ECOLE POLYTECHNIQUE

CENTRE DE MATHÉMATIQUES APPLIQUÉES

UMR CNRS 7641

91128 PALAISEAU CEDEX (FRANCE). Tél: 01 69 33 41 50. Fax: 01 69 33 30 11
http://www.cmap.polytechnique.fr/

Out-of-core implementations of Cholesky factorization: loop-based versus recursive algorithms.

Natacha Béreux

R.I. 602 *October 2006*

OUT-OF-CORE IMPLEMENTATIONS OF CHOLESKY FACTORIZATION: LOOP-BASED VERSUS RECURSIVE ALGORITHMS.

NATACHA BÉREUX*

Abstract. The aim of this paper is to develop an out-for-core Cholesky type solver for large dense complex symmetric linear systems stemming from Boundary Element Methods in electromagnetics or acoustics. We present and compare several variants of the Cholesky factorization algorithm. The candidate variants are the classical blocked left-looking variant and a more recent recursive formulation. Both have been implemented out-of-core for real positive definite matrices: the former in the POOCLAPACK library and the latter in the SOLAR library. We perform a theoretical analysis of the amount of I/O operations required by each variant in an out-of-core implementation. We consider two memory layouts for the left-looking algorithm: the one-tile and two-tiles approaches. We show that when the main memory is restricted, the one-tile approach yields less I/O operations. We then show that the left-looking variant requires less I/O operations than the recursive variant. We have implemented all the variants for complex matrices, and we report on numerical experiments.

Key words. Cholesky factorization, Out-of-Core algorithms

AMS subject classifications. 15A23, 65F05, 65Y20

1. Introduction.

1.1. Motivation. The Boundary Element Method (BEM), which is widely used in electromagnetic or acoustic scattering, consists of transforming the original scattering problem set in an unbounded domain into an integral equation set on the boundary of the scatterer [17]. In many situations, a symmetric formulation of the integral equation is preferred (such as the Electric Field Integral Equation in electromagnetism [5, 7]). The discretization of such an integral equation e.g., by Finite Element Methods (FEM) leads to a linear system of equations:

$$A\,X = B, \tag{1.1}$$

where the coefficient matrix A of order N is dense, complex valued, symmetric but non-Hermitian. It is observed that for complex symmetric matrices issued from the discretization of Boundary Integral Equations, pivoting is not required: thus, a Cholesky factorization can be used [6, 4].
For very large problems (e.g., $N \geq 100000$), direct solvers are too costly and preconditioned iterative algorithms are preferred [8, 9]. For intermediate size problems, where the matrix is too large to fit in-core, an out-of-core solver is very effective.

1.2. State of the art. The limited size of the memory is a major bottleneck for solving large industrial problems, e.g. in electromagnetics. Therefore, out-of-core implementations, which perform linear algebra operations on matrices stored on the disk, are still an active resarch field. We refer to [20] for an extensive survey of trends in out-of-core algorithms. The Cholesky factorization is one of the classical linear algebra operations of LAPACK library [3]. The parallel extension of LAPACK, ScaLAPACK, provides a parallel out-of-core implementation of Cholesky factorization [10]: the matrix is partitioned into column panels and a left-looking variant of Cholesky factorization is chosen. The out-of-core extension of PLAPACK library, POOCLAPACK, also provides a parallel out-of-core implementation of Cholesky factorization:

*CMAP, ECOLE POLYTECHNIQUE, CNRS, Route de Saclay 91128 Palaiseau France

the matrix is partitioned into tiles, and is factorized by a left-looking variant [13]. The tile approach is more scalable than the panel approach used in ScaLAPACK. Two slightly different tile approaches have been proposed in POOCLAPACK: the one-tile approach [23] and the two-tiles approach [18], depending on the number of full tiles allowed to reside in the memory simultaneously. The SOLAR library provides routines for out-of-core linear algebra computations [21]. It is designed to be portable across various architectures (personal, shared-memory, distributed memory computers) thanks to a portability layer, the Matrix Input Output Subroutines (MIOS), which manages the data transfers to and from the disk. Matrices are partitioned into primary blocks. A primary block is the basic unit of the matrix, stored contiguously on the disk. SOLAR includes in particular an implementation of Cholesky factorization routine, based on a recursive formulation. The matrix is recursively splitted. At each level, the solution of a large system with triangular matrix is computed and a large symmetric rank-k update is performed, through calls to out-of-core BLAS. At the leaf level, the matrix is factorized by an in-core Cholesky routine (from LAPACK or ScaLAPACK). The recursion is stopped when the matrix size is equal to a predefined blocksize (for instance when the matrix consists of 2 or 4×4 primary blocks). The performance reported in [21] are quite impressive: the recursive algorithm outperforms the classical left-looking algorithm.
We refer to [11] for a complete survey on recursion as a key for designing high-performance linear algebra library. Linear algebra algorithms may be formulated in a recursive way. This recursive formulation is able to improve the locality of reference: its in-core implementation performs fewer cache misses, and its out-of-core implementation less I/O operations. This fact is demonstrated for LU decomposition in [19]. Recursive data structures have also been developped [2, 1]. Recently, a sparse Cholesky solver based on a recursive formulation of Cholesky algorithm, at the sparse and at the dense level, combined with a recursive layout of the data has been proposed in [15]. This solver is part of the TAUCS library.

