Communication-Optimal Parallel and Sequential Cholesky Decomposition

[Extended Abstract] [*]

Grey Ballard[†]
Computer Science Department
University of California Berkeley
CA 94720
ballard@eecs.berkeley.edu

James Demmel
Mathematics Department and CS Division
University of California Berkeley
CA 94720
demmel@cs.berkeley.edu

Olga Holtz[‡]
Departments of Mathematics
University of California, Berkeley
and Technische Universität Berlin.
holtz@math.berkeley.edu

Oded Schwartz[§]
Departments of Mathematics
Technische Universität Berlin
10623 Berlin.
odedsc@math.tu-berlin.de

ABSTRACT

Numerical algorithms have two kinds of costs: arithmetic and communication, by which we mean either moving data between levels of a memory hierarchy (in the sequential case) or over a network connecting processors (in the parallel case). Communication costs often dominate arithmetic costs, so it is of interest to design algorithms minimizing communication. In this paper we first extend known lower bounds on the communication cost (both for bandwidth and for latency) of conventional ($O(n^3)$) matrix multiplication to Cholesky factorization, which is used for solving dense symmetric positive definite linear systems. Second, we compare the cost of various Cholesky decomposition implementations to this lower bound, and draw the following conclusions:

(1) "Naïve" sequential algorithms for Cholesky attain neither the bandwidth nor latency lower bounds.

(2) The sequential blocked algorithm in LAPACK (with the right block size), as well as various recursive algorithms [AP00, GJ01, AGW01, ST04], and one based on work of Toledo [Tol97], can attain the bandwidth lower bound.

(3) The LAPACK algorithm can also attain the latency bound if used with blocked data structures rather than column-wise or row-wise matrix data structures, though the Toledo algorithm cannot.

(4) The recursive sequential algorithm due to [AP00] attains the bandwidth and latency lower bounds at *every level* of a multi-level memory hierarchy, in a "cache-oblivious" way.

(5) The parallel implementation of Cholesky in the ScaLAPACK library (again with the right block-size) attains both the bandwidth and latency lower bounds to within a poly-logarithmic factor.

Combined with prior results in [DGHL08a, DGHL08b, DGX08] this gives a complete set of communication-optimal algorithms for $O(n^3)$ implementations of three basic factorizations of dense linear algebra: LU with pivoting, QR and Cholesky. But it goes beyond this prior work on sequential LU and QR by optimizing communication for any number of levels of memory hierarchy.

*A full version of this paper is available at `http://arxiv.org/abs/0902.2537`.

†Research supported by Microsoft and Intel funding (Award #20080469) and by matching funding by U.C. Discovery (Award #DIG07-10227).

‡O. Holtz acknowledges support of the Sofja Kovalevskaja programm of Alexander von Humboldt Foundation.

§This work was done while visiting University of California, Berkeley.

Categories and Subject Descriptors

F.2.1 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Numerical Algorithms and Problems,Computations on matrices*

General Terms

Algorithms, Performance.

Keywords

Cholesky decomposition, bandwidth, latency, communication avoiding, algorithm, lower bound.

1. INTRODUCTION

Let A be a real symmetric and positive definite matrix. Then there exists a real lower triangular matrix L so that $A = L \cdot L^T$ (L is unique if we restrict its diagonal elements to be positive). This is called the Cholesky decomposition. We are interested in finding efficient parallel and sequential algorithms for the Cholesky decomposition. Efficiency is measured both by the number of arithmetic operations, and by the amount of communication, either between levels of a memory hierarchy on a sequential machine, or between processors communicating over a network on a parallel machine. Since the time to move one word of data typically exceeds the time to perform one arithmetic operation by a large and growing factor, our goal will be to minimize communication.

We model communication costs in more detail as follows. In the sequential case, with two levels of memory hierarchy (fast and slow), communication means reading data items (*words*) from slow memory to fast memory and writing data from fast memory to slow memory. If words are stored contiguously, they can be read or written in a bundle which we will call a *message*. We assume that a message of n words can be communicated between fast and slow memory in time $\alpha + \beta n$ where α is the *latency* (seconds per message) and β is the *inverse bandwidth* (seconds per word). We assume that the matrix being factored initially resides in slow memory, and is too large to fit in the smaller fast memory. Our goal is to minimize the total number of words and the total number of messages communicated between fast and slow memory [1].

In the parallel case, we are interested in the communication among the processors. As in the sequential case, we assume that a message of n consecutively stored words can be communicated in time $\alpha + \beta n$. We assume that the matrix is initially evenly distributed among all P processors, and that there is only enough memory to store about $1/P$-th of a matrix per processor. As before, our goal is to minimize the number of words and messages communicated. In order to measure the communication complexity of a parallel algorithm, we will count the number of words and messages communicated along the critical path of the algorithm.

