $\sigma(m)$ is thus the smallest element of $\mathbb{N}$ such that $m \leqslant \frac{\sigma(m)(\sigma(m)-1)}{2}$. By solving the quadratic equation, we get:

Lemma 3.6. *For $m \in \mathbb{N}^*$, $\sigma(m) = \lceil\sqrt{\frac{1}{4} + 2m} + \frac{1}{2}\rceil$, and $\sigma(0) = 0$.*

For any $m \in \mathbb{N}$, we define $T(m)$ as any size-m subset of $\mathrm{TB}([|1,\sigma(m)|])$. We use $T(m)$ as a canonical way of performing m computations in an iteration, while minimizing the number of data accesses. Indeed, by definition $|T(m)| = m$, and it is easy to see that $|\tau(T(m))| = \sigma(m)$.

4 LOWER BOUNDS

4.1 Symmetric Multiplication (SYRK)

As mentioned above, in order to obtain a lower bound on the data movements required for the SYRK computation, we first provide an upper bound on the largest subcomputation H than can be performed while accessing at most X data elements. We are thus looking for (a bound on) the optimal value of the following optimization problem:

$$\mathcal{P}(X): \quad \max \ |H|$$
$$\text{s.t.} \quad D(H) = \left|\cup_k H_{|k}\right| + \sum_{k=1}^{M} \left|\tau\left(H_{|k}\right)\right| \leqslant X$$
$$H \subseteq \mathcal{S}$$

The main result of this section can be stated as:

Theorem 4.1. *The optimal value of $\mathcal{P}(X)$ is at most $\frac{\sqrt{2}}{3\sqrt{3}}X^{\frac{3}{2}}$.*

To prove this theorem, we first show that $\mathcal{P}(X)$ admits triangle-shaped optimal solutions, which we call *balanced solutions*. We then compute an upper bound on the size of such a balanced solution.

4.1.1 Balanced Solutions.

Definition 4.2. For given x and m, we define the *balanced solution* $B = B(x,m)$ by:

$$\begin{cases} B_{|k} = T(m) & \text{for all } k \in [|0, K-1|], \\ B_{|K} = T(m') & \text{for } k = K, \\ B_{|k} = \emptyset & \text{for all } k > K, \end{cases}$$

where $K = \lfloor \frac{x}{m} \rfloor$ and $m' = x - Km < m$.

It is clear that $|B(x,m)| = x$ (since $K \cdot m + (x - Km) = x$) and $\left|\cup_k B(x,m)_{|k}\right| = m$. The next lemma shows that any solution H can be turned into a balanced solution with lower cost.

Lemma 4.3. *If H is a solution of $\mathcal{P}(X)$, let the corresponding balanced solution be $B = B\left(|H|, \max_k \left|H_{|k}\right|\right)$. Then $D(B) \leqslant D(H)$.*

Proof. Given a solution H, let us define $m_k = \left|H_{|k}\right|$ and denote $m = \max_k m_k$. As mentioned above, we have $|B| = |H|$ and $\left|\cup_k B_{|k}\right| = m = \max_k m_k \leqslant \left|\cup_k H_{|k}\right|$. Furthermore, since $\sum_k m_k = |H| = K \cdot m + m'$ and since the $\sigma(\cdot)$ function is concave, we have:

$$\begin{aligned} \sum_k \left|\tau\left(B_{|k}\right)\right| &= K\sigma(m) + \sigma(m') \\ &\leqslant \sum_k \sigma(m_k) \\ &= \sum_k \left|\tau\left(H_{|k}\right)\right| \end{aligned}$$

This shows that $D(B) \leqslant D(H)$. □

In particular, if H is an optimal solution, we obtain the following corollary.

Corollary 4.4. *There exist x and m such that $B(x,m)$ is an optimal solution to $\mathcal{P}(X)$.*

4.1.2 Optimal Balanced Solution.

A balanced solution B can be described with three integer values I, J in $[|1,N|]$ with $J \leqslant I$, and $K \in [|1,M|]$ such that

$$\begin{cases} \forall k \in [|0,K-1|],\ B_k = T(I) \\ B_K = T(J) \end{cases}$$

Such a solution satisfies $|B| = K\frac{I(I-1)}{2} + \frac{J(J-1)}{2}$ and $D(B) = \frac{I(I-1)}{2} + KI + J$. By relaxing integrity constraints and upper bounds on I, J, K, we get that the optimal size of a balanced solution is at most the optimal value of the following problem $\mathcal{P}'(X)$:

$$\begin{aligned} \mathcal{P}'(X): \quad & \max\left(K\frac{I(I-1)}{2} + \frac{J(J-1)}{2}\right) \\ & \text{s.t.} \quad \begin{cases} \frac{I(I-1)}{2} + KI + J \leqslant X \\ J \leqslant I \end{cases} \end{aligned}$$

Lemma 4.5. *For any (I,J,K) solution to $\mathcal{P}'(X)$, define $K' = K + \frac{J(J-1)}{I(I-1)}$. Then $(I,0,K')$ is a solution to $\mathcal{P}'(X)$ with the same value.*

Proof. The solution $(I,0,K')$ is feasible:

$$\begin{aligned} \frac{I(I-1)}{2} + K'I &= \frac{I(I-1)}{2} + KI + J\frac{J-1}{I-1} \\ &\leqslant \frac{I(I-1)}{2} + KI + J && \text{since } J \leqslant I \\ &\leqslant X && \text{since } (I,J,K) \text{ is feasible} \end{aligned}$$

Furthermore, its objective value is $K'\frac{I(I-1)}{2} = K\frac{I(I-1)}{2} + \frac{J(J-1)}{2}$, which is the objective value of (I,J,K). □

This lemma shows that the optimum value of $\mathcal{P}'$ is equal to the optimum value of the simpler $\mathcal{P}''$ problem below:

$$\begin{aligned} \mathcal{P}''(X): \quad & \max\left(K\frac{I(I-1)}{2}\right) \\ & \text{s.t.} \quad \frac{I(I-1)}{2} + KI \leqslant X \end{aligned}$$

This problem is now simple enough and we can provide a direct bound on its optimum value.