1.3. Aim. We seek to compute the Cholesky decomposition of large dense matrices, stored on the disk, on a computer with a limited amount of memory available. We focus on sequential algorithms. We survey several variants, and memory layouts for computing this decomposition. Then, we propose a theoretical comparison of these variants, substantiated by numerical experiments, to find out the most effective implementation.
The paper is organised as follows. Section 2 describes two variants of Cholesky algorithm, the left-looking variant and the recursive variant. Then, out-of-core implementations of these variants are detailed in §3. We describe in particular the one-tile and the two-tiles approaches, which correspond to different memory layout. These out-of-core implementations rely on out-of-core BLAS, which are described in §4. A theoretical analysis of the data transfers from the disk to the memory is performed in §5 for the left-looking variant and in §6 for the recursive variant. Numerical experiments are conducted in §7. Our conclusions are presented in §8.

2. The Cholesky factorization. We briefly recall the definition of the Cholesky decomposition of a symmetric matrix A. Then, we review two algorithms for the computation of the Cholesky factor L. Both algorithms are partitioned algorithms. The matrices A and L are partitioned into tiles. Basic operations in partitioned algorithms are matrix-matrix operations on these tiles. Therefore, partitioned algorithms are known to enjoy a high level of data-reuse. This is critical for an efficient out-of-core implementation [20]. We review a loop-based algorithm, the left-looking

Cholesky algorithm, and a recursive algorithm. A loop-based algorithm is based on nested do-loops, whereas a recursive algorithm is based on a recursive splitting of the matrix.

2.1. The Cholesky factorization. Let A be a symmetric matrix, such that A admits a symmetric factorization

$$A = LL^T, \tag{2.1}$$

where L is a lower triangular matrix. This condition is satisfied by real definite positive matrices and also by some complex, symmetric matrices, in particular by matrices issued from Boundary Element Method [4, 6]. We assume hereafter that A admits a symmetric factorization and we review different algorithms for the computation of the triangular factor L.
Upon completion of the algorithm, the triangular factor L overwrites the original matrix A. In the following, CHOL denotes a generic routine for the computation of the lower triangular Cholesky factor. Depending on the context, CHOL stands for a variant of the algorithm or for an optimized computational kernel.

2.2. A loop-based algorithm: the left-looking variant. We follow the formalism introduced by R. Van De Geijn [22] to recap the left-looking variant of Cholesky algorithm.
Let A and L be partitioned into quadrants with square diagonal blocks:

$$A = \begin{pmatrix} A_{TL} & \star \\ A_{BR} & A_{BL} \end{pmatrix}, \quad L = \begin{pmatrix} L_{TL} & 0 \\ L_{BR} & L_{BL} \end{pmatrix}.$$

The diagonal blocks of L, L_{TL} and L_{BL} are lower triangular. The $\star$ indicates that the corresponding part of the matrix is not referenced (due to symmetry).
Using the partitionings in (2.1), we obtain the partitioned matrix expression:

$$\begin{pmatrix} A_{TL} & \star \\ A_{BR} & A_{BL} \end{pmatrix} = \begin{pmatrix} L_{TL}\,L_{TL}^T & \star \\ L_{BL}\,L_{TL}^T & L_{BL}\,L_{BL}^T + L_{BR}\,L_{BR}^T \end{pmatrix}. \tag{2.2}$$

This equality must hold when the factorization is completed. A possible choice for the content of A at an intermediate step of the factorization is:

$$A := \begin{pmatrix} L_{TL} & \star \\ A_{BL}\,L_{TL}^{-T} & A_{BR} \end{pmatrix}. \tag{2.3}$$

The bottom-right part of the matrix A_{BR} is not changed. The loop-invariant (2.3) yields the left-looking variant, described on Figure 2.1.

2.3. A recursive algorithm. Recursion is expected to provide a new generation of high-performance linear algebra algorithm. Recursive algorithms are based on a recursive splitting of the matrices, and this splitting is able to take advantage of a hierarchical memory. In [19], S. Toledo shows that a recursive formulation of LU decomposition improves the locality of reference. The recursive algorithm is thus not only more concise but also more efficient than the classical right-looking variant of LAPACK. Recursive blocked algorithms have been introduced for several dense and sparse linear algebra operations, e.g. Cholesky decomposition [1], [14], [2], [15]. In [21], the author proposes the use of recursive algorithms in an out-of-core library, SOLAR.

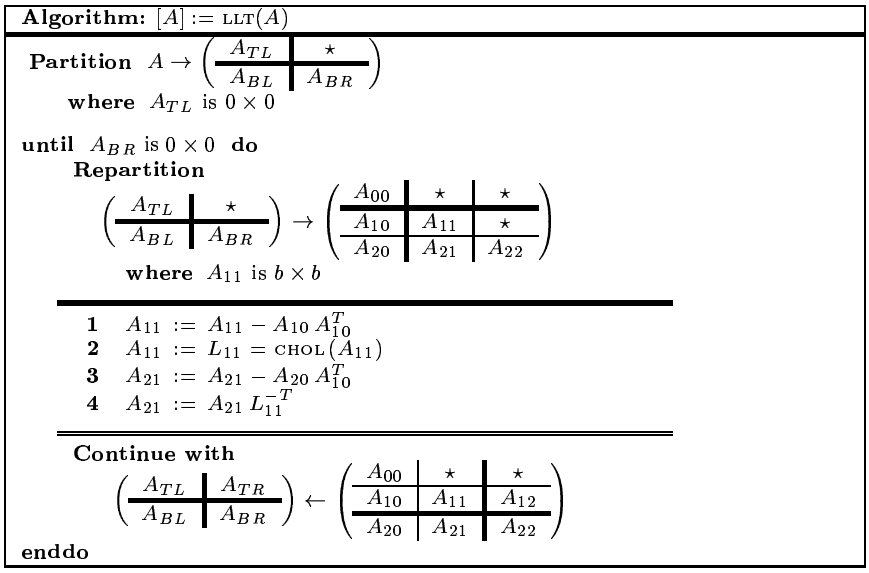

FIGURE 2.1. *Loop-based, partitioned, left-looking variant of Cholesky algorithm*

