SIAM J. Sci. Comput.
Vol. 32, No. 6, pp. 3495–3523

COMMUNICATION-OPTIMAL PARALLEL AND SEQUENTIAL CHOLESKY DECOMPOSITION[*]

GREY BALLARD[†], JAMES DEMMEL[‡], OLGA HOLTZ[§], AND ODED SCHWARTZ[¶]

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 costs of various Cholesky decomposition implementations to these lower bounds and identify the algorithms and data structures that attain them. In the sequential case, we consider both the two-level and hierarchical memory models. Combined with prior results in [J. Demmel et al., *Communication-optimal Parallel and Sequential QR and LU Factorizations*, Technical report EECS-2008-89, University of California, Berkeley, CA, 2008], [J. Demmel et al., *Implementing Communication-optimal Parallel and Sequential QR and LU Factorizations*, SIAM. J. Sci. Comp., submitted], and [J. Demmel, L. Grigori, and H. Xiang, *Communication-avoiding Gaussian Elimination*, Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008] this gives a set of communication-optimal algorithms for $O(n^3)$ implementations of the three basic factorizations of dense linear algebra: LU with pivoting, QR, and Cholesky. But it goes beyond this prior work on sequential LU by optimizing communication for any number of levels of memory hierarchy.

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

AMS subject classifications. 68Q25, 68W10, 68W15, 68W40, 65Y05, 65Y10, 65Y20, 65F30

DOI. 10.1137/090760969

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

[*]Received by the editors June 2, 2009; accepted for publication (in revised form) September 21, 2010; published electronically December 2, 2010. A preliminary version of this paper was accepted to the 21*st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'*09*)*, 2009, pp. 245–252.

http://www.siam.org/journals/sisc/32-6/76096.html

[†]Computer Science Department, University of California, Berkeley, CA 94720 (ballard@eecs.berkeley.edu). Research supported by Microsoft and Intel funding (award 20080469) and by matching funding by U.C. Discovery (award DIG07-10227).

[‡]Mathematics Department and CS Division, University of California, Berkeley, CA 94720 (demmel@cs.berkeley.edu). Research supported by Microsoft and Intel funding (award 20080469) and by matching funding by U.C. Discovery (award DIG07-10227).

[§]Departments of Mathematics, University of California, Berkeley and Technische Universität Berlin (holtz@math.berkeley.edu). The third author acknowledges support of the Sofja Kovalevskaja programme of Alexander von Humboldt Foundation.

[¶]The Weizmann Institute of Science, Rehovot 76100, Israel (oded.schwartz@weizmann.ac.il). This work was done while at the Department of Mathematics, Technische Universität Berlin, and while visiting the University of California, Berkeley.

the time to perform one arithmetic operation by a large and growing factor [18], our goal will be to minimize communication.

1.1. Communication model. 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. Words that are contiguous 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 define the *bandwidth cost* of an algorithm to be the total number of words communicated and the *latency cost* of an algorithm to be the total number of messages communicated. 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 then 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 $\alpha + \beta n$. This cost includes the time required to "pack" noncontiguous words into a single message if necessary. 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 cost of a parallel algorithm, we count the number of words and messages communicated along the critical path as defined in [30], as this metric is closely related to the total running time of the algorithm.

We assume that (1) computation and communication are not overlapped (all communication is "blocking"), (2) the cost per flop is the same on each processor, and the communication costs (α and β) are the same between each pair of processors, and (3) there is no communication resource contention among processors. For example, if processor 0 sends a message of size n to processor 1 at time 0 and processor 2 sends a message of size n to processor 3 also at time 0, the cost along the critical path is $\alpha + \beta n$. However, if both processors 0 and 1 try to send a message to processor 2 at the same time, the cost along the critical path will be the sum of the costs of each message.

1.2. Communication lower bounds. 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.

THEOREM 1.1 (main theorem). *Any sequential or parallel classical algorithm for the Cholesky decomposition of n-by-n matrices can be transformed into a classical*

[1]The sequential communication model used here is sometimes called the *two-level input/output (I/O) model* or *disk access machine (DAM)* model (see [2, 9, 12]). Our bandwidth cost model follows that of [23] and [25] in that it assumes the block-transfer size is one word of data ($B = 1$ in the common notation). However, our model allows message sizes to vary from one word up to the maximum number of words that can fit in fast memory.

algorithm for $\frac{n}{3}$-by-$\frac{n}{3}$ matrix multiplication in such a way that the bandwidth cost of the matrix multiplication algorithm is at most a constant times the bandwidth cost of the Cholesky algorithm.

Therefore, any bandwidth cost lower bound for classical matrix multiplication applies to classical Cholesky decomposition, asymptotically. In particular, since a sequential classical n-by-n matrix multiplication algorithm has a bandwidth cost lower bound of $\Omega(n^3/M^{1/2})$, where M is the fast memory size [23, 25], classical Cholesky decomposition has the same lower bound (we discuss the parallel case later).

To get a latency cost lower bound, we use the simple observation [13] that the number of messages is at least the bandwidth cost 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 cost lower bound is $\Omega(n^3/M^{3/2})$.

1.3. Communication upper bounds.

1.3.1. Two-level memory model. In the case of matrix multiplication on a sequential machine, a well-known algorithm attains the bandwidth cost lower bound [23]. It computes $C = A \cdot B$ by performing blocked matrix multiplication that multiplies and accumulates $\sqrt{\frac{M}{3}}$-by-$\sqrt{\frac{M}{3}}$ submatrices of A, B, and C. 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), the latency cost lower bound is reached; we call such a data structure *block-contiguous 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 cost lower bound.

In particular, we draw the following conclusions about the communication costs of sequential classical Cholesky decomposition in the two-level memory model, as summarized in Table 1.1 and what follows:

1. "Naïve" sequential variants of Cholesky that operate on single rows and columns (be they left-looking, right-looking, etc.) attain neither the bandwidth nor the latency cost lower bound.
2. A sequential blocked algorithm used in *LAPACK* (with the correct block size) attains the bandwidth cost lower bound, as do the recursive algorithms in [3, 4, 20, 21, 27]. A recursive algorithm analogous to Toledo's LU algorithm [28] attains the bandwidth cost lower bound in nearly all cases, except 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 cost lower bound depends on the matrix layout: If the input matrix is given in rowwise or columnwise format and this is not changed by the algorithm, then the latency cost 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 cost lower bound can be attained by the *LAPACK* algorithm. Toledo's algorithm cannot minimize latency (at least when $M > n^{2/3}$); see also conclusion 5 below.

1.3.2. Hierarchical memory model. Since most computers have multiple levels (e.g., L1, L2, L3 caches, main memory, and disk), we also consider the communication costs in the hierarchical memory model. In this case, an optimal algorithm

TABLE 1.1
Sequential bandwidth and latency costs: 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 $O(n^3)$. Refer to section 3 for definitions of terms and details.

	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 [5]			
Column-major	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M}\right)$	$\times$
Block-contiguous	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M^{3/2}}\right)$	$\times$
Rectangular recursive [28]			
Column-major	$\Theta\left(\frac{n^3}{\sqrt{M}}+n^2\log n\right)$	$\Omega\left(\frac{n^3}{M}\right)$	✓
Block-contiguous	$\Theta\left(\frac{n^3}{\sqrt{M}}+n^2\log n\right)$	$\Omega\left(n^2\right)$	✓
Square recursive [3, 4, 20, 21, 27]			
"Recursive packed format" [4]	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$\Omega\left(\frac{n^3}{M}\right)$	✓
Column-major [20]	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M}\right)$	✓
Block-contiguous [3]	$O\left(\frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^3}{M^{3/2}}\right)$	✓

should simultaneously minimize communication between *all* pairs of adjacent levels of memory hierarchy (e.g., minimize bandwidth and latency costs between L1 and L2, between L2 and L3, etc.).

In the case of sequential matrix multiplication, bandwidth cost 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*, *Morton ordering*, or storage using *space-filling curves* and is described in [3, 6, 16, 17, 29].

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 carry out the recursion 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* [17] and has the advantage of simplicity and portability compared to a cache-aware algorithm, though it might also have more overhead in practice.

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:

TABLE 1.2
Parallel bandwidth and latency costs: 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. Refer to section 3 for definitions of terms and details.

	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 [11]			
General	$O\left(\left(\frac{n^2}{\sqrt{P}} + nb\right)\log P\right)$	$O\left(\frac{n}{b}\log P\right)$	
Choosing $b = \Theta\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n^2}{\sqrt{P}}\log P\right)$	$O(\sqrt{P}\log P)$	$O\left(\frac{n^3}{P}\right)$

4. The recursive algorithm modeled on Toledo's LU can be implemented in a cache-oblivious way so as to minimize bandwidth cost, but not the latency cost.[2]
5. The cache-oblivious recursive Cholesky algorithm which first appeared in [20] (and can also be found in [3, 4, 21, 27]) minimizes both bandwidth and latency costs (assuming recursive block storage) for all matrices across all memory hierarchy levels. None of the other algorithms have this property.