We consider *classical* algorithms for Cholesky decomposition, i.e., those that perform "the usual" $O(n^3)$ arithmetic operations, possibly reordered by associativity and commutativity of addition. That is, our results do not apply when using distributivity to reorganize the algorithm (such as Strassen-like algorithms); we also assume no pivoting is performed. We define "classical" more carefully later. We show that the communication complexity of any such Cholesky algorithm shares essentially the same lower bound as does the classical matrix multiplication (i.e., using the usual $2n^3$ arithmetic operations possibly reordered using associativity and commutativity of addition).

[1]The sequential communication model used here is sometimes called the *two-level I/O model* or *disk access machine (DAM)* model (see [AV88], [BBF$^+$07], [CR06]). Our model follows that of [HK81] and [ITT04] in that it assumes the block-transfer size is one word of data ($B = 1$ in the common notation).

Theorem 1 (Main Theorem). *Any sequential or parallel classical algorithm for the Cholesky decomposition of n-by-n matrices can be transformed into a classical algorithm for $\frac{n}{3}$-by-$\frac{n}{3}$ matrix multiplication, in such a way that the bandwidth of the matrix multiplication algorithm is at most a constant times the bandwidth of the Cholesky algorithm.*

Therefore any bandwidth lower bound for classical matrix multiplication applies to classical Cholesky, in a Big-O sense: $bandwidth =$

$$\Omega(\#arithmetic\_operations / fast\_memory\_size^{1/2})$$

Similarly, the latency lower bound for Cholesky is: $latency =$

$$\Omega(\#arithmetic\_operations fast\_memory\_size^{3/2})$$

In particular, since a sequential classical n-by-n matrix multiplication algorithm has a bandwidth lower bound of $\Omega(n^3/M^{1/2})$ where M is the fast memory size [HK81, ITT04], classical Cholesky has the same lower bound (we discuss the parallel case later). We always assume that $M = O(n^2)$, as otherwise no communication is needed except reading the entire input once and writing the output once.

To get the latency lower bound, we use the simple observation [DGHL08a] that the number of messages is at least the bandwidth lower bound divided by the maximum message size, and that the maximum message size is at most fast memory size in the sequential case (or the local memory size in the parallel case). So for sequential matrix multiplication this means the latency lower bound is $\Omega(n^3/M^{3/2})$.

Attainability of the latency lower bound depends on the data structure more strongly than does attainability of the bandwidth lower bound. As a simple example, consider computing the sum of $n \le M$ numbers in slow memory, which obviously requires reading these n words. If they are in consecutive memory locations, this can be done in 1 read operation, the minimum possible latency. But if they are not consecutive, say they are separated by at least $M - 1$ words, this may require n read operations, the maximum possible latency.

In the case of matrix multiplication, the well-known blocked matrix multiplication algorithm for $C = A \cdot B$ that multiplies and accumulates $\sqrt{\frac{M}{3}}$-by-$\sqrt{\frac{M}{3}}$ submatrices of A, B and C attains the bandwidth lower bound. But only if each matrix is stored so that the $\frac{M}{3}$ entries of each of its submatrices are contiguous (not the case with columnwise or rowwise storage) can the latency lower bound be reached; we call such a data structure *contiguous block storage* and describe it in more detail below. Alternatively, one could try to copy A and B from their input format (say columnwise) to contiguous block storage doing (asymptotically) no more communication than the subsequent matrix multiplication; we will see this is possible provided $M = \Omega(n)$. There will be analogous requirements for Cholesky to attain its latency lower bound.

In particular, we will draw the following conclusions [2] about the communication costs of sequential classical Cholesky, as summarized in Table 1:

1. "Naïve" sequential variants of Cholesky that operate on single rows and columns (be they left-looking, right-

[2]We number our main conclusions consecutively from 1 to 6.

looking, etc.) attain neither the bandwidth nor the latency lower bound.

2. A sequential blocked algorithm used in LAPACK (with the correct block size) attains the bandwidth lower bound, as do the recursive algorithms in [AP00, GJ01, AGW01, ST04]. A recursive algorithm analogous to Toledo's LU algorithm [Tol97] attains the bandwidth lower bound in nearly all cases, expect possibly for an $O(\log n)$ factor in the narrow range $\frac{n^2}{\log^2 n} < M < n^2$.

3. Whether the LAPACK algorithm also attains the latency lower bound depends on the matrix layout: If the input matrix is given in row-wise or column-wise format, and this is not changed by the algorithm, then the latency lower bound cannot be attained. But if the input matrix is given in contiguous block storage, or $M = \Omega(n)$ so that it can be copied quickly to contiguous block format, then the latency lower bound can be attained by the *LAPACK* algorithm[3]. Toledo's algorithm cannot minimize latency (at least when $M > n^{2/3}$).