Lemma 4.6. *The optimum value of $\mathcal{P}''(X)$ is at most $\frac{\sqrt{2}}{3\sqrt{3}}X^{\frac{3}{2}}$.*

Proof. Reformulated as a minimization problem, $\mathcal{P}''(X)$ becomes:

$$\begin{aligned} & \min\left(f(K,I) = -K\frac{I(I-1)}{2}\right) \\ \text{s.t.} \quad & g(K,I) = \frac{I(I-1)}{2} + KI - X \leqslant 0 \end{aligned}$$

Since the regularity conditions are met over the whole definition space of variables I and K, we can write Karush-Kuhn-Tucker necessary conditions: if (K,I) is a local optimum for $\mathcal{P}''(X)$ then

$$\begin{aligned} & \exists u \geqslant 0, \quad \nabla f(K,I) + u\nabla g(K,I) = 0 \\ \Leftrightarrow\ & \exists u \geqslant 0, \quad \begin{cases} -K(I - \frac{1}{2}) + u(I - \frac{1}{2} + K) = 0 \\ -\frac{I(I-1)}{2} + uI = 0 \end{cases} \end{aligned}$$

which implies $u = \frac{I-1}{2}$, and then $KI = (I-1)(I-\frac{1}{2})$.

Let us denote by (K,I) a local minimum of f. Then $KI = (I-1)(I-\frac{1}{2})$. Besides we can select (K,I) such that $\frac{I(I-1)}{2} + KI - X = 0$. This yields $3I^2 - 4I - (2X-1) = 0$, and we obtain $I = \frac{2}{3} + \frac{\sqrt{1+6X}}{3}$.

An optimal solution of $\mathcal{P}''(X)$ is thus given by

$$\begin{cases} I^* = \frac{2}{3} + \frac{\sqrt{1+6X}}{3} \\ K^* = (I^* - \frac{1}{2})(1 - \frac{1}{I^*}) \end{cases}$$

and its objective value is

$$\begin{aligned} \mathcal{H}''(X) &= K^*\frac{I^*(I^*-1)}{2} \\ &= \frac{1}{4}(I^*-1)^2(2I^*-1) \\ &= \frac{1}{108}(\sqrt{1+6X}-1)^2(2\sqrt{1+6X}+1) \\ &\leqslant \frac{(\frac{\sqrt{6X}}{3})^3}{2} = \frac{\sqrt{2}}{3\sqrt{3}}X^{\frac{3}{2}} \end{aligned}$$

To understand why the last inequality holds, one can observe that the function $X \mapsto \mathcal{H}''(X) - \frac{\sqrt{2}}{3\sqrt{3}}X^{\frac{3}{2}}$ equals 0 for $X = 0$. Besides,

$$\frac{\partial}{\partial X}\left[\mathcal{H}''(X) - \frac{\sqrt{2}}{3\sqrt{3}}X^{\frac{3}{2}}\right] = \frac{1}{6}\left(\sqrt{1+6X} - 1\right) - \sqrt{\frac{X}{6}}$$
$$= \frac{1}{6}\left[\sqrt{1+6X} - (1+\sqrt{6X})\right]$$

which is obviously negative. □

4.1.3 Final Result.

Proof of Theorem 4.1. The result follows directly from Corollary 4.4, Lemma 4.5 and Lemma 4.6. □

Corollary 4.7. *The number of data accesses required to perform a SYRK operation where* $\mathbf{A}$ *has size* $N \times M$*, with memory* S*, is at least*

$$Q_{SYRK}(N, M, S) \geqslant \frac{1}{\sqrt{2}}\frac{N^2M}{\sqrt{S}}.$$

Proof. Consider the computational DAG of the SYRK operation, which has $|\mathcal{S}| = \frac{N^2M}{2}$ vertices. According to Theorem 4.1, for any X, any subcomputation H of this DAG which reads at most X elements has size $|H| \leqslant \frac{\sqrt{2}}{3\sqrt{3}}X^{\frac{3}{2}}$.

In particular[1], for $X = 3S$, we get $|H| \leqslant \sqrt{2} \cdot S^{\frac{3}{2}}$. According to Lemma 3.1, the maximal operational intensity of SYRK is $\rho = \frac{|H|}{3S-S} \leqslant \sqrt{\frac{S}{2}}$. This yields the following bound on the number of data accesses for the complete SYRK operation:

$$Q_{\text{SYRK}}(N, M, S) \geqslant \frac{|\mathcal{S}|}{\rho} = \frac{1}{\sqrt{2}}\frac{N^2M}{\sqrt{S}}.$$

□

4.2 Cholesky factorization

We now consider the Cholesky factorization, as described by Algorithm 2. As mentioned above, we focus on the *update operations*, described by the set

$$\mathcal{C} = \left\{(i, j, k) \in [|1, N|]^3 \mid i > j > k\right\}.$$

For a given X, the largest subset H that accesses at most X elements can be found by solving $\mathcal{P}(X)$, in which the constraint $H \subseteq \mathcal{S}$ is replaced by $H \subseteq \mathcal{C}$. We consider a relaxed version, in which the constraint is instead $H \subseteq \mathcal{C}'$, where

$$\mathcal{C}' = \left\{(i, j, k) \in [|1, N|]^3 \mid i > j\right\}.$$

Since $\mathcal{C} \subseteq \mathcal{C}'$, the optimal value of this relaxed version is not smaller than the optimal value of the original one. We can now remark that the relaxed version is a special case of $\mathcal{P}(X)$ where $M = N$, so that we can directly apply Theorem 4.1, which leads to the following corollary:

[1]The value $X = 3S$ is chosen to obtain the strongest possible bound by maximizing the ratio $\frac{|H|}{X-S}$.