1.3.3. Parallel model.

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 here 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 two-dimensional (2D) case). See [25] for the general lower bound, that holds for three-dimensional (3D) algorithms, for matrix multiplication. A 3D algorithm for LU decomposition without pivoting (along with an algorithm for triangular system solution) is presented with communication analysis in [24]. While the algorithms minimize the total volume of communication, they do not minimize the communication costs along the critical path as measured in this paper.

For the 2D case, the consequence of our main theorem is again a bandwidth cost lower bound of the form $\Omega(n^3/(PM^{1/2})) = \Omega(n^2/P^{1/2})$, and a latency cost lower bound of the form $\Omega(n^3/(PM^{3/2})) = \Omega(P^{1/2})$.

6. The Cholesky algorithm in *ScaLAPACK* [11] 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. With the right choice of block size b, namely, $b = \Theta(\frac{n}{\sqrt{P}})$, it attains the above bandwidth and latency cost lower bounds to within a factor of $\log P$. This is summarized in Table 1.2. See section 3.3.1 for details.

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 cost lower bounds of [23] and [25] to a bandwidth cost lower bound for the sequential and parallel implementations of Cholesky decomposition. We also discuss latency cost lower bounds. In section 3 we present known Cholesky decomposition algorithms and compare their bandwidth and latency costs with the lower bounds.

[2]Toledo's algorithm is designed to retain numerical stability for the LU case. The algorithm of [20] deals with the Cholesky case only and therefore requires no pivoting for numerical stability. Thus a simpler recursion suffices, and the latency cost diminishes.

A shorter version of this paper (an extended abstract) appeared in the proceedings of the SPAA 2009 conference; the shorter version includes the reduction proof of the lower bounds and Tables 1.1 and 1.2. This paper proves the claims made in the tables by presenting the algorithms and their analyses in section 3.

2. Communication lower bounds.

Consider an algorithm for a parallel computer with P processors that multiplies matrices in the "classical" way (the usual $O(n^3)$ arithmetic operations possibly reordered using associativity and commutativity of addition), and each of the processors has memory of size M. Irony, Toledo, and Tiskin [25] showed that at least one of the processors has to send or receive this minimal number of words.

THEOREM 2.1 (see [25]). *Any "classical" implementation of matrix multiplication of $n\times n$ matrices A and B 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.*

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 2.2. *Any "classical" implementation of matrix multiplication of $n\times n$ matrices A and B on a P processor machine, each equipped with memory of size M, requires that one of the processors sends or receives at least $\frac{n^3}{2\sqrt{2}PM^{\frac{3}{2}}} - 1$ 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 coincide with the lower bounds for the bandwidth and the latency costs of the sequential case. The lower bound on bandwidth cost of sequential matrix multiplication was previously shown (up to some multiplicative constant factor) by Hong and Kung [23].

One means of extending these lower bounds to more algorithms is by reduction. It is easy to reduce matrix multiplication to LU decomposition of a slightly larger order, as the following identity shows:

$$\begin{pmatrix} I & 0 & -B \\ A & I & 0 \\ 0 & 0 & I \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ 0 & 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 & -B \\ & I & A\cdot B \\ & & I \end{pmatrix}.$$

This identity means that LU factorization without pivoting can be used to perform matrix multiplication; to accommodate 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 decomposition that uses only 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 nine square blocks each of dimension n; then the Cholesky decomposition of T is

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

where X is the Cholesky factor of $D' \equiv D - B^TB - B^TA^TAB$ 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 three times larger) as $A \cdot B$ appears in L^T, the decomposition of T. Note, however, that $A \cdot A^T$ also appears in T. Conceivably, computing $A \cdot B$ from A and B incurs less communication cost if we are also given $A \cdot A^T$, so the above is not a sufficient reduction from matrix multiplication to Cholesky decomposition.[3] 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.

- The set is commutative with respect to addition and to multiplication (by the symmetries of the corresponding tables).
- The set is associative with respect to addition: regardless of ordering of summation, the sum is 1^* if one of the summands is 1^*, otherwise it is 0^* if one of the summands 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 main 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[4]

$$C = \begin{pmatrix} 1^* & 0 & \dots & 0 \\ 0^* & 1^* & & \vdots \\ \vdots & & \ddots & 0 \\ 0^* & \cdots & 0^* & 1^* \end{pmatrix} \cdot \begin{pmatrix} 1^* & 0^* & \cdots & 0^* \\ & \ddots & \ddots & \vdots \\ \vdots & & 1^* & 0^* \\ 0 & \dots & & 1^* \end{pmatrix} \equiv C' \cdot C'^T. \tag{2.1}$$

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

[4]By writing $X \cdot Y$ we mean the resulting matrix, assuming the straightforward n^3 matrix multiplication algorithm. This has to be stated clearly, as the distributivity does not hold for the starred values.

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

$$T' \equiv \begin{pmatrix} I & A^T & -B \\ A & C & 0 \\ -B^T & 0 & C \end{pmatrix} = \begin{pmatrix} I & & \\ A & C' & \\ -B^T & (A\cdot B)^T & C' \end{pmatrix} \cdot \begin{pmatrix} I & A^T & -B \\ & C'^T & A\cdot B \\ & & C'^T \end{pmatrix} \equiv L \cdot L^T. \tag{2.2}$$

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 in Algorithm 1.

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: Construct $T'$ as in (2.2).
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 check this bit before performing an operation. This increases the bandwidth cost by at most a constant factor. Alternatively, we can use signalling NaNs as defined in the IEEE floating point standard [1] 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^*. One can also eliminate 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.

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.

Classical Cholesky decomposition algorithms. 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:

$$L(i,i) = \sqrt{T'(i,i) - \sum_{k \in [i-1]} (L(i,k))^2}, \tag{2.3}$$

$$L(i,j) = \frac{1}{L(j,j)} \left(T'(i,j) - \sum_{k \in [j-1]} L(i,k) \cdot L(j,k) \right), \ i > j, \tag{2.4}$$

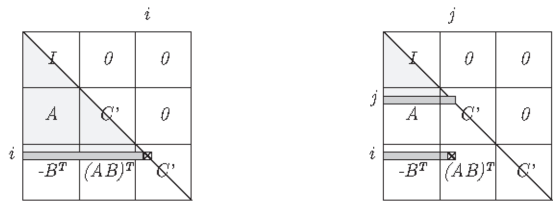

FIG. 2.1. *Dependencies of $L(i,j)$ for diagonal entries (left) and other entries (right). Dark grey represents the sets $S_{i,i}$ (left) and $S_{i,j}$ (right). Light grey represents indirect dependencies.*

where $[t] = \{1, \ldots, t\}$. A "classical" Cholesky decomposition algorithm computes each of these $O(n^3)$ FLOPS, which may be reordered using only commutativity and associativity of addition. By the no-pivoting and no-distributivity restrictions on 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 on the entries of L and impose a partial ordering on the computation of the entries of L (see Figure 2.1). That is, when an entry $L(i,i)$ is computed, by (2.3), all the entries $\{L(i,k)\}_{k\in[i-1]}$ have already been computed.[5] Denote this set of entries by $S_{i,i}$, namely,

$$S_{i,i} \equiv \{L(i,k)\}_{k\in[i-1]}. \tag{2.5}$$

Similarly, when an entry $L(i,j)$ (for $i > j$) is computed, by (2.4), 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,

$$S_{i,j} \equiv \{L(i,k)\}_{k\in[j-1]} \cup \{L(j,k)\}_{k\in[j]}. \tag{2.6}$$

LEMMA 2.3. *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 (2.5) and (2.6). The base case—the correctness of computing $L(1,1)$—is immediate. Assume by 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 2.1 or (2.2), (2.5), and (2.6)). 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 main diagonal and the case where it is not.
  If $i = j$, then by (2.3) $L(i,i)$ 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 2.1 and (2.3)).

[5]While this partial ordering constrains the scheduling of FLOPS, it does not uniquely identify a computation directed acyclic graph (DAG), for the additions within one summation can be in arbitrary order (forming arbitrary subtrees in the computation DAG).

TABLE 2.1
Arithmetic operations: x, y stand for any real *values. For consistency, $-0^* \equiv 0^*$ and $-1^* \equiv 1^*$.*

$\pm$	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^*
$x \geq 0$	$\sqrt{x}$

If $i > j$, then by the inductive assumption the divisor $L(j,j)$ of (2.4) is correctly computed to be 1^* (recall Figure 2.1 and the definition of C' in (2.1)). 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 (2.4)). However, every subtracted product (recall (2.4)) 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 2.1, right-hand side, row j). However, every subtraction performed (recall (2.4)) 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 2.3. ☐

Communication Analysis. We now know that Algorithm 1 correctly multiplies matrices "classically" and so has known communication lower bounds given by Theorem 2.1 and Corollary 2.2. 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 cost) 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 2.4 (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 cost of the matrix multiplication algorithm is at most a constant times the bandwidth cost of the Cholesky algorithm.*

Therefore any bandwidth or latency cost lower bound for classical matrix multiplication applies to classical Cholesky, asymptotically speaking.