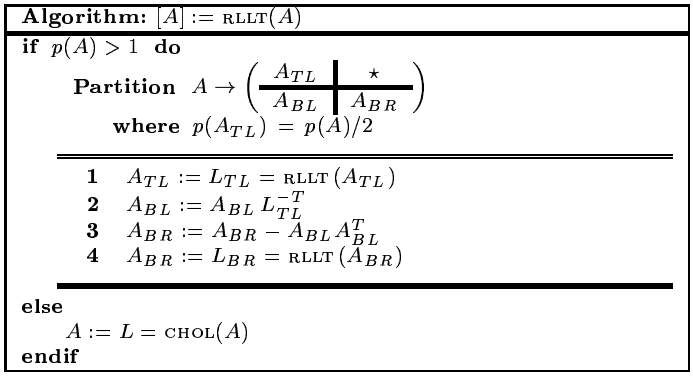

FIGURE 2.2. *Recursive variant of Cholesky algorithm. The number of tiles of a matrix A is denoted by* $p(A)$.

A recursive template for Cholesky factorization can be found e.g. in [11]. The matrix A is seen as a block matrix of $p(A) \times p(A)$ tiles. It is recursively splitted, until the subblocks consist of a single tile. Hence, the leaves of the recursion tree are matrices (and not scalars) and the computation at the leaf level is based on level 3 BLAS. We show in Figure 2.2 the main steps of the recursive variant of Cholesky algorithm.

3. Out-of-core implementations. From now on, the matrix is assumed to reside on disk. We describe the data layout, and then detail the implementation of a left-looking variant and of a recursive variant of Cholesky algorithm. Two implementations of Cholesky left-looking variant are sketched: a two-tiles and a one-tile, depending on the number of tiles (respectively two and one) allowed to reside in memory simultaneously.

3.1. Data layout. We use the tile approach, already implemented in POOCLAPACK [13] [23]. In the tile approach, the matrix A of size $n \times n$ is partitioned into

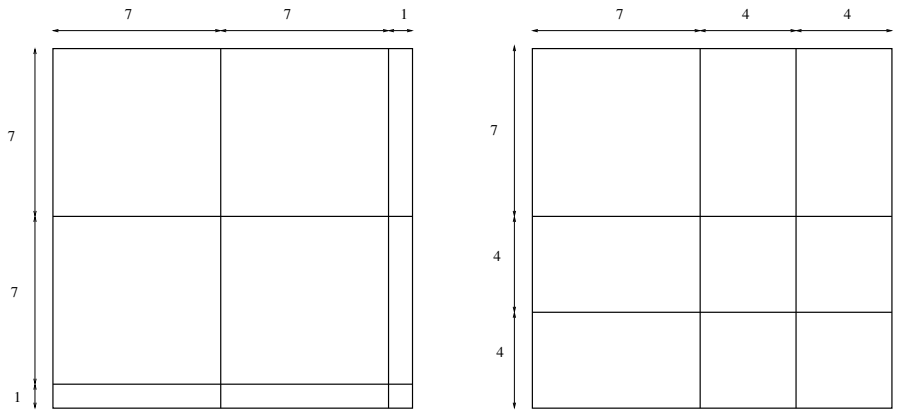

FIGURE 3.1. *A matrix of size* 15×15 *is partitioned using an arithmetic partitioning (on the left) and using a recursive partitioning (on the right). In both cases, the tile size* t *is equal to* 7*: it represents the default block size in the arithmetic partitioning and the maximum size of the leaves in the recursion tree for the recursive partitioning.*

$p \times p$ tiles.

3.1.1. Matrix tiling. Let $t \in \mathbb{N}$ be the tile size. We use two types of partitioning of the matrix:

- a **recursive** partitioning: the interval $[1, n]$ is recursively halved until the size of the subintervals is lower than t.
- a **arithmetic** partitioning: the interval $[1, n]$ is partitioned into subintervals of size t, except possibly for the last subinterval.

Examples of partitioning are shown on Figure 3.1. The recursive variants of Cholesky algorithms are based on a recursive partitioning of the matrix. The left-looking variants are usually based on an arithmetic partitioning.
The size of the main memory constraints the size of the tile, but p can be very large (it is only constrained by the size of the disk). Hence, this approach is well-adapted to large out-of-core matrices. Another choice is made in the out-of-core parallel implementation of Cholesky factorization in ScaLAPACK: the matrix is partitioned into slabs of columns (possibly of variable width), instead of tiles. But this approach is known to lack scalability [20] [16].