Corollary 4.8. *The number of data accesses required to perform a Cholesky operation on a matrix* $\mathbf{A}$ *of size* $N \times N$*, with memory* S*, is at least*

$$Q_{Chol}(N, S) \geqslant \frac{1}{3\sqrt{2}}\frac{N^3}{\sqrt{S}}.$$

Proof. The computational DAG of the update operations of the Cholesky kernel contains $|\mathcal{C}| = \frac{N^3}{6}$ update operations. According to Theorem 4.1, for any X, any subcomputation H of this DAG which reads at most X elements has size $|H| \leqslant \frac{\sqrt{2}}{3\sqrt{3}}X^{\frac{3}{2}}$.

As previously, we apply Lemma 3.1 to the case where $X = 3S$, and obtain that the maximal operational intensity of the update operations in Cholesky is $\rho = \frac{|H|}{3S-S} \leqslant \sqrt{\frac{S}{2}}$. Since a Cholesky kernel needs to perform all update operations, this yields the following bound on the number of data accesses

$$Q_{\text{Chol}}(N, S) \geqslant \frac{|\mathcal{C}|}{\rho} = \frac{1}{3\sqrt{2}}\frac{N^3}{\sqrt{S}}.$$

□

5 COMMUNICATION-OPTIMAL ALGORITHMS

In this section, we propose algorithms which perform the same operations as Algorithms 1 and 2, but with an ordering that allows to perform fewer I/O operations. We start by presenting an algorithm for the SYRK kernel, which we then use to design an algorithm for the Cholesky kernel.

To simplify the presentation of the algorithms, we index the matrices in the range $[|0, N-1|]$ instead of $[|1, N|]$. Our algorithms rely on previously proposed algorithms from Béreux [4], more specifically the one-tile, narrow-block variants of **OOC_SYRK** and **OOC_TRSM**, and the one-tile, left-looking variant of Cholesky **OOC_CHOL**. For conciseness, we denote them respectively by OCS, OCT and OCC, with the following number of I/O operations:

$$Q_{\text{OCS}}(N, M) = \frac{N^2M}{\sqrt{S}} + O(NM)$$
$$Q_{\text{OCT}}(N, M) = \frac{N^2M}{\sqrt{S}} + O(NM)$$
$$Q_{\text{OCC}}(N) = \frac{N^3}{3\sqrt{S}} + O(NM)$$

The analysis of communication cost in this section is asymptotic in the following sense: we assume that S remains constant, and that the sizes N and M of the matrices grow without bounds.

In the following algorithms, given a matrix $\mathbf{A}$ and two sets of indices X and Y, we use $\mathbf{A}[X, Y]$ to denote the submatrix of $\mathbf{A}$ indexed with indices in $X \times Y$.

5.1 TBS: Triangular Block SYRK

The proof of Theorem 4.1 shows that the largest operational intensity in the SYRK kernel is achieved when computing the elements of $\mathbf{C}$ in a triangle $T(m)$, which is located at the top-left of matrix $\mathbf{C}$. The result of Corollary 4.7 is tight if all (or at least most) parts of the computation have the same operational intensity. But it is

not clear whether it is possible to tile the whole computation space with triangles. It is easy around the diagonal, but what about the elements of the matrix away from the diagonal?

Algorithm 3: Generic out-of-core SYRK algorithm

```
Partition C in blocks of size S
for each block B do
    Load the corresponding elements of C in memory
    for i = 0 to M − 1 do
        Load the required elements of A[·, i]
        Update block B with these elements
    Remove block B from memory
```

Our algorithm uses the generic scheme described in Algorithm 3: store a block of elements of the result matrix in memory, and iteratively load elements from A to update this block. To maximize memory efficiency, it makes sense that blocks would contain S elements. In the **OOC_SYRK** algorithm proposed by Béreux, the blocks are squares of $\sqrt{S} \times \sqrt{S}$, which is the optimal shape without data reuse (for example, squares are the optimal shape for non-symmetric GEMM multiplication). As mentioned above, in order to match the lower bound for SYRK, we need to have blocks shaped as triangles, up to row and column reordering: such blocks are *triangle blocks* $\mathrm{TB}(R)$, as defined in Definition 3.5. Indeed, $\mathrm{TB}(R)$ is the set of indices of the elements of C that can be updated with elements of A whose row belong in R.

We prove here that it is actually possible to tile (almost all) the result matrix C with triangle blocks, each containing roughly S elements.

5.1.1 Partitioning the result matrix. We fix k such that

$$S \geqslant k + \frac{k(k-1)}{2} = \frac{k(k+1)}{2}.$$

This ensures that the memory can fit a triangle of side length k from the result matrix **C**, plus a vector of k elements of **A** used for the update. Let us assume for the moment that $N = ck$ for some value c. We will see later that not all values of c are eligible, and we will discuss how to choose an appropriate value. We decompose the result matrix **C** in $\frac{k(k-1)}{2}$ square *zones* of size $c \times c$. The rest of the matrix (k triangle-shaped zones on the diagonal) will be considered later. In **TBS**, a block contains exactly one element from each of these square zones, as depicted in Figure 1. For $0 \leqslant i, j < c$, we denote by $B^{i,j}$ the block which contains the element (i, j) of the top-most zone (which is the element $(i + c, j)$ of the matrix **C**).