COROLLARY 2.5. *In the sequential case, with a fast memory of size M, the bandwidth cost lower bound for Cholesky decomposition is $\Omega(n^3/M^{1/2})$, and the latency cost 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, as each message delivers at most M words, the latency lower bound for step 3 is by a factor of M smaller than its bandwidth cost lower bound, as desired. ∎

COROLLARY 2.6. *In the parallel case, with a 2D layout on P processors, the bandwidth cost lower bound for Cholesky decomposition is $\Omega(n^2/P^{1/2})$, and the latency cost lower bound is $\Omega(P^{1/2})$.*

Proof. The argument in the parallel case is analogous to that of Corollary 2.5. The construction of input and retrieval of output at steps 2 and 4 of Algorithm 1 contribute bandwidth of $O(\frac{n^2}{P})$. Therefore the lower bound of the bandwidth $\Omega(\frac{n^3}{P\sqrt{M}})$ is determined by step 3, the Cholesky decomposition. The lower bound on the latency of step 3 is therefore $\Omega(\frac{n^3}{PM^{3/2}})$, as each message delivers at most M words. Plugging in $M = \Theta(\frac{n^2}{P})$ yields $B = \Omega(P^{1/2})$. ∎

3. Upper bounds.

In this section we discuss known algorithms for Cholesky decomposition and their bandwidth and latency cost analyses in the sequential two-level memory model, in the sequential hierarchical memory model, and in the parallel computing model. Throughout this section, we will use the notation $B(n)$ and $L(n)$ for bandwidth and latency costs, respectively, where n is the dimension of the square matrix. These functions may also depend on the parameters M, the size of the fast memory, and P, the number of processors, but we omit its reference when it is clear from context. See also [10] for latency analysis of two of the Cholesky decomposition algorithms in the two-level (out-of-core) memory model.

Note that all the Cholesky decomposition algorithms in this section perform almost exactly the same computation: they all perform the computations defined in (2.3) and (2.4) and differ only in the order of summation and the timing of computation. Therefore, none of them use distributivity. Without this latter fact, it would not make sense to compare their bandwidth and latency costs to the lower bound.

In section 10.1.1 of [22], standard error analyses of the Cholesky decomposition algorithm are given. These hold for any ordering of the summation of (2.3) and (2.4) and therefore apply to all Cholesky decomposition algorithms below.

3.1. Sequential algorithms.

3.1.1. Data storage.

Before reviewing the various sequential algorithms let us first consider the underlying data structure which is used for storing the matrix. Although it does not affect the bandwidth cost, it may have a significant influence on the latency cost of the algorithm: it may or may not allow retrieving many relevant words using only a single message. As a simple example, consider computing the sum of $n \leq M$ numbers in slow memory, which obviously requires reading these n words. If they are in consecutive memory locations, this can be done in one read operation, the minimum possible latency cost. 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 cost.

The various methods for data storage (only some of which are implemented in *LAPACK*) appear in Figure 3.1. We can partition these data structures into two classes: column-major and block-contiguous.

Column-major storage. The column-major data structures store the matrix entries in columnwise order. This means that they are most fit for algorithms that access a column at a time (e.g., the left and right looking naïve algorithms). However, the

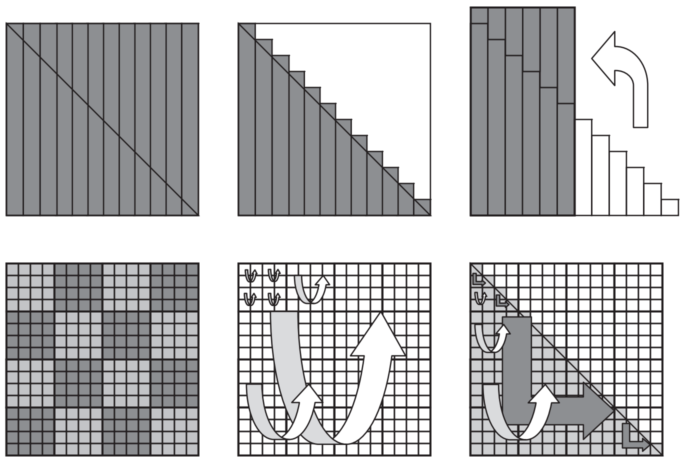

FIG. 3.1. *Underlying data structures. Top (column-major): full, old packed, rectangular full packed. Bottom (block-contiguous): blocked, block-recursive format, recursive full packed.*

latency cost of algorithms that access a block at a time (e.g., $LAPACK$'s implementation) will fail to achieve optimality, as retrieving a block of size $b \times b$ requires at least b messages, even if a single message can deliver b^2 words. This means a possible increase in the latency cost by a factor of b (where b is typically the order of $\sqrt{M}$).

As the matrices of interest for Cholesky decomposition are symmetric, storing the entire matrix (known as "full storage") wastes space. About half of the space can be saved by using the "old packed" or the "rectangular full packed" storages. The latter has the advantage of a more uniform indexing, which allows faster addressing.

Block-contiguous storage. The block-contiguous data structures store the matrix entries in a way that allows a read or write of a block using a single message. This may reduce the latency cost of a Cholesky decomposition algorithm that accesses a $b \times b$ block at a time by a factor of b, compared with using column-major storage. We distinguish between two types of block-contiguous storage formats. The *blocked* data structure stores each block of the matrix in a contiguous space of the memory. For this, one has to know in advance the size of blocks to be used by the algorithm (which is a machine-specific parameter).

The elements of each block may be stored as contiguous subblocks, where each subblock is of a predefined size. The data structure may include several such layers of subblocks. This "layered" data structure may fit the hierarchical memory model (see section 3.2 for further discussion of this model). The next data structure allows the benefits of block-contiguous storage without knowledge of machine-specific parameters.

The *block-recursive* format [6, 16, 17, 29] (also known as the bit interleaved layout, space-filling curve storage, or Morton ordering format) stores each of the four $n/2 \times n/2$ submatrices contiguously, and the elements of each submatrix are ordered so that the smaller submatrices are each stored contiguously, and so on recursively. This format is "cache-oblivious." This means that it allows access to a single block using a single message (or a constant number of messages), without knowing in advance the size of the blocks (see Figure 3.1). Both the blocked and block-recursive formats have packed versions, where about half of the space is saved by using the symmetry of the matrices (see [16] for recursive full packed data structures). The algorithm in [4] uses a hybrid data structure in which only half the matrix is stored, and recursive ordering is used on triangular submatrices and column-major ordering is used on square submatrices.

Conversion on the fly. Since block-contiguous storage yields better latency cost than column-major storage for some algorithms, it can be worthwhile to convert a matrix stored in column-major order to the blocked data structure (with block size $b = \Theta(M)$) before running the algorithm. This can be done by reading $\Theta(M)$ elements at a time, in a columnwise order (which requires one message), and then writing each of these elements to the right location of the new matrix. We write these words using $\Theta(\sqrt{M})$ messages (one per each relevant block). Thus, the total number of messages is $O(\frac{n^2}{\sqrt{M}})$, which is asymptotically dominated by $O(\frac{n^3}{M^{3/2}})$ for $M = \Omega(n)$. Converting back to column-major storage can be done in similar way (with the same asymptotic cost).

Other variants of these data structures. Similar to the column-major data structures, there are row-major data structures that store the matrix entries rowwise. All the above-mentioned packed data structures have versions that are indexed to efficiently store the lower (as in Figure 3.1) and upper triangular part of a matrix.

We consider the application of each of the algorithms below to column-major or block-contiguous data structures only. Adapting each of these algorithms to other compatible data structures (row-major vs. column-major, upper triangular vs. lower triangular, full storage vs. packed storage) is straightforward.

3.1.2. Arithmetic count of sequential algorithms.

The arithmetic operations of all the sequential algorithms considered below are exactly the same, up to reordering. The arithmetic count of all these algorithm is therefore the same and is given only once.

By (2.3), (2.4) the total arithmetic count is

$$A(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} (2j-1) = \frac{n^3}{3} + \Theta(n^2).$$

3.1.3. Upper bound for matrix multiplication.

In this section we establish the bandwidth and latency costs of matrix multiplication, which is used as a subroutine in some of the Cholesky decomposition algorithms.

We recall the recursive matrix multiplication algorithm of Frigo et al. [17]. Their algorithm, given as Algorithm 2, works by a divide-and-conquer approach, where at each step the algorithm splits the largest of three dimensions.

LEMMA 3.1 (see [17]). *The bandwidth cost $B_{MM}(n,m,r)$ of multiplying two matrices of dimensions $n \times m$ and $m \times r$ is*

$$B_{MM}(n,m,r) = \Theta\left(\frac{nmr}{\sqrt{M}} + nm + mr + nr\right).$$

This result is stated slightly differently in [17]. We obtain the form above by setting the cache-line length (or transfer block size) equal to 1. The latency cost of this algorithm varies according to the data structure used and the dimensions of the input and output matrices.

LEMMA 3.2. *When using a recursive contiguous-block data structure, the latency cost of matrix multiplication is*

$$L_{MM}(n,m,r) = \Theta\left(\frac{nmr}{M^{3/2}} + \frac{nm + mr + nr}{M}\right),$$

Algorithm 2 $C = RMatMul(A, B)$: A Recursive Matrix Multiplication Algorithm.