3.1.2. Storage details. The partitioned matrix is decomposed in block-rows of width t. Each row is stored tile by tile. Each tile is itself in column-major order.
From a practical standpoint, each block-row is stored as a collection of records in a direct-access file. In this file, each record corresponds to a primary block, stored contiguously on disk. The storage is illustrated by Figure 3.2 We distinguish between a block and a tile. A **block** corresponds to a physical record in the direct-access file storing a block-row of the matrix. A **tile** corresponds to an element of the logical partitioning of the matrix. A tile is split on disk into several blocks of size $t \times b$, with $b \leq t$. In the following, blocks are called **narrow blocks**, since they have fewer columns than rows. Note that if $b = t$, a tile is identical to a block.

3.2. Implementations of the left-looking variant. Let the matrix A be partitioned into tiles. We describe two implementations of the left-looking variant for the computation of Cholesky factor:

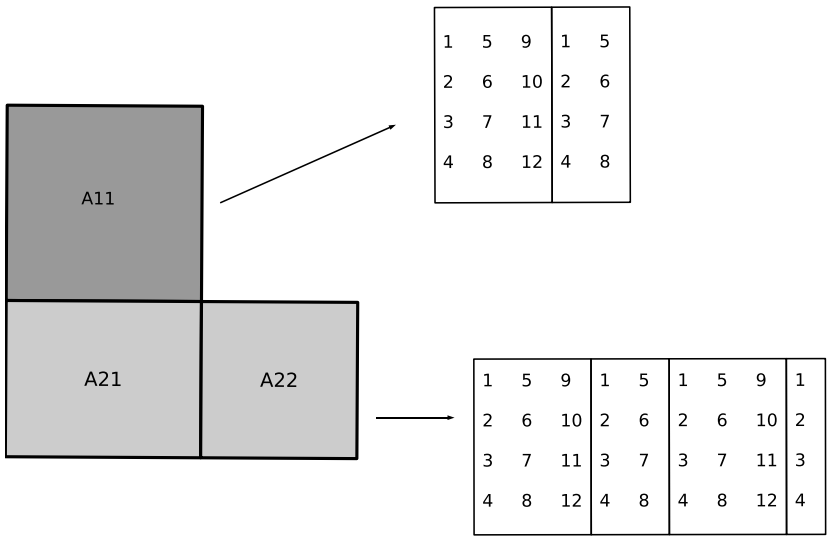

FIGURE 3.2. *Example of a matrix out-of-core storage. The matrix of size* 9×9 *is recursively partitioned into tiles of size* $t \times t$ *with* $t = 5$*. The matrix is stored on the disk by block-rows. Hence, two direct-access files are needed: one contains the tile* A_{11}*, and the other the tiles* A_{21} *and* A_{22}*. Each tile is stored as a collection of records, where each record corresponds to a block of size* $t \times b$ *(*$b = 3$*), stored in column-major order.*

- in the **two-tiles** approach, two tiles are allowed to reside in-core simultaneously [13].
- in the **one-tile** approach, there is only one tile in-core [23]. Therefore, this tile may be larger, and the implementation benefits from a better ratio between computations and I/O operations.

In both approaches, two extra buffers are needed for the in-core storage of two narrow blocks.

3.2.1. Two-tiles approach. We strictly follow the sequential implementation stated in [13]. Two tiles simultaneously reside in the main memory. One contains the diagonal tile A_{11} and the other the current tile of column A_{21}. We recap the main steps and refer to [13] and [18] for further details:

- **Step 1**: Tile A_{11} is read from disk and updated as $A_{11} := A_{11} - A_{10}A_{10}^T$ by a sequence of narrow out-of-core symmetric rank-k updates (routine OOC_TILE_SYRK).
- **Step 2**: Tile A_{11} is factored by an in-core factorization routine. Then, tile A_{11} is written back on disk, but a copy is kept in memory.
- **Steps 3 and 4**: The update of block-column A_{21}, $A_{21} := (A_{21} - A_{20}\,A_{10}^T)L_{11}^{-T}$ is performed tile by tile. Each tile of A_{21} is brought in turn into memory, updated by a sequence of narrow out-of-core multiply and adds (routine OOC_TILE_GEMM) and next, by solving (in-core) a multiple right-hand side triangular system of matrix A_{11}^T. Then, the current tile of A_{21} is written back on disk, and the next tile is read.

To optimize data transfers, tile A_{11} remains in-core for Steps 1 to 4. Therefore, we define mixed BLAS, OOC_TILE_SYRK and OOC_TILE_GEMM, whose arguments are an in-core tile and out-of-core tiles:

- OOC_TILE_SYRK overwrites the (in-core) tile C of size $t \times t$ with $C - A\,A^T$, where the tile A is stored on disk.
- OOC_TILE_GEMM overwrites the (in-core) tile C of size $t \times t$ with $C + A\,B^T$, where A and B are out-of-core tiles.

These operations can be implemented as sequences of operations on narrow matrices, as advocated in [13]. The narrow block technique, analyzed in §5.3 allows to increase the ratio of computation over I/O operations.

3.2.2. One-tile approach. The one-tile approach is introduced in [23]. The motivation is to increase the size of the current tile, which is read and written. We briefly state the differences with the two-tiles implementation, for each step of the algorithm:

- Steps 1 and 2 are not changed
- Tile A_{11} no longer remains in-core after Step 2. It is replaced in-core by the current tile of A_{21}. Then the routine OOC_TILE_GEMM is used to update A_{21}.
- For Step 4, a new mixed BLAS is needed for the solution of the triangular system with matrix A_{11}: OOC_TILE_TRSM. This routine overwrites the (in-core) tile A_{21} with the solution X of $X\,A_{11}^T = A_{21}$, where the triangular tile A_{11} is stored on disk. Tile A_{11} is read narrow block by narrow block in one of the two extra buffers provided.