Let $R^{i,j}$ be the row indices of block $B^{i,j}$. Since we search for blocks with one element per zone, we can write

$$B^{i,j} = \mathrm{TB}(R^{i,j}), \quad \text{with} \quad R^{i,j} = \left\{u \cdot c + f^{i,j}(u) \mid 0 \leqslant u < k\right\}, \quad (1)$$

where $0 \leqslant f^{i,j}(u) < c$ gives the position of the row of $B^{i,j}$ within the u-th row of zones (see left of Figure 2). To ensure that $B^{i,j}$ contains $(i + c, j)$, we just need to have $f^{i,j}(0) = j$ and $f^{i,j}(1) = i$. We can thus specify our triangle blocks with an *indexing family*:

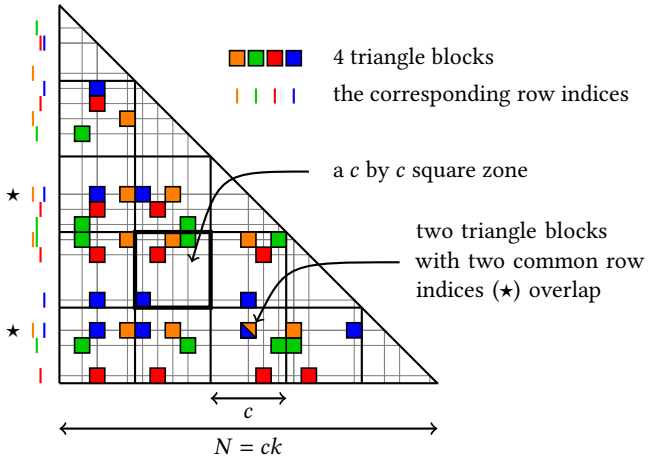

Figure 1: Zones and blocks in the TBS algorithm. Each block has one element in each zone.

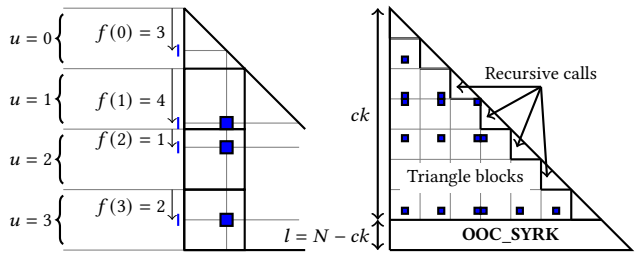

Figure 2: *Left:* $f^{i,j}(u)$ gives the position of the row of $B^{i,j}$ within the u-th row of zones. *Right:* which parts of the matrix C are computed by which method in the TBS Algorithm.

Definition 5.1 (Indexing family). A (c, k)-*indexing family* is a family of functions $f^{i,j}(u)$, defined for (i, j) in $[|0, c-1|]^2$, with:

$$f^{i,j} : [|0, k-1|] \mapsto [|0, c-1|]$$
$$\forall i, j, \quad f^{i,j}(0) = j \quad \text{and} \quad f^{i,j}(1) = i$$

To enforce the validity of the algorithm, triangle blocks $B^{i,j}$ must not overlap. If two functions $f^{i,j}$ and $f^{i',j'}$ agree for two different values u and v, the corresponding blocks $B^{i,j}$ and $B^{i',j'}$ have two row indices in common, and as can be seen on Figure 1, these blocks are not disjoint. We thus need to consider *valid* indexing families:

Definition 5.2 (Validity). A (c, k)-indexing family f is *valid* if

$$\forall u \neq v, \begin{cases} f^{i,j}(u) & = f^{i',j'}(u) \\ f^{i,j}(v) & = f^{i',j'}(v) \end{cases} \implies i = i' \text{ and } j = j'.$$

It turns out that this condition is sufficient to ensure no collisions:

Lemma 5.3. *If f is a valid (c, k)-indexing family, then the sets $B^{i,j}$ defined in Equation 1 are pairwise disjoint.*

Proof. We prove the contrapositive of this statement: if two $B^{i,j}$ sets are not disjoint, then f is not valid. Indeed, let us consider two different pairs (i,j) and (i',j') such that $B^{i,j} \cap B^{i',j'} \neq \emptyset$. There exist (u,v) and (u',v'), with $u \neq v$ and $u' \neq v'$, such that:

$$uc + f^{i,j}(u) = u'c + f^{i',j'}(u')$$
$$vc + f^{i,j}(v) = v'c + f^{i',j'}(v')$$

Since the values of an indexing function are in $[|0, c-1|]$, this implies $u = u'$ and $v = v'$.

Thus, there exist $u \neq v$ and i, j, i', j', with $(i,j) \neq (i',j')$, such that $f^{i,j}(u) = f^{i',j}(u)$ and $f^{i,j}(v) = f^{i',j'}(v)$: f is not valid. □

This shows that using a valid indexing family allows to partition the square zones from Figure 1 in disjoint triangle blocks. The remaining elements, from the triangular zones close to the diagonal, can be computed by recursive calls to the **TBS** algorithm. We thus require several valid indexing families for a fixed k and different values of c, since the recursive calls will be made with different value of c. However, we see below that we cannot obtain valid indexing families for all values of c, so we are not yet ready to describe the complete algorithm.