So far we have discussed a two-level memory hierarchy, with fast and slow memory. It is natural to ask about more than 2 levels, since most computers have multiple levels (e.g., L1, L2, L3 caches, main memory, and disk). In this case, an optimal algorithm should simultaneously minimize communication between *all* pairs of adjacent levels of memory hierarchy (e.g., minimize bandwidth and latency between L1 and L2, between L2 and L3, etc.).

In the case of sequential matrix multiplication, bandwidth is minimized in this sense by simply applying the usual blocked algorithm recursively, where each level of recursion multiplies matrices that fit at a particular level of the memory hierarchy, by using the blocked algorithm to multiply submatrices that fit in the next smaller level. This is easy since matrix multiplication naturally breaks into smaller matrix multiplications.

For matrix multiplication to minimize latency across all memory hierarchy levels, it is necessary for all submatrices of all sizes to be stored contiguously. This leads to a data structure variously referred to as *recursive block storage* or storage using *space-filling curves*, and described in [FLPR99, AP00, EGJK04].

Finally, sequential matrix multiplication can achieve communication optimality as just described in one of two ways: (1) We can choose the number of recursion levels and sizes of the subblocks with prior knowledge of the number of levels and sizes of the levels of memory hierarchy, a *cache-aware* process called tuning. (2) We can simply always recur down to 1-by-1 blocks (or some other small constant size), repeatedly dividing block sizes by 2 (perhaps padding submatrices to have even dimensions as needed). Such an algorithm is called *cache-oblivious* [FLPR99], and has the advantage of simplicity and portability compared to a cache-aware algorithm, though it might also have more overhead in practice.

[3]This can be done by reading M elements at a time, in a columnwise order (which requires one message) then writing each of these elements to the right location of the new matrix. We write these words using $\sqrt{M}$ messages (one per each relevant block). Thus, the total number of messages is $O\left(\frac{n^2}{\sqrt{M}}\right)$ which is asymptotically dominated by $\frac{n^3}{M^{3/2}}$ for $M \geq n$.

It is indeed possible for sequential Cholesky to be organized to be optimal across multiple memory hierarchy levels in all the senses just described, assuming we use recursive block storage:

4. The recursive algorithm modelled on Toledo's LU can be implemented in a cache-oblivious way so as to minimize bandwidth, but not latency [4].

5. The cache-oblivious recursive Cholesky algorithm of Ahmed and Pingali [AP00] minimizes both bandwidth and latency for all matrices across all memory hierarchy levels. None of the other algorithms do so.

Finally, we address the case of parallel Cholesky, where there are P processors connected by a network with latency α and reciprocal bandwidth β. We consider only the memory-scalable case, where each processor's local memory is of size $M = O(n^2/P)$, so that only $O(1)$ copies of the matrix are stored overall (the so-called "2D case", see [ITT04] for the general case, including 3D, for matrix multiplication). The consequence of our Main Theorem is again a bandwidth lower bound of the form $\Omega(n^3/(PM^{1/2})) = \Omega(n^2/P^{1/2})$, and a latency lower bound of the form $\Omega(n^3/(PM^{3/2})) = \Omega(P^{1/2})$.

ScaLAPACK attains a matching upper bound. It does so by partitioning the matrix into submatrices and distributing them to the processors in a block cyclic manner. See full version of this paper for details [BDHS09].

6. With the right choice of block size b, namely the largest possible value $b = n/\sqrt{P}$, the Cholesky algorithm in ScaLAPACK [BJCD+97] attains the above bandwidth and latency lower bounds to within a factor of $\log P$. This is summarized in Table 2.

A 'real' computer may be more complicated than any model we have discussed so far, with both parallelism and multiple levels of memory hierarchy (where each sequential processor making up a parallel computer has multiple levels of cache), or with multiple levels of parallelism (i.e., where each 'parallel processor' itself consists of multiple processors), etc. And it may be 'heterogenous', with functional units and communication channels of greatly differing speeds. We leave lower and upper communication bounds on such processors for future work.

The rest of this paper is organized as follows. In Section 2 we show the reduction from matrix multiplication to Cholesky decomposition, thus extending the bandwidth lower bounds of [HK81] and [ITT04] to a bandwidth lower bound for the sequential and parallel implementations of Cholesky decomposition. We also discuss latency lower bounds. In the full version of this paper (see [BDHS09]) we recall known Cholesky decomposition algorithms and compare their bandwidth and latency costs with the lower bounds (see also [B$\acute{0}$8]).

[4]Toledo's algorithm is designed to retain numerical stability for the LU case. The [AP00] algorithm deals with the Cholesky case, therefore requires no pivoting for numerical stability. Thus a simpler recursion suffices, and the latency improves.