Input: A, B, two matrices of dimensions $n\times m$ and $m\times r$.
Output: C, a matrix of dimension $n\times r$, so that $C = A \cdot B$

```
\\ Partition $A, B, C$ by dividing each dimension in half, e.g., $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$
1: if $n = m = r = 1$ then
2:     $C_{11} = A_{11} \cdot B_{11}$
3: else if $n = \max\{n, m, r\}$ then
4:     $\begin{pmatrix} C_{11} & C_{12} \end{pmatrix} = RMatMul\left(\begin{pmatrix} A_{11} & A_{12} \end{pmatrix}, B\right)$
5:     $\begin{pmatrix} C_{21} & C_{22} \end{pmatrix} = RMatMul\left(\begin{pmatrix} A_{21} & A_{22} \end{pmatrix}, B\right)$
6: else if $m = \max\{n, m, r\}$ then
7:     $C = RMatMul\left(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, \begin{pmatrix} B_{11} & B_{12} \end{pmatrix}\right)$
8:     $C = C + RMatMul\left(\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix}, \begin{pmatrix} B_{21} & B_{22} \end{pmatrix}\right)$
9: else
10:    $\begin{pmatrix} C_{11} \\ C_{21} \end{pmatrix} = RMatMul\left(A, \begin{pmatrix} B_{11} \\ B_{21} \end{pmatrix}\right)$
11:    $\begin{pmatrix} C_{12} \\ C_{22} \end{pmatrix} = RMatMul\left(A, \begin{pmatrix} B_{12} \\ B_{22} \end{pmatrix}\right)$
12: end if
13: return C
```

and when using column-major or row-major layout, the latency cost is

$$L_{MM}(n, m, r) = \Theta\left(\frac{nmr}{M} + \frac{nm + mr + nr}{\sqrt{M}}\right).$$

Proof. Let m', n', r' be the dimensions of the problem the first time all three matrices fit into the fast memory. Then m', n', r' differ by at most a factor of 2 and thus are all $\Theta(\sqrt{M})$. Therefore, assuming the recursive contiguous-block data structure, each of the three matrices resides in $O(1)$ square blocks, and so reading and/or writing each matrix incurs a latency cost of $\Theta(1)$. Thus, the total latency cost is $L_{MM}(n) = \Theta(\frac{B_{MM}(n)}{M})$.

If the column-major or row-major data structures are used, then each such block of size $\Theta(\sqrt{M})\times\Theta(\sqrt{M})$ is read or written using $\Theta(\sqrt{M})$ messages (one for each of its rows or columns), and therefore the total latency cost is $L_{MM}(n) = \Theta(\frac{B_{MM}(n)}{\sqrt{M}})$. $\square$

We note that Algorithm 2 is cache-oblivious. That is, the algorithm achieves the bandwidth and latency costs given above with no knowledge of the size of the fast memory. The iterative blocked matrix multiplication algorithm can also achieve this performance, but the block size must be chosen to be $\Theta(\sqrt{M})$.

3.1.4. Upper bound for triangular system solution. In this section we establish the bandwidth and latency costs of another subroutine used in some of the Cholesky decomposition algorithms, that of a triangular system solution (also known as triangular solve) with multiple right-hand sides (TRSM). Algorithm 3 is a recursive version of TRSM (a variation of this algorithm appears in [4], designed to utilize a nonrecursive, tuned implementation of matrix multiplication). Below, we assume each matrix multiplication is implemented with the recursive algorithm (Algorithm 2).

ALGORITHM 3 $X = RTRSM(A, U)$: RECURSIVE TRIANGULAR SOLVER.

```
Input: $A, U$ two $n \times n$ matrices, $U$ is upper triangular.
Output: $X$, so that $X = A \cdot U^{-1}$
    \\ Partition $A, U, X$ by dividing each dimension in half, e.g., $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$
1: if $n = 1$ then
2:     $X = A/U$
3: else
4:     $X_{11} = RTRSM(A_{11}, U_{11})$
5:     $X_{12} = RTRSM(A_{12} - X_{11} \cdot U_{12}, U_{22})$
6:     $X_{21} = RTRSM(A_{21}, U_{11})$
7:     $X_{22} = RTRSM(A_{22} - X_{21} \cdot U_{12}, U_{22})$
8: end if
9: return $X$
```

Bandwidth cost. As no communication is needed for sufficiently small matrices (other than reading the entire input and writing the output), the bandwidth cost of this algorithm is given by the recurrence

$$B_{TRSM}(n) = \begin{cases} 4 \cdot B_{TRSM}\left(\frac{n}{2}\right) + 2 \cdot B_{MM}(\frac{n}{2}) & \text{if } n > \sqrt{\frac{M}{3}}, \\ 3n^2 & \text{otherwise,} \end{cases}$$

where $B_{MM}(n) = B_{MM}(n, n, n)$ is the bandwidth cost of multiplying two n-by-n matrices. Assuming the matrix multiplication is done using Algorithm 2, we have $B_{MM}(n) = O(\frac{n^3}{\sqrt{M}} + n^2)$. The solution to this recurrence (and the total bandwidth cost of recursive TRSM) is then

$$B_{TRSM}(n) = O\left(\frac{n^3}{\sqrt{M}} + n^2\right).$$

Latency cost. Assuming a block-contiguous data structure (recursive, or with the correct block size picked), the latency cost is given by the recurrence

$$L_{TRSM}(n) \leq \begin{cases} 4 \cdot L_{TRSM}\left(\frac{n}{2}\right) + 2 \cdot L_{MM}(\frac{n}{2}) & \text{if } n > \sqrt{\frac{M}{3}}, \\ 3 & \text{otherwise,} \end{cases}$$

where $L_{MM}(n) = L_{MM}(n, n, n) = O(\frac{n^3}{M^{3/2}})$ is the latency cost of recursive matrix multiplication. The solution to this recurrence is

$$L_{TRSM}(n) = O\left(\frac{n^3}{M^{3/2}}\right).$$

Again, we note that Algorithm 3 is cache-oblivious. The iterative blocked TRSM algorithm can also achieve this performance, but the block size must be chosen to be $\Theta(\sqrt{M})$.

Equipped with the communication costs of matrix multiplication and triangular system solving, we proceed to the following subsections, where we present several Cholesky decomposition algorithms and provide the communication cost analyses which yield the first five enumerated conclusions in the introduction as well as the results shown in Table 1.1.

3.1.5. The naïve left-looking Cholesky algorithm.

We start with revisiting the two naïve algorithms (left-looking and right-looking), and we see that both have non-optimal bandwidth and latency, as stated in conclusion 1 of the introduction.

The naïve left-looking Cholesky decomposition algorithm is given in Algorithm 4 and Figure 3.2. Note that Algorithm 4 assumes that two columns (k and j) of the matrix can fit into fast memory simultaneously (i.e., $M > 2n$).

ALGORITHM 4 NAÏVE LEFT-LOOKING CHOLESKY ALGORITHM.

```
1: for $j = 1$ to $n$ do
2:   read $A(j:n, j)$ from slow memory
3:   for $k = 1$ to $j-1$ do
4:     read $A(j:n, k)$ from slow memory
5:     update diagonal element: $A(j,j) \leftarrow A(j,j) - A(j,k)^2$
6:     for $i = j+1$ to $n$ do
7:       update $j$th column element: $A(i,j) \leftarrow A(i,j) - A(i,k)A(j,k)$
8:     end for
9:   end for
10:  calculate final value of diagonal element: $A(j,j) \leftarrow \sqrt{A(j,j)}$
11:  for $i = j+1$ to $n$ do
12:    calculate final value of $j$th column element: $A(i,j) \leftarrow A(i,j)/A(j,j)$
13:  end for
14:  write $A(j:n, j)$ to slow memory
15: end for
```

Bandwidth cost. Assuming $M > 2n$, we follow Algorithm 4, and communication occurs at lines 2, 4, and 14, so the total number of words transferred between fast and slow memory while executing the entire algorithm is given by

$$\sum_{j=1}^{n}\left[2(n-j+1)+\left(\sum_{k=1}^{j-1}(n-j+1)\right)\right]=\frac{1}{6}n^3+n^2+\frac{5}{6}n.$$

In the case when $M < 2n$, each column j is read into fast memory in segments of size $M/2$. For each segment of column j, the corresponding segments of previous columns k are read into fast memory in sequence to update the current segment. In this way, the total number of words transferred between fast and slow memory does not change.

Latency cost. Assuming $M > 2n$ and the matrix is stored in column-major order, each column is contiguous in memory, so the total number of messages is given by

$$\sum_{j=1}^{n}\left[2+\left(\sum_{k=1}^{j-1}1\right)\right]=\frac{1}{2}n^2+\frac{3}{2}n.$$

In the case when $M < 2n$, an entire column cannot be read/written in one message, so the column must be read/written in multiple messages of size $O(M)$. Again, assuming the matrix is stored in column-major order, segments of columns will be contiguous in memory, and the latency is given by the bandwidth divided by the message size: $O(\frac{n^3}{M})$.