5.1.2 Defining a valid indexing family. In this section, we show that it is possible to define a valid indexing family for some values of $c \geqslant k-1$. We do this using simple modulo operations:

Definition 5.4. The *cyclic* (c,k)-indexing family is defined by:

$$f_c^{i,j}(u) = \begin{cases} j & \text{if } u = 0 \\ i + j(u-1) \mod c & \text{if } u > 0 \end{cases}$$

Lemma 5.5. *If $c \geqslant k-1$ is coprime with all integers in $[|2, k-2|]$, then the cyclic indexing family f_c is valid.*

Proof. Consider any $u, v \in [|0, k-1|]$ with $u < v$, and assume that i, j, i', j' in $[|0, c-1|]$ are such that $f_c^{i,j}(u) = f_c^{i',j'}(u)$ and $f_c^{i,j}(v) = f_c^{i',j'}(v)$.

We first prove $j = j'$. If $u = 0$, this is direct. Otherwise, we can write

$$\begin{cases} i + j(u-1) &= i' + j'(u-1) \mod c \\ i + j(v-1) &= i' + j'(v-1) \mod c \end{cases}$$
$$\Leftrightarrow \begin{cases} i - i' &= (j'-j)(u-1) \mod c \\ i - i' &= (j'-j)(v-1) \mod c \end{cases}$$

This implies:

$$(j'-j)(u-1) = (j'-j)(v-1) \mod c$$
$$\Leftrightarrow (j'-j)(u-v) = 0 \mod c$$

Since $u < v$, $0 < u, v \leqslant k-1$, we know that $0 < v - u \leqslant k-2$. From our assumption, $v-u$ is coprime with c, so we obtain $j'-j = 0 \mod c$, and thus $j = j'$.

Then, since $i + j(v-1) = i' + j(v-1) \mod c$, we deduce $i = i' \mod c$. Since i, i' are in $[|0, c-1|]$, we have $i = i'$. □

We define the constant integer q as the product of all primes no larger than $k-2$: $q = \prod_{p \text{ prime}, p \leqslant k-2} p$. Then c is coprime with all integers in $[|2, k-2|]$ if and only if c is coprime with q. Notice that q is constant: it only depends on k, thus on S, but not on N or M.

Algorithm 4: Triangle Block Syrk **TBS**(**A**, **C**)

Input: matrices **A** of size $N \times M$ and **C** symmetric of size $N \times N$
Output: **C** += **A** · **A**$^\mathsf{T}$
Assumes: memory of size $S = \frac{k(k+1)}{2}$

```
q ← product of all primes in [|2, k-2|]
c ← the largest integer coprime with q below N/k
l ← N - ck
if c < k - 1 then                                   c is too small
    OOC_SYRK (A, C);
else
    Use OOC_SYRK to compute the last l rows of C
    for i = 0 to k - 1 do                           recursive calls for triangular zones
        R ← [|ic, (i+1)c|]
        TBS(A[R, ·], C[R, R])
    for (i, j) ∈ [|0, c-1|]^2 do                    loop over all blocks
        R ← {r_u = uc + f_c^{i,j}(u) | 0 ⩽ u < k}    see Def. 5.4
        Load the elements of C indexed by TB(R)
        for i = 0 to M - 1 do                       loop over columns of A
            Load elements of A indexed by {(r, i) | r ∈ R}
            for u = 0 to k - 1 do                   loops over elements
                for v = 0 to u - 1 do               of the block
                    C_{r_u, r_v} += A_{r_u, i} · A_{r_v, i}
```

Now that we know how to build valid indexing families, we are ready to describe the **TBS** algorithm. However, with the constraints on c imposed by Lemma 5.5, it is not possible to use triangle blocks on the whole matrix **C**. Instead, given a matrix size N, we set c to be the largest number coprime with q such that $c \leqslant \frac{N}{k}$. If the obtained c is lower than $k-1$, we can use the simple **OOC_SYRK** with square blocks. Otherwise, c satisfies the condition of Lemma 5.5, so we can use triangle blocks to compute the first ck rows of C, and the **OOC_SYRK** algorithm for the remaining $l = N - ck$ rows (see right of Figure 2). The resulting algorithm is called **TBS**, and is described in Algorithm 4.

5.1.3 Communication cost analysis. Let us first notice that the **TBS** algorithm loads each entry of C exactly once (even for the elements computed with **OOC_SYRK**), so loading these elements has a communication cost of $\frac{N^2}{2}$. In the following, we denote by $\widetilde{Q}_{\text{TBS}}(N,M)$ the communication cost of **TBS** related to elements of **A**, for a matrix **A** of size $N \times M$.

The definition of c yields $\frac{N}{k} = c+g$, and we need an upper bound on g to estimate the amount of work performed by **OOC_SYRK**. It is easy to see that for any integer a, $aq+1$ is coprime with q. In particular, $\lfloor N/kq \rfloor q + 1$ is coprime with q, thus $c \geqslant \lfloor N/kq \rfloor q + 1$, and $g \leqslant q$. Since q only depends on S and not on N or M, we get $g = O(1)$. Even though q is a constant, it may be considered very large relative to S. However, the bound $g \leqslant q$ is very pessimistic:

sieve methods allow to show that the number of integers coprime with q in any interval $[|(a-1)q, aq-1|]$ is exactly $\prod(p-1)$, where p spans the prime numbers below $k-1$ (see Example 1.5 in [5]). In practice, one can expect the value of g to be much lower than q.

We first consider the elements computed with the **TBS** algorithm (in the first ck rows). There are c^2 triangle blocks, and each triangle block loads kM elements of $\mathbf{A}$. This yields a communication cost $Q_1 = c^2kM$, and with $c \leqslant \frac{N}{k}$, we obtain $Q_1 \leqslant \frac{N^2M}{k}$.