This implementation uses a larger current tile and benefits from a better flops to I/O operations ratio for the mixed BLAS. Nevertheless, tile A_{11} has to be read twice.

3.3. Implementations of the recursive variant. Let us now consider the out-of-core implementation of the recursive factorization sketched in Algorithm 2.2. A similar implementation is reported in [21], in the framework of SOLAR project.

3.3.1. Basic implementation. The main features of the factorization are the following:

- **recursive partitioning** of the matrix until the matrix consists of a single tile.
- **Step 1**: at the leaf level, tile A_{11} is brought into memory, factored by an in-core Cholesky routine, and written back to the disk. At another level, the factorization is called recursively.
- **Step 2** consists in solving a triangular linear system of matrix L_{TL}. At the leaf level, this operation is performed by an in-core routine (for instance LAPACK TRSM), after reading the tile L_{TL}. If the matrices are larger, an out-of-core routine OOC_TRSM is called.
- **Step 3** is a symmetric update operation. It is performed either by calling an in-core routine (for instance LAPACK SYRK), or an out-of-core routine OOC_SYRK.

The recursive implementation calls two out-of-core BLAS, whose arguments are all out-of-core matrices:

- OOC_SYRK overwrites the out-of-core matrix C of size $n \times n$ with $C - A\,A^T$, where A is an out-of-core matrix of size $n \times k$. If $n = t$, A and C are brought into memory, and an in-core routine is called.

- OOC_TRSM overwrites the out-of-core matrix B of size $m \times n$ by the solution X of $X\,A^T = B$, where A is a triangular out-of-core matrix of size $n \times n$. When $m = n = t$, both arguments are brought into memory and an in-core routine is called.

So, out-of-core routines switch to in-core routine as soon as the arguments are sufficiently small matrices, that is matrices of size $t \times t$, where t is the size of tiles of the recursively partitioned matrix.

3.3.2. Implementation of a hybrid variant. A small modification of the implementation of the recursive algorithm allows to reduce the disk to memory traffic. It is based on the following observation.
Let A be a matrix of 2×2 tiles, initially stored on disk. Let us examine the recursive procedure for the factorization of A. The matrix is split into:

$$A = \begin{pmatrix} A_{11} & \star \\ A_{21} & A_{22} \end{pmatrix}.$$

The first tile A_{11} is read, factored in-core and written back on the disk. Then, it is read again for the update of A_{21}. Tile A_{21} once computed is written back on the disk and immediately read again for the update of A_{22}. Once updated, tile A_{22} is written and read again for the computation of its Cholesky factor. As a result, the recursive factorization of a matrix A of 2×2 tiles demands reading 5 tiles and writing 4 tiles. The following schedule for the factorization of a matrix of 2×2 tiles avoids reading a tile immediately after it has been written:

- read, factorize and write A_{11} (keeping a copy in-core),
- read, update and write A_{21} (keeping a copy in-core),
- read, update, factorize and write A_{22}.

It only requires reading 3 tiles and writing 3 tiles.
Hence, we suggest to stop the recursion when the matrix size is $2t \times 2t$ and to switch to this variant. The total gain in I/O operations is evaluated in §6.3.

4. Out-of-core BLAS. The recursive schedule of Cholesky factorization stated in §3.3 calls out-of-core BLAS, for the out-of-core computation of:

- a symmetric rank-k update, OOC_SYRK, $C - A\,A^T$, where A is a $n \times k$ matrix and C is a $k \times k$ matrix.
- the solution X of $X\,A^T = B$, OOC_TRSM where A is a triangular matrix of size $n \times n$, and X overwrites B (of size $m \times n$).

These operations (as Cholesky factorization) may be performed either by a recursive algorithm or by a loop-based algorithm. In [21], the out-of-core sequential implementation of Cholesky factorization is based on a recursive strategy for the factorization and calls to loop-based out-of-core BLAS. We follow this option. We recall block-partitioned algorithms for the symmetric rank-k update SYRK and the triangular system solver TRSM as they are derived in Chapter 8 and Chapter 16 of [12].

4.1. Out-of-core SYRK. We show the block down-right moving variant of the symmetric rank-k update, SYRK.

The main features of the out-of-core implementation of this algorithm are the following:

- Tile C_{11} is brought into the main memory.
- **Step 1** is performed by applying a sequence of narrow rank-k updates through a call to OOC_TILE_SYRK. Then, C_{11} is written back on the disk.