The algorithm can be adjusted to work one row at a time (up-looking) if the matrix is stored in row-major order (with no change in bandwidth or latency), but

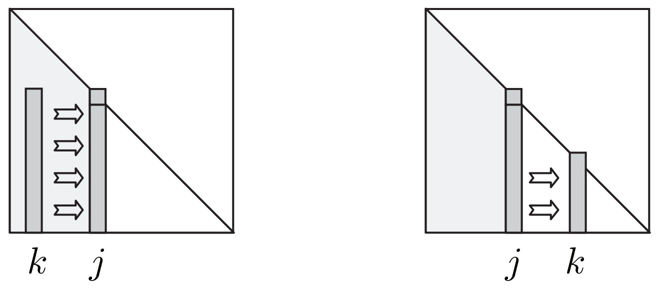

FIG. 3.2. *Naïve algorithms. Left: left-looking. Right: right-looking. Column j is currently being computed (both algorithms). Light grey is the area already computed. Column k is the column being read (left-looking) or written (right-looking).*

a block-contiguous storage format will increase the latency (columns would not be contiguous in memory in this case). This analysis, along with that of section 3.1.6, yields conclusion 1 in the introduction.

3.1.6. The naïve right-looking Cholesky algorithm. The naïve right-looking Cholesky decomposition algorithm is given in Algorithm 5 and Figure 3.2. Note that Algorithm 5 assumes that two columns (k and j) of the matrix can fit into fast memory simultaneously (i.e., $M > 2n$).

ALGORITHM 5 NAÏVE RIGHT-LOOKING CHOLESKY ALGORITHM.

```
1: for $j = 1$ to $n$ do
2:   read $A(j:n, j)$ from slow memory
3:   calculate final value for diagonal element: $A(j,j) \leftarrow \sqrt{A(j,j)}$
4:   for $i = j+1$ to $n$ do
5:     calculate final value for $j$th column element: $A(i,j) \leftarrow A(i,j)/A(j,j)$
6:   end for
7:   for $k = j+1$ to $n$ do
8:     read $A(k:n, k)$ from slow memory
9:     for $i = k$ to $n$ do
10:      update $k$th column element: $A(i,k) \leftarrow A(i,k) - A(i,j)A(k,j)$
11:    end for
12:    write $A(k:n, k)$ to slow memory
13:  end for
14:  write $A(j:n, j)$ to slow memory
15: end for
```

Bandwidth cost. Assuming $M > 2n$, we follow Algorithm 5, and communication occurs at lines 2, 8, 12, and 14, so the total number of words transferred between fast and slow memory while executing the entire algorithm is given by

$$\sum_{j=1}^{n} \left[2(n-j+1) + \sum_{k=1}^{j-1} 2(n-k+1) \right] = \frac{1}{3}n^3 + n^2 + \frac{2}{3}n.$$

In the case when $M < 2n$, more communication is required. In order to perform the column factorization, the column is read into fast memory in segments of size $M-1$, updated with the main diagonal element (which must remain in fast memory), and written back to slow memory. In order to update the trailing kth column, the

kth column is read into fast memory in segments of size $(M-1)/2$ along with the corresponding segment of the jth column. In order to compute the update for the entire column, the kth element of the jth column must remain in fast memory. After updating the segment of the kth column, it is written back to slow memory. Thus, the naïve right-looking algorithm requires more reads from slow memory when two columns of the matrix cannot fit into fast memory, but this increases the bandwidth by only a constant factor.

Latency cost. Assuming $M > 2n$ and the matrix is stored in column-major order, the total number of messages is given by

$$\sum_{j=1}^{n}\left[2+\left(\sum_{k=j+1}^{n}2\right)\right]=n^2+n.$$

As in the case of the naïve left-looking algorithm, when $M < 2n$, the size of a message is no longer the entire column. Assuming the matrix is stored in column-major order, message sizes are of the order equal to the size of fast memory. Thus, for $O(n^3)$ bandwidth, the latency is $O(\frac{n^3}{M})$.

This right-looking algorithm can be easily transformed into a down-looking row-wise algorithm in order to handle row-major data storages, with no change in bandwidth or latency. This analysis, along with that of section 3.1.5, yields conclusion 1 in the introduction.

3.1.7. *LAPACK*'s POTRF. We next consider an implementation available in *LAPACK* (see [5]) and show that it is bandwidth optimal when using the right block size (as stated in conclusion 2 of the introduction) and can also be made latency-optimal, assuming the correct data structure is used (as stated in conclusion 3 of the introduction).

LAPACK's Cholesky algorithm, POTRF (Algorithm 6), is a blocked left-looking algorithm. For simplicity, we assume the block size b evenly divides the matrix size n. See Figure 3.3 for a diagram of the algorithm. Note that by the partitioning of line 2, A_{21} is $b{\times}(j-1)b$, A_{22} is $b{\times}b$, A_{31} is $(n/b-j)b{\times}(j-1)b$, and A_{32} is $(n/b-j)b{\times}b$.

Algorithm 6 *LAPACK* POTRF.

```
1: for $j = 1$ to $n/b$ do
2:     partition matrix so that diagonal block $(j,j)$ is $A_{22}$: $\begin{pmatrix} A_{11} & * & * \\ A_{21} & A_{22} & * \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$
3:     update diagonal block $(j,j)$ (SYRK): $A_{22} \leftarrow A_{22} - A_{21}A_{21}^T$
4:     factor diagonal block $(j,j)$ (POTF2): $A_{22} \leftarrow \mathrm{Chol}(A_{22})$
5:     update column panel (GEMM): $A_{32} \leftarrow A_{32} - A_{31}A_{21}^T$
6:     triangular system solution for column panel (TRSM): $A_{32} \leftarrow A_{32}A_{22}^{-T}$
7: end for
```

The communication costs of Algorithm 6 depend on the subroutines which perform symmetric rank-b update, matrix multiply, and triangular system solve, respectively. For simplicity, we will assume that the symmetric rank-b update is computed with the same bandwidth and latency as a general matrix multiply. Although general matrix multiply requires more communication, this assumption will not affect the asymptotic count of the communication required by the algorithm. We will also

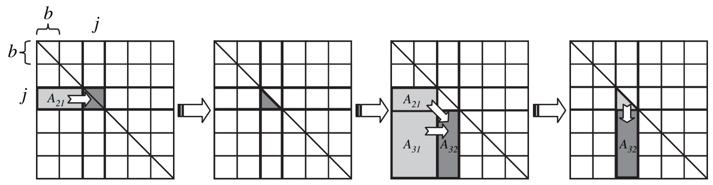

FIG. 3.3. *LAPACK's* `POTRF` *Cholesky decomposition algorithm.*

assume that the block size b is chosen so that three blocks can fit into fast memory simultaneously; that is, we assume that

$$1 \le b \le \sqrt{\frac{M}{3}}.$$

In this case, the factorization of the diagonal block (line 4) requires only $\Theta(b^2)$ words to be transferred since the entire block fits into fast memory. Also, in the triangular system solution for the column panel A_{32} (line 6), the triangular matrix A_{22} is only one block, so the computation can be performed by reading the blocks of A_{32} in sequence and updating them individually in fast memory. Thus, the amount of communication required by that subroutine is $\Theta(b^2)$ times the number of blocks in the column panel. Finally, the dimensions of the matrices in the matrix multiply of line 3 during the jth iteration of the loop are $b\times b$, $b\times(j-1)b$, and $(j-1)b\times b$, and the dimensions of the matrices in the matrix multiply of line 5 during the jth iteration are $(n/b-j)b\times b$, $(n/b-j)b\times(j-1)b$, and $(j-1)b\times b$.

Bandwidth cost. Under the assumptions above, an upper bound on the number of words transferred between slow and fast memory while executing Algorithm 6 is given by

$$B(n) \le \sum_{j=1}^{n/b}\Big[B_{MM}(b,(j-1)b,b) \\ + \Theta(b^2) + B_{MM}((n/b-j)b,(j-1)b,b) + (n/b-j)\Theta(b^2)\Big],$$

where $B_{MM}(m,n,r)$ is the bandwidth cost required to execute a matrix multiplication of matrices of size $m\times n$ and $n\times r$ in order to update a matrix of size $n\times r$. Since B_{MM} is nondecreasing in each of its variables, we have

$$\begin{aligned} B(n) &\le \frac{n}{b}\left[B_{MM}(b,n,b) + \Theta(b^2) + B_{MM}(n,n,b) + \frac{n}{b}\Theta(b^2)\right] \\ &\le \frac{n}{b}\left[2B_{MM}(n,n,b) + \Theta(b^2) + \frac{n}{b}\Theta(b^2)\right]. \end{aligned}$$

Assuming the matrix multiply algorithm used by *LAPACK* achieves the same bandwidth cost as the one given in section 3.1.3 (the iterative blocked algorithm with the correct block size suffices), we have

$$B_{MM}(n,n,b) = \Theta\left(\frac{n^2 b}{\sqrt{M}} + n^2 + nb\right).$$

Since $b \le \sqrt{M}$ and $b \le n$, $B_{MM}(n,n,b) = O(n^2)$. Thus,

$$B(n) = O\left(\frac{n^3}{b} + n^2\right),$$

and choosing a block size $b = \Theta(\sqrt{M})$ gives

$$B(n) = O\left(\frac{n^3}{\sqrt{M}} + n^2\right)$$

and achieves the lower bound (as stated in conclusion 2 of the introduction). Note that choosing a block size $b = 1$ reduces the blocked algorithm to the naïve left-looking algorithm (see Algorithm 4), which has bandwidth cost $O(n^3)$.