	Bandwidth	Latency	Cache Oblivious
Lower bound	$\Omega\left(\frac{n^3}{\sqrt{M}}\right)$	$\Omega\left(\frac{n^3}{M^{3/2}}\right)$	
Naïve: left/right looking			
Column-major	$\Theta(n^3)$	$\Theta\left(n^2+\frac{n^3}{M}\right)$	✓
LAPACK [ABB$^+$92]			
Column-major	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M}\right)$	$\times$
Contiguous blocks	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M^{3/2}}\right)$	$\times$
Rectangular Recursive [Tol97]			
Column-major	$\Theta\left(\frac{n^3}{\sqrt{M}}+n^2\log n\right)$	$\Omega\left(\frac{n^3}{M}\right)$	✓
Contiguous blocks	$\Theta\left(\frac{n^3}{\sqrt{M}}+n^2\log n\right)$	$\Omega\left(n^2\right)$	✓
Square Recursive [AP00]			
"Recursive Packed Format" [AGW01]	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$\Omega\left(\frac{n^3}{M}\right)$	✓
Column-major [AP00]	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M}\right)$	✓
Contiguous blocks [AP00]	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M^{3/2}}\right)$	✓

Table 1: Sequential bandwidth and latency: lower bound vs. algorithms. M denotes the size of the fast memory. We assume $n^2 > M$ in this table. FLOPs count of all is $O(n^3)$.
Further details of these algorithms can be found in the full version of this paper [BDHS09].

	Bandwidth	Latency	FLOPS
Lower-bound			
General	$\Omega\left(\frac{n^3}{P\sqrt{M}}\right)$	$\Omega\left(\frac{n^3}{PM^{3/2}}\right)$	$\Omega\left(\frac{n^3}{P}\right)$
2D layout: $M=O\left(\frac{n^2}{P}\right)$	$\Omega\left(\frac{n^2}{\sqrt{P}}\right)$	$\Omega\left(\sqrt{P}\right)$	$\Omega\left(\frac{n^3}{P}\right)$
ScaLAPACK [BJCD$^+$97]			
General	$O\left(\left(\frac{n^2}{\sqrt{P}}+nb\right)\log P\right)$	$O\left(\frac{n}{b}\log P\right)$	$O\left(\frac{n^3}{P}+\frac{n^2b}{\sqrt{P}}+nb^2\right)$
Choosing $b=\frac{n}{\sqrt{P}}$	$O\left(\frac{n^2}{\sqrt{P}}\log P\right)$	$O(\sqrt{P}\log P)$	$O\left(\frac{n^3}{P}\right)$

Table 2: Parallel, lower bound vs. algorithms. M denotes the size of the memory of each processor. P is the number of processors. b is the block size.
Further details of these algorithms can be found in the full version of this paper [BDHS09].

2. COMMUNICATION LOWER BOUNDS

Consider an algortihm for a parallel computer with P processors that multiplies matrices in the 'classical' way (the usual $2n^3$ arithmetic operations possibly reordered using associativity and commutativity of addition) and each of the processors has memory of size M. Irony *et al.* [ITT04] showed that at least one of the processors has to send or receive this minimal number of words:

THEOREM 2 ([ITT04]). *Any 'classical' implementation of matrix-multiplication of $n \times n$ matrices on a P processor machine, each equipped with memory of size M, requires that one of the processors sends or receives at least $\frac{n^3}{2\sqrt{2}PM^{\frac{1}{2}}} - M$ words. These can be entries of A, of B or of $A \cdot B$.*

If A and B are of size $n \times m$ and $m \times r$ respectively, then the corresponding bound is $\frac{nmr}{2\sqrt{2}PM^{\frac{1}{2}}} - M$

As any processor has memory of size M, any message it sends or receives may deliver at most M words. Therefore we deduce the following:

COROLLARY 2.1. *Any 'classical' implementation of matrix-multiplication on a P processor machine, each processor equipped with memory of size M, requires that one of the processors sends or receives at least $\frac{n^3}{2\sqrt{2}PM^{\frac{3}{2}}} - 1$ messages.*

If A and B are of size $n \times m$ and $m \times r$ respectively, then the corresponding bound is $\frac{nmr}{2\sqrt{2}PM^{\frac{3}{2}}} - 1$

For the case of $P = 1$ these give lower bounds for the bandwidth and the latency of the sequential case. These lower bounds for bandwidth for the sequential case were previously shown (up to some multiplicative constant factor) by Hong and Kung [HK81].

It is easy to reduce matrix multiplication to LU decomposition of a slightly larger order, as the following identity shows:

$$\begin{pmatrix} I & 0 & -B \\ A & I & 0 \\ 0 & 0 & I \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ 0 & 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 & -B \\ & I & A \cdot B \\ & & I \end{pmatrix} \qquad (1)$$

This identity means that LU factorization can be used to perform matrix multiplication; to accomodate pivoting A and/or B can be scaled down to be too small to be chosen as pivots, and $A \cdot B$ can be scaled up accordingly. Thus an $O(n^3)$ implementation of LU that only uses associativity and commutativity of addition to reorganize its operations (thus eliminating Strassen-like algorithms) must perform at least as much communication as a correspondingly reorganized implementation of $O(n^3)$ matrix multiplication.