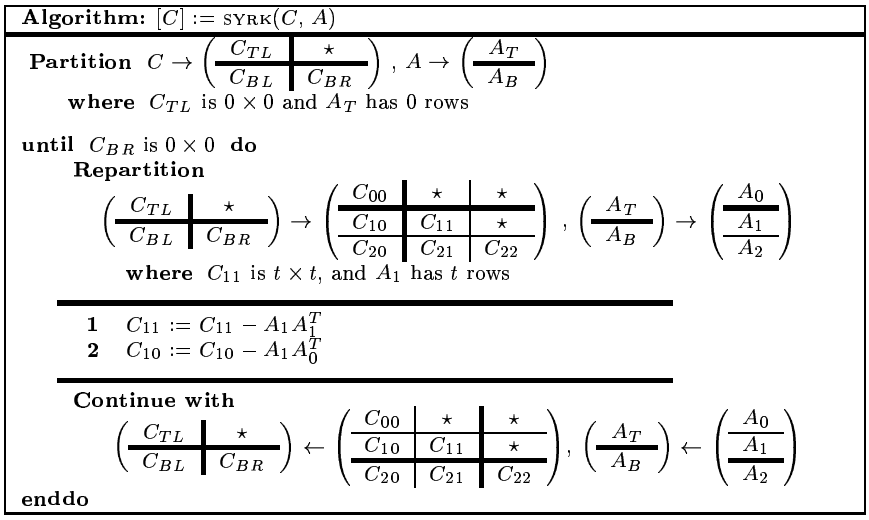

FIGURE 4.1. *Blocked, loop-based algorithm for the computation of the symmetric rank-k update,* $C := C - A\,A^T$.

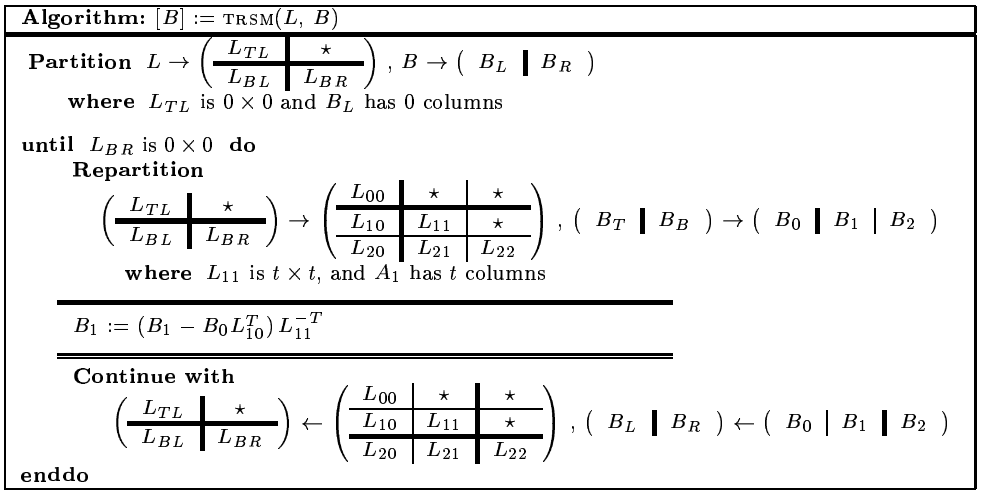

FIGURE 4.2. *Blocked, loop-based algorithm for the computation of the solution of the linear system* $X\,L^T = B$, *with lower triangular matrix* L, *where* X *overwrites* B.

- **Step 2** is performed tile by tile. Each tile of C_{10} is read, updated by a call to OOC_TILE_GEMM, and written back on disk.

4.2. Out-of-core TRSM. The solution of $X\,L^T = B$ is computed by the blocked right-moving variant of the triangular solver TRSM:

The out-of-core two-tiles implementation uses the same techniques as Step 3 and 4 of Cholesky left-looking factorization:

- Tile L_{11} is brought into memory.
- Column B_1 is replaced tile by tile by the corresponding column of tiles of the solution. The current tile of B_1 is read, updated by calling the routine OOC_TILE_GEMM. Then, the linear sytem with matrix L_{11}^T is solved in-core, and the current tile of B_1 is written back to the disk.

4.3. But recursive algorithms can also be used. Matrix operations, as the Level 3 BLAS operations, can benefit from recursion. Therefore, recursive templates based on a recursive splitting of the matrix operands have been proposed, studied and experienced in-core [11].
Nevertheless, the out-of-core implementation of these templates is not very efficient. In particular, the amount of I/O operations is larger.

5. Analysis of the I/O costs of the left-looking variant. This section presents a theoretical analysis of data transfers between the disk and the main memory in left-looking Cholesky algorithms. Let A be a matrix of size n, partitioned into $p \times p$ tiles of size $t \times t$. For the sake of simplicity, we assume that n is a multiple of t, and thus $p(n,t) = n/t$. Our aim is to compare different implementations corresponding to different layouts of the memory in terms of I/O operations. We focus on two approaches, the one-tile approach and the two-tiles approach already described in section 3.2. We compute the number of tiles read from and written to the disk by both approaches. In both approaches, two extra buffers store narrow blocks of size $t \times b$. If the size of the tiles, t, is fixed, the width of the narrow blocks has no influence on the amount of I/O operations. Consider now that the main memory is limited to M words. The size of the tiles and of the narrow blocks is constrained by the following inequality:

$$2t^2 + 2tb \leq M,$$

for the two-tiles approach. Hence, if b is small, t can be increased, and thereby the amount of I/O operations is modified. We analyze this effect for the two tiles approach and obtain an asymptotic expression for the gain expected (in terms of I/O operations), when narrow blocks are used.
Next, we compare the one-tile and two-tiles approaches when the size of the main memory is limited to M words. We obtain a simple criterion for choosing one or the other approach, when one wishes to minimize the amount of I/O operations during the factorization.