Latency cost. Since all reads and writes are done block by block, the optimal latency cost

$$L(n) = O\left(\frac{n^3}{M^{3/2}}\right)$$

is achieved if a block-contiguous data structure is used with block size of $\Theta(\sqrt{M})$. However, as the current implementations of $LAPACK$ use column-major data structures, the latency cost in practice is

$$L(n) = O\left(\frac{n^3}{M} + \frac{n^2}{\sqrt{M}}\right).$$

As mentioned in section 3.1.1, in the case where $M = \Omega(n)$, converting a matrix in column-major order to the blocked layout incurs a bandwidth cost of $O(n^2)$ and latency cost of $O(\frac{n^3}{M^{3/2}})$. Thus, in this case, the costs of conversion-on-the-fly and Algorithm 6 combined still achieve the communication lower bounds. These results are stated in conclusion 3 of the introduction.

3.1.8. Rectangular recursive Cholesky algorithm. The next recursive algorithm for Cholesky decomposition and its bandwidth analysis follow the recursive LU-decomposition algorithm of Toledo (see [28]). The algorithm given here (Algorithm 7) is in fact a simplified version of Toledo's: there is no pivoting, and $L = U^T$. For completeness we repeat the bandwidth analysis and provide a latency analysis. The bandwidth proves to be optimal (as stated in conclusion 2 of the introduction), but the latency does not (as stated in conclusion 3 of the introduction).

Bandwidth cost. The analysis of the bandwidth cost follows that of Toledo [28]. Following Algorithm 7 we either read/write one column (if $n = 1$) or make two recursive calls and perform one matrix multiplication. Therefore, the bandwidth cost is given by the recurrence

$$B(m,n) = \begin{cases} B\left(m, \frac{n}{2}\right) + B_{MM}(m - \frac{n}{2}, \frac{n}{2}, \frac{n}{2}) + B\left(m - \frac{n}{2}, \frac{n}{2}\right) & \text{if } n > 1, \\ 2m & \text{if } n = 1. \end{cases}$$

This recurrence assumes that no more than one column of the matrix can fit in fast memory ($m \leq M$). We note that Algorithm 7 communicates less than Toledo's algorithm, so the analysis of [28] provides an upper bound on the solution of this recurrence. One can use a similar analysis to show in the case where $m \geq M$, $B(m,n) = \Omega(\frac{mn^2}{\sqrt{M}} + mn\log n)$, yielding the following lemma.

LEMMA 3.3 (see [28]). *Given a matrix multiplication subroutine with bandwidth cost equivalent to Algorithm* 2, *the bandwidth cost of the rectangular recursive Cholesky algorithm on an* $m \times n$ *matrix* ($m \geq n$) *where at most one column of the matrix can fit into fast memory at a time* ($m \geq M$) *is*

$$B(m,n) = \Theta\left(\frac{mn^2}{\sqrt{M}} + mn\log n\right).$$

ALGORITHM 7 $L = RectangularRChol(A)$: RECTANGULAR RECURSIVE CHOLESKY ALGORITHM.

```
Input: $A$, an $m \times n$ section of positive semidefinite matrix ($m \geq n$). See Figure 3.4
    for block partitioning.
Output: $L$, a lower triangular matrix, so that $A = L \cdot L^T$
1: if $n = 1$ then
2:     $L = A/\sqrt{A(1,1)}$
3: else
4:     $\begin{pmatrix} L_{11} \\ L_{21} \\ L_{31} \end{pmatrix} = RectangularRChol\left(\begin{pmatrix} A_{11} \\ A_{21} \\ A_{31} \end{pmatrix}\right)$
5:     $\begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - RMatMul\left(\begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix}, L_{21}^T\right)$ {See section 3.1.3 for
    $RMatMul$.}
6:     $\begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} = RectangularRChol\left(\begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix}\right)$
7:     $L_{12} = 0$
8: end if
9: return L
```

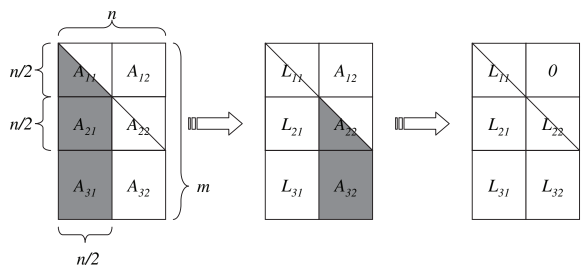

FIG. 3.4. *Rectangular recursive Cholesky algorithm.*

Thus, the application of the algorithm to an n-by-n matrix (where $n^2 > M$) yields a bandwidth cost of

$$B(n,n) = \Theta\left(\frac{n^3}{\sqrt{M}} + n^2 \log n\right).$$

This is optimal, provided that $M \leq \frac{n^2}{\log^2 n}$ or, since any algorithm is bandwidth optimal when the whole matrix fits into fast memory, when $n^2 \leq M$. Hence the range of nonoptimality is very small, so we consider Algorithm 7 to have optimal bandwidth cost.

We note that in the case where $m < M$, the analysis is slightly different. In the case where multiple columns fit into fast memory, the bandwidth cost recurrence is

$$B(m,n) = \begin{cases} B\left(m, \frac{n}{2}\right) + B_{MM}(m - \frac{n}{2}, \frac{n}{2}, \frac{n}{2}) + B\left(m - \frac{n}{2}, \frac{n}{2}\right) & \text{if } mn > M, \\ \Theta(mn) & \text{if } mn \leq M, \end{cases}$$

where the base case arises when the subproblem fits entirely in fast memory (which may be before Algorithm 7 reaches its base case). The solution of this recurrence is $B(m,n) = \Theta(\frac{mn^2}{\sqrt{M}} + mn \log \frac{mn}{M})$, which is smaller than the cost in the case $m \geq$

M. Because this difference is so small, we omit the distinction in conclusion 2 and Table 1.1.

Latency cost. For the latency cost, we consider the column-major and block-recursive data storage formats. In order to show that this algorithm does not attain optimal latency cost, we provide lower bounds in each case.

In the case of column-major storage, the latency cost of the entire algorithm is bounded below by the multiplication of step 5 at the top of the recursion tree. This is a square multiplication of size $\frac{n}{2}$, so the latency cost with column-major storage is $\Omega(\frac{n^3}{M})$ (from section 3.1.3). Thus, using column-major storage, Algorithm 7 can never achieve optimal latency cost.

In the case of block-recursive storage, the latency of the entire algorithm is bounded below by the latency required to resolve the base cases of the recursion. Assuming $m > M$, the base case occurs when $n = 1$, and the algorithm factors a single column. However, the block-recursive data structure does not store columns contiguously (up to 2 elements of the same column could be stored consecutively, depending on the recursive pattern), so reading a column requires $\Omega(n)$ messages. Since a base case is reached for each column, the latency cost contribution of all the base cases is $\Omega(n^2)$, a lower bound for the total latency cost of Algorithm 7. Since n^2 asymptotically dominates $\frac{n^3}{M^{3/2}}$ for $M > n^{2/3}$, this algorithm cannot achieve optimal latency cost in this case either.

We note that Algorithm 7 is cache-oblivious.

3.1.9. Square recursive Cholesky algorithm. The next recursive algorithm is a natural adaptation of the rectangular recursive algorithm of section 3.1.8. Instead of dividing the matrix only vertically, because no pivoting is required, we can also divide horizontally, yielding a square recursive algorithm (Algorithm 8). The algorithm has been considered before, but no asymptotic analysis was given prior to this work. We believe the algorithm first appears in [20] and can also be found in [3, 4, 21, 27]. Ahmed and Pingali [3] suggest the algorithm (though without asymptotic analysis of the bandwidth and latency costs). In [27] Simecek and Tvrdik give a detailed probabilistic cache-misses performance; however, no asymptotic claims on the worst-case bandwidth and latency are stated. We note that the algorithms in [4, 21] achieve the optimal bandwidth cost, but they assume data storages that prevent achieving optimal latency cost (neither the latency nor the bandwidth cost is explicitly given in those papers).

The analysis in this section proves that Algorithm 8 minimizes the bandwidth cost (as stated in conclusion 2 in the introduction) and if the block-recursive data storage is used, the latency cost as well.