Elements computed with **OOC_SYRK** (in the last $l = gk$ rows) are computed by square $\sqrt{S} \times \sqrt{S}$ blocks, and each block loads $2M\sqrt{S}$ elements from matrix $\mathbf{A}$. Since there are at most gkN such elements, this yields a communication cost $Q_2 \leqslant \frac{gkN}{S} \cdot 2M\sqrt{S} = O(NM)$.

Adding the elements covered by the recursive calls, we get:

$$\widetilde{Q}_{\text{TBS}}(N,M) \leqslant \frac{N^2M}{k} + k\widetilde{Q}_{\text{TBS}}\left(\frac{N}{k}, M\right) + O(NM)$$

We can iteratively apply this inequality t times, where t is the smallest integer such that $\frac{N}{k^t} < k-1$. We thus have $k^{t-1} < \frac{N}{k}$, and $t = O(\log N)$. Then we get:

$$\begin{aligned}
\widetilde{Q}_{\text{TBS}}(N,M) &\leqslant \sum_{i=1}^{t} \frac{N^2M}{k^i} + k^t\widetilde{Q}_{\text{OCS}}(k,M) + t \cdot O(NM) \\
&\leqslant \sum_{i=1}^{\infty} \frac{N^2M}{k^i} + N \cdot \frac{k^2M}{\sqrt{S}} + O(NM\log N) \\
&\leqslant N^2M\left(\frac{1}{1-\frac{1}{k}} - 1\right) + O(NM\log N) \\
&\leqslant \frac{N^2M}{k-1} + O(NM\log N)
\end{aligned}$$

Remember that k is defined by $S = \frac{k(k+1)}{2}$, so that $k - 1 \simeq \sqrt{2S}$. In total (with the communications required to load elements of $\mathbf{C}$), we get:

Theorem 5.6. *The total communication cost $Q_{TBS}(N,M)$ of the **TBS** algorithm for a matrix $\mathbf{A}$ of size $N \times M$, with a memory of size S, is bounded by:*

$$Q_{TBS}(N,M) \leqslant \frac{1}{\sqrt{2}} \cdot \frac{N^2M}{\sqrt{S}} + \frac{N^2}{2} + O(NM\log N)$$

*5.1.4 Tiled version of **TBS**.* The **TBS** algorithm, as presented in Algorithm 4 achieves an asymptotic complexity which matches the lower bound from Theorem 4.7. However, this requires very large values of N, since the condition $c \geqslant k-1$ together with $k \simeq \sqrt{2S}$ means that the triangular block approach can only be used for $N \geqslant 2S$. In that case, the matrix is so large that half a column does not fit in memory.

To make the **TBS** algorithm more practical, it is possible to design a *tiled* version of it, where elements of $\mathbf{C}$ are no longer considered individually, but as tiles of size $b \times b$. We thus choose b and k such that $S = b^2\frac{k(k-1)}{2}$, and set $c = \frac{N}{kb}$ (actually the largest integer coprime with q below this value). Instead of loading elements of C, we thus load complete tiles; however we still load elements of $\mathbf{A}$ one row at a time. The update operation $\mathbf{C}_{r_u,r_v}$ += $\mathbf{A}_{r_u,i} \cdot \mathbf{A}_{r_v,i}$ thus becomes an outer product.

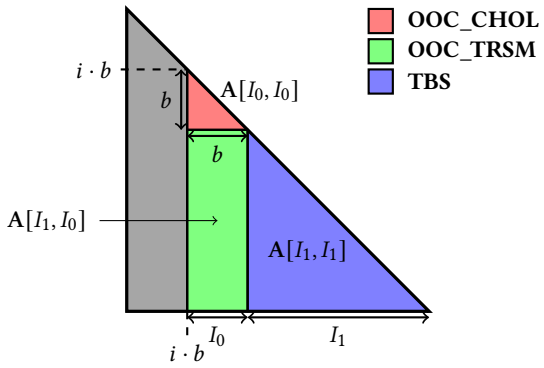

Figure 3: Algorithm LBC: updating the three parts of A at iteration i

The communication cost analysis is very similar, only the value of Q_1 changes. There are still c^2 blocks, each of which loads kbM elements of $\mathbf{A}$. We get $Q_1 = c^2kbM$, with $c \leqslant \frac{N}{kb}$. Thus $Q_1 \leqslant \frac{N^2M}{kb}$. In turn, this yields $\widetilde{Q}_{\text{TBS}}(N,M) \leqslant \frac{N^2M}{(k-1)b} + O(NM\log N)$. With $b = \sqrt{\frac{2S}{k(k-1)}}$, we get:

$$Q_{\text{TBS}}(N,M) \leqslant \frac{N^2M}{\sqrt{2S \cdot \frac{k-1}{k}}} + \frac{N^2}{2} + O(NM\log N).$$

The leading term is now larger than the lower bound by a factor $\sqrt{k/(k-1)}$, but this tiled version of the algorithm is valid for smaller values of N. Indeed, the constraint $c \geqslant k-1$ implies $N \geqslant \frac{2S}{b} = \sqrt{2S \cdot k(k-1)}$, and thus $\frac{N^2}{2} \geqslant k(k-1)$: **TBS** is useful as soon as storing the matrix requires $k(k-1)$ times more memory than available.

5.2 LBC: Large Block Cholesky

The lower bound detailed in Section 4.2 is based on the idea that Cholesky factorization generates at least as many data transfers as SYRK operation. Since TBS algorithm performs the SYRK kernel with the minimum amount of I/O operations, the idea is to use it for the largest possible part of the computation of the Cholesky factorization.