5.1. Two-tiles approach. We analyze the data transfers performed by the schedule described in §3.2.1. Suppose that $k-1$ columns of tiles of A have already been factored. Matrix A is partitioned as:

$$\left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \tag{5.1}$$

with A_{11} is a tile and A_{22} is a matrix of $(p-k) \times (p-k)$ tiles.
During the k^{th} iteration, the following I/O operations are performed:

- read A_{11}, A_{10} (k tiles).
- write A_{11} (1 tile).
- read A_{21} tile by tile ($k-1$ tiles).
- for each tile of A_{21}, read A_{10} and $k-1$ tiles of A_{20} ($2 \times (k-1)$ tiles).
- write A_{21} ($k-1$ tiles).

Hence, the total number of tiles read $\mathrm{Tr}_{\mathrm{LLT}}^{(2\mathrm{T})}(p)$ is:

$$\mathrm{Tr}_{\mathrm{LLT}}^{(2\mathrm{T})}(p) = \sum_{k=1}^{p} p + 2\,(k-1)\,(p-k) = \frac{p}{3}\left(\frac{p^2}{2} + 2\right).$$

The total number of tiles written $\mathrm{Tw}_{\mathrm{LLT}}^{(2\mathrm{T})}$ is:

$$\mathrm{Tw}_{\mathrm{LLT}}^{(2\mathrm{T})}(p) \;=\; \sum_{k=1}^{p} k \;=\; \frac{p}{2}\,(p+1)\,.$$

5.2. One-tile approach. We examine the number of tiles read and written during the k^{th} iteration of the one-tile approach described in §3.2.2. The matrix A is partitioned as in (5.1).
The tile A_{11} does not stay in memory for the whole iteration: it has to be read for each tile of A_{21}, which results in $p-k$ extra tiles to read. In the end, the number of tiles read, $\mathrm{Tr}_{\mathrm{LLT}}^{(1\mathrm{T})}(p)$, is:

$$\mathrm{Tr}_{\mathrm{LLT}}^{(1\mathrm{T})}(p) \;=\; \mathrm{Tr}_{\mathrm{LLT}}^{(2\mathrm{T})}(p) + \sum_{k=1}^{p}(n-k) \;=\; \frac{p}{6}\,\left(2p^2+3p+1\right),$$

and the number of tiles written, $\mathrm{Tw}_{\mathrm{LLT}}^{(1\mathrm{T})}(p)$, is:

$$\mathrm{Tw}_{\mathrm{LLT}}^{(1\mathrm{T})}(p) \;=\; \mathrm{Tw}_{\mathrm{LLT}}^{(2\mathrm{T})}(p) \;=\; \frac{1}{2}\,p\,(p+1)\,.$$

5.3. Narrow blocks. Some basic linear algebra operations can be implemented using the narrow block technique. The matrix operands that are left unchanged on exit of these operations are partitioned into narrow blocks. Hence, a symmetric rank-k update:

$$C \;\leftarrow\; C - AA^T$$

is implemented as a sequence of narrow rank-b updates: $C - \sum_i A_i\,A_i^T$, where A_i is a narrow subblock of A of width b. A multiply and add:

$$C \;\leftarrow\; C - AB^T$$

is implemented as a sequence of narrow multiply and adds: $C - \sum_i A_i\,B_i^T$, where A_i and B_i are narrow subblocks of A and B of width b.
If the narrow block technique is used, the two-tiles approach requires in-core two full tiles (of size $t \times t$) and two narrow blocks (of size $t \times b$).
Assume now that the size of the main memory is limited to M words. Several memory layouts are possible:

(a) Ignore the narrow block technique, and divide the M words of memory into 4 tiles of equal size $t \times t$, with $t = \sqrt{M/4}$.
(b) Apply the narrow block technique and divide the M words of memory into 2 tiles of equal size $t \times t$, with $\sqrt{M/4} < t < \sqrt{(M/2)}$ and two narrow blocks of size $t \times b$ with $b = \alpha\,t$, $0 < \alpha < 1$, under the constraint: $2\,tb + 2\,t^2 \leq M$.

The number of terms read, with the two-tiles approach is given by:

$$\mathrm{Nr}_{\mathrm{LLT}}^{(2\mathrm{T})}(n,t) \;=\; \mathrm{Tr}_{\mathrm{LLT}}^{(2\mathrm{T})}(n,t) \times t^2.$$

If n is fixed, $t \mapsto \mathrm{Nr}_{\mathrm{LLT}}^{(2\mathrm{T})}(n,t)$ is decreasing with respect to t for $t < n/2$. Hence, the larger t, the lower the number of terms read. The narrow block technique allows to have a larger size of tile t, and thus seems to be more efficient in terms of I/O operations.

The number of terms read during the factorization without using the narrow blocks technique, $\mathrm{Nr}^{(a)}(n)$ is equal to:

$$\mathrm{Nr}^{(a)}(n) \;=\; \mathrm{Nr}_{\mathrm{LLT}}^{(2\mathrm{T})}(n, t = \sqrt{M/4}).$$

Otherwise, if the narrow block technique is used, an asymptotic bound for the number of terms read $\mathrm{Nr}^{(b)}(n)$ is:

$$\mathrm{Nr}_{\mathrm{LLT}}^{(2\mathrm{T})}(n, t = \sqrt{M/2}).$$

Hence, when n is large, we have the asymptotic relation:

$$\mathrm{Nr}^{(b)}(n) \sim \frac{1}{\sqrt{2}} \mathrm{Nr}^{(a)}(n). \tag{5.2}$$

One can expect substantial savings in I/O operations when the narrow blocks technique is used. The numerical experiments reported in §7.1 confirm these asymptotic I/O savings.