Bandwidth cost. As no communication is needed for sufficiently small matrices (other than reading the entire input and writing the output), the bandwidth cost of this algorithm is given by the recurrence

$$B(n) = \begin{cases} 2 \cdot B\left(\frac{n}{2}\right) + B_{TRSM}\left(\frac{n}{2}\right) + B_{MM}\left(\frac{n}{2}\right) & \text{if } n > \sqrt{\frac{M}{3}}, \\ 2n^2 & \text{otherwise.} \end{cases}$$

Assuming the matrix multiplication is performed using Algorithm 2 and the triangular system solving is performed using Algorithm 3, the solution to this recurrence (and the total bandwidth cost of Algorithm 8) is

$$B(n) = O\left(\frac{n^3}{\sqrt{M}} + n^2\right).$$

ALGORITHM 8 $L = SquareRChol(A)$: SQUARE RECURSIVE CHOLESKY ALGORITHM.

```
Input: $A$, an $n \times n$ semidefinite matrix. See Figure 3.5 for block partitioning.
Output: $L$, a lower triangular matrix, so that $A = L \cdot L^T$
1: if $n = 1$ then
2:     $L = \sqrt{A(1,1)}$
3: else
4:     $L_{11} = SquareRChol(A_{11})$
5:     $L_{21} = RTRSM(A_{21}, L_{11}^T)$ {See section 3.1.4 for $RTRSM$.}
6:     $A_{22} = A_{22} - RMatMul(L_{21}, L_{21}^T)$ {See section 3.1.3 for $RMatMul$.}
7:     $L_{22} = SquareRChol(A_{22})$
8: end if
9: return $L$
```

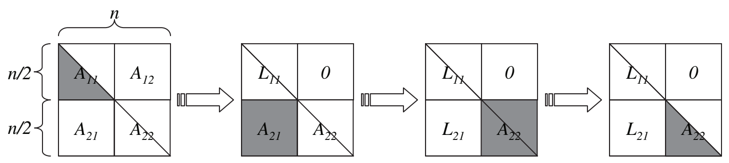

FIG. 3.5. *Square recursive Cholesky algorithm.*

Latency cost. The recurrence for the latency cost is similar:

$$L(n) = \begin{cases} 2 \cdot L\left(\frac{n}{2}\right) + L_{TRSM}\left(\frac{n}{2}\right) + L_{MM}\left(\frac{n}{2}\right) & \text{if } n > \sqrt{\frac{M}{3}}, \\ 2 & \text{otherwise.} \end{cases}$$

Assuming the matrix is stored using the block-recursive data structure and the recursive subroutines are used, the latency cost is given by

$$L(n) = O\left(\frac{n^3}{M^{3/2}}\right).$$

We note that like the rectangular recursive algorithm, Algorithm 8 is cache-oblivious.

3.2. Sequential algorithm for more than two memory levels. In real computers, there are usually more than two types of memory, and in fact there is a hierarchy of memories, ranging from a very fast small memory to the largest and slowest memory [4]. We next consider the lower and upper bounds of Cholesky decomposition assuming such a model of *hierarchical memory machines*. We observe that none of the known algorithms for Cholesky decomposition allows cache-oblivious achievement of optimal latency cost (conclusion 4 of the introduction), except the square recursive Cholesky algorithm (conclusion 5 of the introduction).

3.2.1. Lower bound. Explicit lower bounds have been derived for various problems (including matrix multiplication) within the hierarchical memory model (see, for example, [26]). Moreover, the known lower bound for the two-level memory model can be applied here by considering any two consecutive levels of the hierarchy as the fast and slow memories, treating all faster memories as part of the fast memory and all the slower memories as part of the slow one. Assuming the number of levels is some constant, this may be the correct lower bound, up to a constant factor. For the

Cholesky decomposition, this gives the following bandwidth and latency cost lower bounds.

COROLLARY 3.4. *Let Alg be a "classical" Cholesky decomposition algorithm implemented on a machine with some constant d levels of memory of size* $M_1 \leq \cdots \leq M_d$*, with inverse bandwidth* $\beta_1 \leq \cdots \leq \beta_d$ *and with latency* $\alpha_1 \leq \cdots \leq \alpha_d$*. Then the bandwidth cost of Alg is*

$$\Omega\left(\sum_{i\in[d-1]}\left\{\beta_i \cdot \left(\frac{n^3}{\sqrt{M_i}} - M_i\right)\right\}\right), \tag{3.1}$$

and the latency cost is

$$\Omega\left(\sum_{i\in[d-1]}\left\{\alpha_i \cdot \frac{n^3}{M_i^{3/2}}\right\}\right). \tag{3.2}$$

3.2.2. Upper bounds revisited. An algorithm may fail to achieve optimal communication costs on a hierarchical memory model even if it achieves optimal performance in the two-level memory model. For example, an algorithm that includes a parameter (e.g., block size) which allows tuning for better communication performance may be harder or impossible to tune optimally in the hierarchical memory model. On the other hand, as argued in [17], a cache-oblivious algorithm which achieves optimal communication costs in the two-level memory model will also perform optimally on a computer with multiple memory levels. We next revisit the communication performance of the Cholesky decomposition algorithms above in the context of the hierarchical memory model.

LAPACK's POTRF. In the two-level memory model, the *LAPACK* implementation can achieve optimal bandwidth cost, and if the block-contiguous data structure is used, it can also achieve optimal latency cost. To this end one must tune the block size parameter b of the algorithm (and the block size of the data structure).

This has a drawback when applying the *LAPACK* algorithm to hierarchical memory machines: setting b to fit a smaller memory level results in inefficient bandwidth and latency costs in the higher levels. Setting b according to a larger memory level results in block I/O that is too large to fit the smaller levels. Either way, one cannot achieve optimal bandwidth and latency costs between every pair of levels in the memory hierarchy with only one tunable parameter.

Rectangular recursive algorithm. Recall from section 3.1.8 that the recursive Cholesky decomposition algorithm adopted from [28] is cache-oblivious and achieves optimal bandwidth cost for practically all ranges of memory sizes, but it cannot guarantee optimal latency cost when $M > n^{2/3}$. Thus, the algorithm may minimize the bandwidth cost in the hierarchical model, but if, for example, there is a memory level of size greater than $n^{2/3}$ (other than the slowest one holding the original input), then this algorithm yields suboptimal latency cost.

Square recursive algorithm. Recall from section 3.1.9 that the square recursive algorithm minimizes both bandwidth and latency costs, and it is cache-oblivious. Thus, it achieves communication optimality in the hierarchical memory model and is the only algorithm to do so.

3.3. 2D parallel algorithms. Let us now consider the so-called 2*D* parallel algorithms, namely, those that assume $M = (\frac{n^2}{P})$ local memory size and start with the n^2 matrix elements spread across the processors (i.e, no repetition of elements).

In the parallel model, communication occurs between pairs of processors, and n words sent as one message can be communicated in time $\alpha+\beta n$. Unlike the sequential model, we do not consider the data structure used to store the parts of the matrix local to each processor. We assume that the cost of "packing" a message into a contiguous block of memory is included in the time $\alpha+\beta n$. We count communication costs along the critical path; that is, if many pairs of processors send messages of the same size simultaneously, the cost along the critical path is that of one message (we do not consider communication resource contention among processors).

We also consider the cost of a "broadcast," where one processor sends the same message to many other processors. Broadcasts can be implemented in many different ways, and implementations are often limited by the network topology. Following [11], we assume a broadcast of one message incurs a latency cost of $\alpha \cdot \log P$. That is, if the root processor sends the message to two processors and each of those processors forward the message to two more, the information can reach P processors after $\log P$ steps. This implementation requires a topology that includes a binary tree of connections among the P processors (a 2D torus or nearest-neighbor connection topology would not suffice, for example).

Assuming a sufficient network topology, we show upper bounds for bandwidth and latency costs which match the lower bounds up to a logarithmic factor (as stated in conclusion 6 of the introduction).

3.3.1. The ScaLAPACK implementation. The ScaLAPACK [11] routine `PxPOTRF` computes the Cholesky decomposition of a symmetric positive definite matrix A of size $n\times n$ distributed over a grid of P processors. The matrix is distributed block cyclically, but only half of the matrix is referenced or overwritten (we assume the lower half here). See Algorithm 9 and Figure 3.6. We assume a network topology such that each processor column and each processor row are connected via a binary tree.

ALGORITHM 9 *ScaLAPACK* `PxPOTRF`.

```
1:  for $j = 1$ to $n/b$ do
2:      processor owning block $(j, j)$ computes Cholesky decomposition
3:      broadcast result to processors down processor column
4:      parallel for each processor owning blocks in column panel $j$ do
5:          update blocks in panel with triangular system solver
6:          broadcast results across processor row
7:      end for
8:      parallel for each processor owning diagonal block $(i, i)$ $(i > j)$ do
9:          rebroadcast results down processor column
10:     end for
11:     parallel for each processor owning blocks in trailing matrix do
12:         update blocks with symmetric rank-$b$ update
13:     end for
14: end for
```

We assume that the block dimension is $b\times b$, where n/b is an integer and the processors are organized in a square grid ($P_r = P_c = \sqrt{P}$), where $n/\sqrt{P}$ is also an integer. After computing the Cholesky decomposition of a diagonal block (j, j) locally, the result must be communicated to all processors which own blocks in the column panel below the diagonal block (i.e., blocks (i, j) for $j < i \leq n/b$) in order to update