5.2.1 Algorithm description. We implement this strategy in the **Large Block Cholesky** (LBC) algorithm. It is a right-looking, blocked algorithm which performs the Cholesky factorization of any input symmetric positive definite matrix $\mathbf{A}$ making use of **OOC_CHOL**, **OOC_TRSM** and **TBS** algorithms. Note that it would be possible to use a recursive call to **LBC** instead of **OOC_CHOL**, since **LBC** performs fewer transfers. However, it turns out that the successive Cholesky factorizations of $\mathbf{A}[I_0, I_0]$ do not contribute to the higher order term, so we opt for **OOC_CHOL** to simplify the presentation. LBC modifies $\mathbf{A}$ in-place to yield a lower triangular matrix $\mathbf{L}$ as output such that $\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^{\mathsf{T}}$. The steps of the algorithm are detailed in Algorithm 5 and described on Figure 3.

LBC is a so-called *right-looking* variant of Cholesky factorization. At each iteration, the final values of the two leftmost panels

Algorithm 5: Large Block Cholesky **LBC**(**A**)

Input: **A**: $N \times N$ symmetric positive definite matrix
Input: b: block size
Assumes: $b|N$
Output: **L**: $N \times N$ lower triangular matrix s.t. $\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^\mathsf{T}$
for $i = 0$ to $\lfloor \frac{N}{b} \rfloor$ **do**
  $I_0 = [|i \cdot b, (i+1) \cdot b|]$
  $\mathbf{A}[I_0, I_0] \leftarrow$ **OOC_CHOL** $(\mathbf{A}[I_0, I_0])$
  if $(i+1) \cdot b < N$ **then**
    $I_1 = [|(i+1) \cdot b, N|]$
    $\mathbf{A}[I_1, I_0] \leftarrow$ **OOC_TRSM** $(\mathbf{A}[I_0, I_0], \mathbf{A}[I_1, I_0])$
    $\mathbf{A}[I_1, I_1] \leftarrow$ **TBS** $(\mathbf{A}[I_1, I_0], \mathbf{A}[I_1, I_1])$

$\mathbf{A}[I_0, I_0]$ and $\mathbf{A}[I_1, I_0]$ are computed; $\mathbf{A}[I_1, I_0]$ is then used to update the right panel $\mathbf{A}[I_1, I_1]$ whose values are still temporary. By contrast, *left-looking variants* perform all the updates of a given value of **A** one after the other, allowing to write each element only once.

Right-looking implementations of Cholesky are known to perform more I/O operations than their left-looking counterparts, because the lower right panel $\mathbf{A}_{I_1,I_1}$ needs to be reloaded at each iteration, so as to be updated using the SYRK kernel. Nevertheless, this overhead can be rendered negligible. Indeed, the main point of **LBC** is to use large enough blocks (of size $\sqrt{N}$), so that the number of iterations is low ($\sqrt{N}$): then, the volume of communications induced by loading $\mathbf{A}_{I_1,I_1}$ remain negligible compared to the one required to update its values.

5.2.2 Communication cost analysis. Let us now analyze the total number of I/O operations required by Algorithm **LBC** on an $N \times N$ matrix **A**; it is denoted $Q_{\text{LBC}}(N)$. As mentioned above, we get from [4] that $Q_{\text{OCT}}(N, M) = \frac{N^2 M}{\sqrt{S}} + O(NM)$ and $Q_{\text{OCC}}(N) = \frac{N^3}{3\sqrt{S}} + O(NM)$. Furthermore, as detailed in 5.1, we also know that $Q_{\text{TBS}}(N, M) = \frac{1}{\sqrt{2}} \frac{N^2 M}{\sqrt{S}} + \frac{N^2}{2} + O(NM \log N)$. Then:

$$
\begin{aligned}
Q_{\text{LBC}}(N) &= \sum_{i=1}^{\frac{N}{b}} Q_{\text{OCC}}(b) + Q_{\text{OCT}}\left(b, \left(\frac{N}{b} - i\right) b\right) + Q_{\text{TBS}}\left(\left(\frac{N}{b} - i\right) b, b\right) \\
&= \frac{N}{b} Q_{\text{OCC}}(b) + \sum_{i=1}^{\frac{N}{b}} Q_{\text{OCT}}(b, ib) + Q_{\text{TBS}}(ib, b) \\
&= \frac{b^2 N}{3\sqrt{S}} + O(b^2) + \\
&\quad + \sum_{i=1}^{\frac{N}{b}} \left(\frac{b^2(ib)}{\sqrt{S}} + \frac{b(ib)^2}{\sqrt{2}\sqrt{S}} + \frac{(ib)^2}{2} + O(b^2 i \log(ib)) \right)
\end{aligned}
$$

Since $0 < b < N$, $O(b^2) = O(N^2)$.
Besides:

$$
\sum_{i=1}^{\frac{N}{b}} O(b^2 i \log(ib)) \leqslant \sum_{i=1}^{\frac{N}{b}} O(b^2 \tfrac{N}{b} \log N) = \frac{N}{b} O(Nb \log N) = O(N^2 \log N)
$$

.