We wish to mimic this lower bound construction for Cholesky. Consider the following reduction from matrix multiplication to Cholesky decomposition. Let T be the matrix defined below, composed of 9 square blocks each of dimension n; then the Cholesky decomposition of T is:

$$\begin{aligned} T &\equiv \begin{pmatrix} I & A^T & -B \\ A & I + A \cdot A^T & 0 \\ -B^T & 0 & D \end{pmatrix} \qquad (2) \\ &= \begin{pmatrix} I & & \\ A & I & \\ -B^T & (A \cdot B)^T & X \end{pmatrix} \cdot \begin{pmatrix} I & A^T & -B \\ & I & A \cdot B \\ & & X^T \end{pmatrix} \\ &\equiv L \cdot L^T \end{aligned}$$

where X is the Cholesky factor of $D' \equiv D - B^T B - B^T A^T AB$, and D can be any symmetric matrix such that D' is positive definite.

Thus $A \cdot B$ is computed via this Cholesky decomposition. Intuitively this seems to show that the communication complexity needed for computing matrix multiplication is a lower bound to that of computing the Cholesky decomposition (of matrices 3 times larger) as $A \cdot B$ appears in L^T, the decomposition of T. Note however that $A \cdot A^T$ appears in T.

One has to consider the possibility that all the communication that is guaranteed by [ITT04] is in fact performed when computing $A \cdot A^T$ and so we have no non-trivial lower bound for the Cholesky decomposition of T.[5] Stated otherwise, maybe computing $A \cdot B$ from A and B incurs less communication cost if we are also given $A \cdot A^T$.[6] So let us instead consider the following approach to prove the lower bound.

In addition to the real numbers $\mathbb{R}$, consider new "starred" numerical quantities, called 1^* and 0^*, with arithmetic properties detailed in the following tables. 1^* and 0^* mask any real value in addition/substraction operation, but behave similarly to $1 \in \mathbb{R}$ and $0 \in \mathbb{R}$ in multiplication and division operations.

Consider this set of values and arithmetic operations. It is commutative with respect to addition and to multiplication (by the symmetries of the corresponding tables). It is associative with respect to addition: regardless of ordering of summation, the sum is 1^* if one of the addends is 1^*, otherwise it is 0^* if one of the addends is 0^*. The set is also associative with respect to multiplication: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. This is trivial if all factors are in $\mathbb{R}$. As 1^* is a multiplicative identity, it is also immediate if some of the factors equal 1^*. Otherwise, at least one of the factors is 0^*, and the product is 0. Distributivity, however, does not hold: $1 \cdot (1^* + 1^*) = 1 \neq 2 = (1 \cdot 1^*) + (1 \cdot 1^*)$.

Let us return to the construction. We set T' to be:

$$T' \equiv \begin{pmatrix} I & A^T & -B \\ A & C & 0 \\ -B^T & 0 & C \end{pmatrix}$$

where C has 1^* on the diagonal and 0^* everywhere else:

$$C \equiv \begin{pmatrix} 1^* & 0^* & & \cdots & 0^* \\ 0^* & 1^* & 0^* & & \vdots \\ & & & \ddots & \\ \vdots & & & & 0^* \\ 0^* & \cdots & & 0^* & 1^* \end{pmatrix}$$

One can verify that the (unique) Cholesky decomposition of C is[7]

[5]Note that computing $A \cdot A^T$ is asymptotically as hard as matrix multiplication: take $A = [X, 0; Y^T, 0]$. Then $A \cdot A^T = [*, XY; *, *]$

[6]Note that the result of [ITT04] does not mean that both A and B are communicated a lot, as one can communicate each of the entries of B only once, and shift all other entries many times, resulting in an inefficient algorithm, but such that no non-trivial lower bound on the communication of the elements of B can be given.