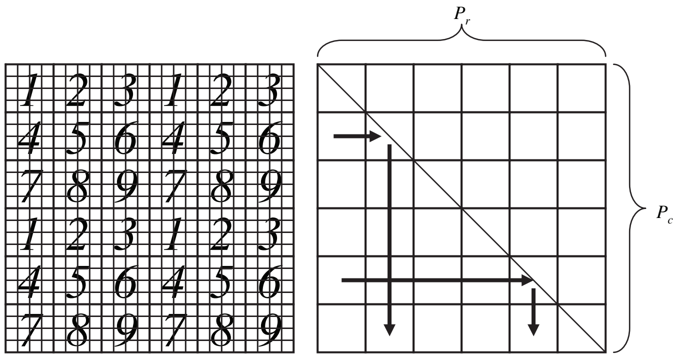

FIG. 3.6. *ScaLAPACK's* PxPOTRF. *Left: Block-cyclic distribution of the matrix to the processors. Here $n = 24, b = 4, P_c = P_r = 3$. Right: Processor grid, information flow. Here $P_c = P_r = 6$.*

those blocks. Since the matrix is distributed block cyclically, there can be at most $\sqrt{P}$ such processors. After the column panel is updated using a triangular system solver, those results must be communicated to other processors in order to perform the rank-b updates to blocks in the trailing matrix. For a given block (k, l) in the trailing matrix A_{22}, the update depends on the kth block of the column panel (block (k, j)) and the transpose of the lth block of the column panel (block (l, j)). Thus, after a processor computes an update to block (i, j) in the column panel, it must broadcast the result to processors which own blocks in the ith row panel ($P_r = \sqrt{P}$ different processors). Then, after the processor owning the diagonal block (i, i) receives that update, it re-broadcasts to processors which own blocks in the ith column panel ($P_c = \sqrt{P}$ different processors).

The result of the Cholesky decomposition of the diagonal block is a lower triangular matrix, so the number of words in each message of the broadcast down the column is only $b(b+1)/2$. A broadcast to P processors requires $O(\log P)$ messages. Any processor which owns a block in the column panel below the diagonal block (j, j) will own $\frac{n/b}{\sqrt{P}} = \frac{n}{b\sqrt{P}}$ blocks. Such a processor computes the updates for all the blocks it owns, and then broadcasts all the results together (total of $\frac{nb}{\sqrt{P}}$ words) to $\sqrt{P}$ processors (which is done using $O(\log P)$ messages). Once a processor owning the corresponding diagonal blocks receives this message, it rebroadcasts the message down the column, requiring another $O(\log P)$ messages. Thus, the total number of messages along the critical path is

$$\sum_{j=1}^{n/b} O(\log P) = O\left(\frac{n}{b}\log P\right),$$

and the number of words along the critical path is

$$\sum_{j=1}^{n/b}\left[O(b^2\log P) + O\left(\frac{nb}{\sqrt{P}}\log P\right)\right] = O\left(\left(nb + \frac{n^2}{\sqrt{P}}\right)\log P\right).$$

Thus, setting the block size at $b = \Theta(\frac{n}{\sqrt{P}})$, the total number of messages required is $O(\sqrt{P}\log P)$, and the total number of words is $O(\frac{n^2}{\sqrt{P}}\log P)$. We note that by setting $b = \frac{n}{\sqrt{P}}$, for example, the matrix is no longer stored block cyclically. Instead, each processor owns one large block of the matrix, and since the full matrix is stored, nearly half of the processors own blocks which are never referenced or updated. Further, parallelism is lost as the algorithm progresses across the column panels. However, this does not cause an asymptotic degradation in the computation time of the algorithm. For each of the $\sqrt{P}$ column panels, there are three phases of computation: Cholesky decomposition of the diagonal block, triangular system solver to update the column panel, and matrix multiply to update the trailing matrix. Thus, setting $b = \frac{n}{\sqrt{P}}$, the computation cost of the algorithm (not including lower order terms) is

$$\sqrt{P}\left[\frac{1}{3}\left(\frac{n}{\sqrt{P}}\right)^3 + \left(\frac{n}{\sqrt{P}}\right)^3 + 2\left(\frac{n}{\sqrt{P}}\right)^3\right] = \frac{10}{3}\cdot\frac{n^3}{P}.$$

Thus, in choosing a large block size to attain the latency cost lower bound, we do not sacrifice the algorithm's ability to meet the computational asymptotic lower bound.[6] This analysis yields conclusion 6 in the introduction.

4. Conclusion.

In this paper we extended known lower bounds on the communication cost (both for bandwidth and for latency) of conventional ($O(n^3)$) matrix multiplication to Cholesky factorization, and we compared the cost of various Cholesky decomposition implementations to these lower bounds. We identified and analyzed existing algorithms that minimized communication for the two-level, hierarchical, and parallel memory models. We showed that the *ScaLAPACK* implementation minimized communication in the parallel model, up to an $O(\log P)$ factor. However, the optimal and cache-oblivious sequential algorithm is yet to be implemented in a publicly available library, such as *LAPACK*. Our six main conclusions detailing these results are listed in the introduction.

A "real" computer may be more complicated than any model we have discussed, 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 more complicated memory models and lower and upper communication bounds on such processors for future work.

Recently we extended lower bounds to other algorithms in [7], including QR decomposition and algorithms for eigenvalue problems, and identified/developed optimal $O(n^3)$ implementations . We further plan to extend results to Strassen's matrix multiplication and other "fast" algorithms.

REFERENCES

[1] *IEEE standard for floating-point arithmetic*, IEEE Std. 754-2008, (2008), pp. 1–58.

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

[3] 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, London, UK, 2000, Springer-Verlag, pp. 368–378.

[6] As noted by Grigori, one can choose a slightly smaller block size and achieve a computational cost of $\frac{1}{3}\frac{n^3}{P}$ while increasing the latency cost by only a polylogarithmic factor [19].

[4] B. S. Andersen, F. G. Gustavson, and J. Wasniewski, *A recursive formulation of Cholesky factorization of a matrix in packed storage format*, ACM Trans. Math. Software, 27 (2001), pp. 214–244.

[5] 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 Users Guide*, 3rd ed., SIAM, Philadelphia, 1999; also available from http://www.netlib.org/lapack/.

[6] M. Bader, R. Franz, S. Guenther, and A. Heinecke, *Hardware-oriented implementation of cache oblivious matrix operations based on space-filling curves*, in Parallel Processing and Applied Mathematics, 7th International Conference, PPAM 2007, Lecture Notes in Comput. Sci. 4967, Springer-Verlag, New York, 2008, pp. 628–638.

[7] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Minimizing communication in linear algebra*, SIAM J. Matrix Anal. Appl., submitted; also available at http://arxiv.org/abs/0905.2485.

[8] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Communication-optimal parallel and sequential Cholesky decomposition*, in SPAA '09: Proceedings of the 21st ACM Symposium on Parallelism in Algorithms and Architectures, 2009, pp. 245–252.

[9] M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob, and E. Vicari, *Optimal sparse matrix dense vector multiplication in the I/O-model*, Theoret. Comput. Sys., 47 (2010), pp. 934–962.

[10] N. Béreux, *Out-of-core implementations of Cholesky factorization: Loop-based versus recursive algorithms*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1302–1319.

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

[12] R. A. Chowdhury and V. Ramachandran, *Cache-oblivious dynamic programming*, in SODA '06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, New York, 2006, ACM, pp. 591–600.

[13] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal Parallel and Sequential QR and LU Factorizations*, Technical report EECS-2008-89, University of California Berkeley, Berkeley, CA, 2008, SIAM. J. Sci. Comput., submitted.

[14] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Implementing communication-optimal parallel and sequential QR and LU factorizations*, SIAM. J. Sci. Comput., submitted.

[15] J. Demmel, L. Grigori, and H. Xiang, *Communication-avoiding Gaussian elimination*, in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008.

[16] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström, *Recursive blocked algorithms and hybrid data structures for dense matrix library software*, SIAM Rev., 46 (2004), pp. 3–45.

[17] 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, Washington, DC, 1999, IEEE Computer Society, pp. 285–297.

[18] S. L. Graham, M. Snir, and C. A. Patterson, eds., *Getting up to Speed: The Future of Supercomputing*, Report of the National Research Council of the National Academies of Sciences, The National Academies Press, Washington, D.C., 2004; also available online from http://www.nap.edu.

[19] L. Grigori. *Personal communication*, 2009.

[20] F. G. Gustavson, *Recursion leads to automatic variable blocking for dense linear-algebra algorithms*, IBM J. Res. Dev., 41 (1997), pp. 737–756.

[21] 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, London, UK, 2001, Springer-Verlag, pp. 82–91.

[22] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.

[23] 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, New York, 1981, ACM, pp. 326–333.

[24] D. Irony and S. Toledo, *Communication-efficient parallel dense LU using a 3-dimensional approach*, in Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing, 2001.

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

[26] J. E. Savage, *Extending the Hong-Kung model to memory hierarchies*, in COCOON, 1995, pp. 270–281.

[27] 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, Washington, DC, 2004, IEEE Computer Society, pp. 190–197.

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

[29] D. Wise, *Ahnentafel indexing into Morton-ordered arrays, or matrix locality for free*, in Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing, London, UK, 2000, Springer-Verlag, pp. 774–783.

[30] C.-Q. Yang and B.P. Miller, *Critical path analysis for the execution of parallel and distributed programs*, in Proceedings from the 8th International Conference on Distributed Computing Systems, IEEE Computer Society, 1988, pp. 366–373.