The number of data transfers necessary to perform algorithm **LBC** is therefore:

$$
\begin{aligned}
Q_{\text{LBC}}(N) &\leqslant \frac{b^2 N}{3\sqrt{S}} + \sum_{i=1}^{\frac{N}{b}} \left(\frac{b^2(ib)}{\sqrt{S}} + \frac{b(ib)^2}{\sqrt{2}\sqrt{S}} + \frac{(ib)^2}{2} \right) + O(N^2 \log N) \\
&\leqslant \frac{b^2 N}{3\sqrt{S}} + \frac{b^3 (\frac{N}{b})^2}{2\sqrt{S}} + \frac{b^3 (\frac{N}{b})^3}{3\sqrt{2}\sqrt{S}} + \frac{b^2 (\frac{N}{b})^3}{6} + O(N^2 \log N) \\
&\leqslant \underbrace{\frac{b^2 N}{3\sqrt{S}}}_{(1)} + \underbrace{\frac{bN^2}{2\sqrt{S}}}_{(2)} + \underbrace{\frac{N^3}{3\sqrt{2}\sqrt{S}}}_{(3)} + \underbrace{\frac{\frac{N^3}{b}}{6}}_{(4)} + O(N^2 \log N)
\end{aligned}
$$

As previously discussed the volume of data transfers induced by loading $\mathbf{A}_{I_1,I_1}$ at each step (4) clearly becomes dominant if b is a constant. On the other hand, if the chosen value for b is of order N, the communications required to perform all TRSM operations (2) becomes dominant. Hence, to ensure that the volume of data transfers used for $\mathbf{A}_{I_1,I_1}$ update (3) is the only dominant term in the formula, we choose to implement **LBC** using $b = \sqrt{N}$ as block size. Then:

$$
\begin{aligned}
Q_{\text{LBC}}(N) &\leqslant \frac{N^2}{3\sqrt{S}} + \frac{N^2\sqrt{N}}{2\sqrt{S}} + \frac{N^3}{3\sqrt{2}\sqrt{S}} + \frac{N^2\sqrt{N}}{6} + O(N^2 \log N) \\
&= \frac{N^3}{3\sqrt{2}\sqrt{S}} + O(N^{5/2})
\end{aligned}
$$

Theorem 5.7. *The total communication cost $Q_{LBC}(N)$ of the **LBC** algorithm for a matrix **A** of size $N \times N$, with a memory of size S, is bounded by:*

$$
Q_{LBC}(N) \leqslant \frac{1}{3\sqrt{2}} \cdot \frac{N^3}{\sqrt{S}} + O(N^{\frac{5}{2}}).
$$

6 CONCLUSION

This paper provides a definitive answer to the asymptotic communication complexities of both the SYRK and Cholesky kernels. The perhaps surprising answer is that the symmetric nature of these computations can actually be taken advantage of, so that their operational intensities are intrinsically higher than those of their non-symmetric counterparts (matrix multiplication and LU factorization). In addition to our theoretical lower bound results, our algorithms provide insights about how to make use of the symmetry to reduce communications. In future works, it might be possible to improve the lower order terms of our results, to obtain efficient algorithms for not so large values of N. More importantly, we believe that the insight provided by this paper can be a starting point to obtain communication efficient parallel algorithms for symmetric linear algebra kernels. Finally, our work might also be extended to other kernels which use the same input several times.

ACKNOWLEDGMENTS

This work is supported in part by the Région Nouvelle-Aquitaine, under grant 2018-1R50119 "HPC scalable ecosystem", and by the ANR, under grant SOLHARIS - ANR-19-CE46-0009.

REFERENCES

[1] 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), 1–155. https://doi.org/10.1017/S0962492914000038

[2] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2010. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM Journal on Scientific Computing* 32, 6 (2010), 3495–3523.

[3] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2011. Minimizing Communication in Numerical Linear Algebra. *SIAM J. Matrix Analysis Applications* 32, 3 (2011), 866–901. https://doi.org/10.1137/090769156

[4] Natacha Béreux. 2009. Out-of-Core Implementations of Cholesky Factorization: Loop-Based versus Recursive Algorithms. *SIAM J. Matrix Anal. Appl.* 30, 4 (2009), 1302–1319. https://doi.org/10.1137/06067256X arXiv:https://doi.org/10.1137/06067256X

[5] John Friedlander and Henryk Iwaniec. 2010. *Opera de cribro*. American Mathematical Society Colloquium Publications, Vol. 57. American Mathematical Society, Providence, RI. https://doi.org/10.1090/coll/057

[6] Dror Irony, Sivan Toledo, and Alexander Tiskin. 2004. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel and Distrib. Comput.* 64, 9 (2004), 1017–1026.

[7] Hong Jia-Wei and H. T. Kung. 1981. I/O Complexity: The Red-blue Pebble Game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing* (Milwaukee, Wisconsin, USA) *(STOC '81)*. ACM, New York, NY, USA, 326–333. https://doi.org/10.1145/800076.802486

[8] Grzegorz Kwasniewski, Marko Kabic, Tal Ben-Nun, Alexandros Nikolaos Ziogas, Jens Eirik Saethre, André Gaillard, Timo Schneider, Maciej Besta, Anton Kozhevnikov, Joost VandeVondele, and Torsten Hoefler. 2021. On the Parallel I/O Optimality of Linear Algebra Kernels: Near-Optimal Matrix Factorizations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 70, 15 pages. https://doi.org/10.1145/3458817.3476167

[9] Auguste Olivry, Guillaume Iooss, Nicolas Tollenaere, Atanas Rountev, P Sadayappan, and Fabrice Rastello. 2021. IOOpt: Automatic Derivation of I/O Complexity Bounds for Affine Programs. In *PLDI 2021 - 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. Virtual, Canada. https://doi.org/10.1145/3453483

[10] Auguste Olivry, Julien Langou, Louis-Noël Pouchet, P Sadayappan, and Fabrice Rastello. 2020. Automated derivation of parametric data movement lower bounds for affine programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 808–822.

[11] Edgar Solomonik, Erin Carson, Nicholas Knight, and James Demmel. 2017. Trade-Offs Between Synchronization, Communication, and Computation in Parallel Linear Algebra Computations. *ACM Trans. Parallel Comput.* 3, 1, Article 3 (jan 2017), 47 pages. https://doi.org/10.1145/2897188

[12] Edgar Solomonik and James Demmel. 2011. Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms. 90–109. https://doi.org/10.1007/978-3-642-23397-5_10