[7]By writing $X \cdot Y$ we mean the resulting matrix assuming the straightforward n^3 matrix multiplication algorithm. This

$\pm$	1^*	0^*	y
1^*	1^*	1^*	1^*
0^*	1^*	0^*	0^*
x	1^*	0^*	$x \pm y$

$\cdot$	1^*	0^*	y
1^*	1^*	0^*	y
0^*	0^*	0	0
x	x	0	$x \cdot y$

$/$	1^*	0^*	$y \neq 0$
1^*	1^*	$-$	$1/y$
0^*	0^*	$-$	0
x	x	$-$	x/y

$\sqrt{\cdot}$	
1^*	1^*
0^*	0^*
$x \geq 0$	$\sqrt{x}$

Table 3: Arithmetic Operations Tables. The variables x and y stand for any *real* values. For consistency, $-0^* \equiv 0^*$ and $-1^* \equiv 1^*$.

$$C = \begin{pmatrix} 1^* & 0 & \dots & 0 \\ 0^* & 1^* & & \vdots \\ \vdots & & \ddots & 0 \\ 0^* & \cdots & 0^* & 1^* \end{pmatrix} \cdot \begin{pmatrix} 1^* & 0^* & \cdots & 0^* \\ & \ddots & \ddots & \vdots \\ \vdots & & 1^* & 0^* \\ 0 & \dots & & 1^* \end{pmatrix} \equiv C' \cdot C'^T \tag{3}$$

Note that if a matrix X does not contain any "starred" values 0^* and 1^* then $X = C \cdot X = X \cdot C = C' \cdot X = X \cdot C' = C'^T \cdot X = X \cdot C'^T$ and $C + X = C$. Therefore, one can confirm that the Cholesky decomposition of T' is:

$$\begin{aligned} T' &\equiv \begin{pmatrix} I & A^T & -B \\ A & C & 0 \\ -B^T & 0 & C \end{pmatrix} \quad (4) \\ &= \begin{pmatrix} I & & \\ A & C' & \\ -B^T & (A \cdot B)^T & C' \end{pmatrix} \cdot \begin{pmatrix} I & A^T & -B \\ & C'^T & A \cdot B \\ & & C'^T \end{pmatrix} \\ &\equiv L \cdot L^T \end{aligned}$$

One can think of C as masking the $A \cdot A^T$ previously appearing in the central block of T, therefore allowing the lower bound of computing $A \cdot B$ to be accounted for by the Cholesky decomposition, and not by the computation of $A \cdot A^T$. More formally, let *Alg* be any 'classical' algorithm for Cholesky factorization. We convert it to a matrix multiplication algorithm as follows:

Algorithm 1 Matrix Multiplication by Cholesky-Decomposition

```
Input: Two $n \times n$ matrices, $A$ and $B$.
1: Let $Alg'$ be $Alg$ updated to correctly handle the new $0^*, 1^*$ values. {note that $Alg'$ can be constructed off-line.}
2: $T' = T'(A, B)$ {constructed as in Equation (4).}
3: $L = Alg'(T')$
4: return $(L_{32})^T$
```

The simplest conceptual way to do step (1) is to attach an extra bit to every numerical value, indicating whether it is "starred" or not, and modify every arithmetic operation to first check this bit before performing an operation. This increases the bandwidth by at most a constant factor. Alternatively, we can use Signalling NaNs as defined in the IEEE Floating Point Standard [IEE08] to encode 1^* and 0^* with no extra bits.

If the instructions implementing Cholesky are scheduled deterministically, there is another alternative: one can run the algorithm "symbolically", propagating 0^* and 1^* arguments from the inputs forward, simplifying or eliminating arithmetic operations whose inputs contain 0^* or 1^*, and also eliminating operations for which there is no path in the directed acyclic graph (describing how outputs of each operation propagate to inputs of other operations) to the desired output $A \cdot B$. The resulting Alg' performs a strict subset of the arithmetic and memory operations of the original Cholesky algorithm.

had to be stated clearly, as the distributivity does not hold for the starred values.

We note that updating *Alg* to form Alg' is done off-line, so that step (1) does not actually take any time to perform when Algorithm 1 is called.

We next verify the correctness of this reduction: that the output of this procedure on input A, B is indeed the multiplication $A \cdot B$, as long as *Alg* is a classical algorithm, in a sense we now define carefully.

Let $T' = L \cdot L^T$ be the Cholesky decomposition of T'. Then we have the following formulas:

$$L(i,i) = \sqrt{T'(i,i) - \sum_{k \in [i-1]} (L(i,k))^2} \tag{5}$$

$$L(i,j) = \frac{1}{L(j,j)} \left(T'(i,j) - \sum_{k \in [j-1]} L(i,k) \cdot L(j,k) \right), \quad i > j \tag{6}$$

where $[t] = \{1, ..., t\}$. By the no-pivoting and no-distributivity restrictions to *Alg*, when an entry of L is computed, all the entries on which it depends have already been computed and combined by the above formulas, with the sums occurring in any order. These dependencies form a dependency graph which is a DAG (directed acyclic graph), and therefore impose a partial ordering on the computation of the entries of L (see Figure 1). That is, when an entry $L(i,i)$ is computed, by Equation (5), all the entries $\{L(i,k)\}_{k \in [i-1]}$ have already been computed. Denote this set by $S_{i,i}$, namely,

$$S_{i,i} \equiv \{L(i,k)\}_{k \in [i-1]} \tag{7}$$

Similarly, when an entry $L(i,j)$ (for $i > j$) is computed, by Equation (6), all the entries $\{L(i,k)\}_{k \in [j-1]}$ and all the entries $\{L(j,k)\}_{k \in [j]}$ have already been computed. Denote this set by $S_{i,j}$ namely,

$$S_{i,j} \equiv \{L(i,k)\}_{k \in [j-1]} \cup \{L(j,k)\}_{k \in [j]} \tag{8}$$

LEMMA 2.2. *Any ordering of the computation of the elements of L that respects the partial ordering induced by the above mentioned directed acyclic graph results in a correct computation of $A \cdot B$.*

PROOF. We need to confirm that the starred entries 1^* and 0^* of T' do not somehow "contaminate" the desired entries of L_{32}^T. The proof is by induction on the partial order on pairs (i,j) implied by (7) and (8). The base case —the correctness of computing $L(1,1)$— is immediate. Assume by

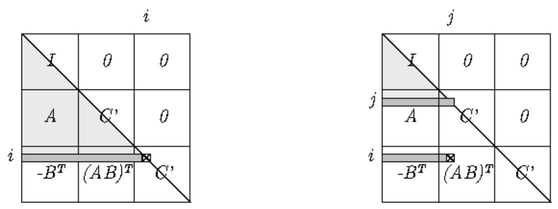

Figure 1: Dependencies of $L(i,j)$, for diagonal entries (left) and other entries (right).
Dark grey represents the sets $S_{i,i}$ (left) and $S_{i,j}$ (right). Light grey represents indirect dependencies.

induction that all elements of $S_{i,j}$ are correctly computed and consider the computation of $L(i,j)$ according to the block in which it resides:

- If $L(i,j)$ resides in block L_{11}, L_{21} or L_{31} then $S_{i,j}$ contains only real values, and no arithmetic operations with 0^* or 1^* occur (recall Figure 1 or Equations (4),(7) and (8)). Therefore, the correctness follows from the correctness of the original Cholesky decomposition algorithm.

- If $L(i,j)$ resides in L_{22} or L_{33} then $S_{i,j}$ may contain "starred" value (elements of C'). We treat separately the case where $L(i,j)$ is on the diagonal and the case where it is not.

  If $i=j$ then by Equation (5) $L(i,i)$ is determined to be 1^* since $T'(i,i)=1^*$ and since adding to, subtracting from and taking the square root of 1^* all result in 1^* (recall Table 3 and Equation (5)).

  If $i>j$ then by the inductive assumption the divisor $L(j,j)$ of Equation (6) is correctly computed to be 1^* (recall Figure 1 and the definition of C' in Equation (3)). Therefore, no division by 0^* is performed. Moreover, $T'(i,j)$ is 0^*. Then $L(i,j)$ is determined to be the correct value 0^*, unless 1^* is subtracted (recall Equation (6)). However, every subtracted product (recall Equation (6)) is composed of two factors of the same column but of different rows. Therefore, by the structure of C', none of them is 1^* so their product is not 1^* and the value is computed correctly.

- If $L(i,j)$ resides in L_{32} then $S_{i,j}$ may contain "starred" values (see Figure 1, right-hand side, row j). However, every subtraction performed (recall Equation (6)) is composed of a product of two factors, of which one is on the ith row (and on a column $k<j$). Hence, by induction (on i,j), the (i,k) element has been computed correctly to be a real value, and by the multiplication properties so is the product. Therefore no masking occurs.

This completes the proof of Lemma 2.2. □

We now know that Algorithm 1 correctly multiplies matrices 'classically', and so has known communication lower bounds given by Theorem 2 and Corollary 2.1. But it remains to confirm that step 2 (setting up T') and step 4 (returning L_{32}^T) do not require much communication, so that these lower bounds apply to step 3, running Alg' (recall that step 1 may be performed off-line and so doesn't count). Since Alg' is either a small modification of Cholesky to add "star" labels to all data items (at most doubling the bandwidth), or a subset of Cholesky with some operations omitted (those with starred arguments, or not leading to the desired output L_{32}), a lower bound on communication for Alg' is also a lower bound for Cholesky.

Theorem 1 (Main Theorem). *Any sequential or parallel classical algorithm for the Cholesky decomposition of n-by-n matrices can be transformed into a classical algorithm for $\frac{n}{3}$-by-$\frac{n}{3}$ matrix-multiplication, in such a way that the bandwidth of the matrix-multiplication algorithm is at most a constant times the bandwidth of the Cholesky algorithm.*

Therefore any bandwidth or latency lower bound for classical matrix multiplication applies to classical Cholesky in a Big-O sense:

COROLLARY 2.3. *In the sequential case, with a fast memory of size M, the bandwidth lower bound for Cholesky decomposition is $\Omega(n^3/M^{1/2})$, and the latency lower bound is $\Omega(n^3/M^{3/2})$.*

PROOF. Constructing T' (in any data format) requires bandwidth of at most $18n^2$ (copying a $3n$-by-$3n$ matrix, with another factor of 2 if each entry has a flag indicating whether it is "starred" or not), and extracting L_{32}^T requires another n^2 of bandwidth. Furthermore, we can assume $n^2 < n^3/M^{1/2}$, i.e., that $M < n^2$, i.e., that the matrix is too large to fit entirely in fast memory (the only case of interest). Thus the bandwidth lower bound $\Omega(n^3/M^{1/2})$ of Algorithm 1 dominates the bandwidth costs of Steps 2 and 4, and so must apply to Step 3 (Cholesky). Finally, the latency lower bound for Step 3 is by a factor of M smaller than its bandwidth lower bound, as desired. □

COROLLARY 2.4. *In the parallel case (with a 2D layout on P processors as described earlier), the bandwidth lower bound for Cholesky decomposition is $\Omega(n^2/P^{1/2})$, and the latency lower bound is $\Omega(P^{1/2})$.*

PROOF. The argument in the parallel case is analogous to that of Corollary 2.3. The construction of input and retrieval output at steps 2 and 4 of Algorithm 1 contribute bandwidth of $O\left(\frac{n^2}{P}\right)$. Therefore the lower bound of the bandwidth $\Omega\left(\frac{n^3}{P\sqrt{M}}\right)$ is determined by Step 3, the Cholesky decomposition. The lower bound on the latency of Step 3 is therefore $\Omega\left(\frac{n^3}{PM^{3/2}}\right)$, as each message delivers at most M words. Plugging in $M=O\left(\frac{n^2}{P}\right)$ yields $B=\Omega(P^{1/2})$. □

3. REFERENCES

[ABB+92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK's user's guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. Also available from http://www.netlib.org/lapack/.

[AGW01] B. S. Andersen, F. G. Gustavson, and J. Wasniewski. A recursive formulation of Cholesky factorization of a matrix in packed storage format. *ACM Transactions on Mathematical Software*, 27(2):214–244, jun 2001.

[AP00] N. Ahmed and K. Pingali. Automatic generation of block-recursive codes. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 368–378, London, UK, 2000. Springer-Verlag.

[AV88] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.

[B0́8] N. Béreux. Out-of-core implementations of cholesky factorization: Loop-based versus recursive algorithms. *SIAM Journal on Matrix Analysis and Applications*, 30(4):1302–1319, 2008.

[BBF$^+$07] M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob, and E. Vicari. Optimal sparse matrix dense vector multiplication in the I/O-model. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 61–70, New York, NY, USA, 2007. ACM.

[BDHS09] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Communication-optimal parallel and sequential Cholesky decomposition, 2009. Available from http://arxiv.org/abs/0902.2537.

[BJCD$^+$97] L. S. Blackford, A. Cleary J. Choi, E. Dï£¡Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, USA, May 1997. Also available from http://www.netlib.org/scalapack/.

[CR06] R. A. Chowdhury and V. Ramachandran. Cache-oblivious dynamic programming. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 591–600, New York, NY, USA, 2006. ACM.

[DGHL08a] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. UC Berkeley Technical Report EECS-2008-89, Aug 1, 2008; Submitted to SIAM. J. Sci. Comp., 2008.

[DGHL08b] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Implementing communication-optimal parallel and sequential QR and LU factorizations. submitted to SIAM. J. Sci. Comp., 2008.

[DGX08] J. Demmel, L. Grigori, and H. Xiang. Communication-avoiding Gaussian elimination. Supercomputing 08, 2008.

[EGJK04] E. Elmroth, F. G. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. 46(1):3–45, March 2004.

[FLPR99] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 285, Washington, DC, USA, 1999. IEEE Computer Society.

[GJ01] F. G. Gustavson and I. Jonsson. High performance Cholesky factorization via blocking and recursion that uses minimal storage. In *PARA '00: Proceedings of the 5th International Workshop on Applied Parallel Computing, New Paradigms for HPC in Industry and Academia*, pages 82–91, London, UK, 2001. Springer-Verlag.

[HK81] J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333, New York, NY, USA, 1981. ACM.

[IEE08] IEEE standard for floating-point arithmetic. *IEEE Std. 754-2008*, pages 1–58, 29 2008.

[ITT04] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[ST04] I. Simecek and P. Tvrdik. Analytical model for analysis of cache behavior during Cholesky factorization and its variants. In *ICPPW '04: Proceedings of the 2004 International Conference on Parallel Processing Workshops*, pages 190–197, Washington, DC, USA, 2004. IEEE Computer Society.

[Tol97] S. Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 18(4):1065–1081, 1997.