5.4. One-tile versus two-tiles implementation. We still consider the factorization of a matrix A of size n with a limited main memory of M words. We again focus on the number of terms read, which dominates the amount of I/O operations. We consider two options:

(a) Partition the main memory into 4 tiles of size $t = \sqrt{M/4}$ and use the two-tiles approach.
(b) Partition the main memory into 3 tiles of size $t = \sqrt{M/3}$ and use the one-tile approach.

We compare these options from the point of view of I/O operations. The number of tiles in A is approximated by $p(n,t) = n/t$.
We denote by $\mathrm{Nr}^{(a)}(n)$ and $\mathrm{Nr}^{(b)}(n)$ the number of terms read respectively with option (a) and option (b).
The number of terms read during the factorization of A of size n with the two-tiles approach is equal to:

$$\mathrm{Nr}_{\mathrm{LLT}}^{(2\mathrm{T})}(n,t) \;=\; \mathrm{Tr}_{\mathrm{LLT}}^{(2\mathrm{T})}(p(n,t))\, t^2.$$

Hence,

$$\mathrm{Nr}^{(a)}(n) \;=\; \mathrm{Nr}_{\mathrm{LLT}}^{(2\mathrm{T})}(n, t = \sqrt{M/4}) \;=\; \frac{n\sqrt{M}}{3}\left(\frac{2n^2}{M} + 1\right).$$

The number of terms read for the factorization of A of size n with the one-tile approach is equal to:

$$\mathrm{Nr}_{\mathrm{LLT}}^{(1\mathrm{T})}(n,t) \;=\; \mathrm{Tr}_{\mathrm{LLT}}^{(1\mathrm{T})}(p(n,t))\, t^2.$$

Hence,

$$\mathrm{Nr}^{(b)}(n) \;=\; \mathrm{Nr}_{\mathrm{LLT}}^{(1\mathrm{T})}(n, t = \sqrt{M/3}) \;=\; \frac{n\sqrt{M}}{6\sqrt{3}}\left(\frac{6n^2}{M} + \frac{3n\sqrt{3}}{\sqrt{M}} + 1\right)$$

Hence, for large n, the one-tile implementation requires less I/O operations than the two-tiles.

Next step is to find out the size n where this transition occurs. We obtain a simple formula criterion for choosing between the one-tile and the two-tiles approach. Finding n such that:

$$\mathrm{Nr}^{(b)}(n) \leq \mathrm{Nr}^{(a)}(n) \tag{5.3}$$

is equivalent to solve

$$N^2(2-\sqrt{3}) - \frac{3}{2}N + (1 - \frac{1}{2\sqrt{3}}) \leq 0$$

where $N = n/\sqrt{M}$. This equation has two roots, $N_1 \approx 0.52$, and $N_2 \approx 5.08$. So, equation (5.3) is satisfied if $N \leq N_1$ or $N \geq N_2$. The first range of n is not very interesting: it corresponds to the case where more than a quarter of the matrix fits within the main memory. The second range corresponds to:

$$n \geq 5.08\sqrt{M}. \tag{5.4}$$

We summarize this result in the following Lemma:

LEMMA 1. *Consider the Cholesky factorization of a matrix A of size n initially stored on disk. The factorization is run on a computer with M words of main memory. If $n \geq 5.08\sqrt{M}$, then the one-tile implementation of the left-looking variant of Cholesky factorization, with a tile size $t = \sqrt{M/3}$, requires less I/O operations. Otherwise, the two-tiles implementation, with a tile size $t = \sqrt{M/4}$, yields less I/O operations.*

Numerical experiments sustaining this claim are performed in §7.2

6. Analysis of the I/O costs of the recursive variant. In this section, we analyze the memory behavior of the recursive variant of the Cholesky factorization. We suppose that 4 tiles can reside simultaneously in the main memory. We first estimate the number of I/Os for the out-of-core BLAS OOC_TRSM and OOC_SYRK. Then, in §6.2, we combine these estimates to obtain the number of tiles read and written by the recursive variant of Cholesky factorization, when the number of tiles of the matrix A is a power of two. Finally, we analyze a hybrid variant of Cholesky factorization.

6.1. Analysis of the out-of-core BLAS. We analyze the data transfer performed by an out-of-core (two-tiles) implementation of OOC_SYRK and OOC_TRSM. All the operands of these routines are out-of-core matrices partitioned into tiles of size $t \times t$.

6.1.1. Out-of-core SYRK. Let A be a symmetric matrix of $p \times p$ tiles, and C a matrix of $p \times k$ tiles. The number of tiles read during the computation of $C - AA^T$ by Algorithm 4.1 is:

$$\mathrm{Tr}_{\mathrm{SYRK}}(p,k) = \sum_{j=1}^{p} 1 + k + (j-1)(2k+1) = kp^2 + \frac{p}{2}(p+1). \tag{6.1}$$

The number of tiles written $\mathrm{Tw}_{\mathrm{SYRK}}(p,k)$ is:

$$\mathrm{Tw}_{\mathrm{SYRK}}(p,k) = \frac{p}{2}(p+1). \tag{6.2}$$

6.1.2. Out-of-core TRSM. Let A be a lower triangular matrix of $p \times p$ tiles, and B a matrix of $m \times p$ tiles. The number of tiles read during the computation of the solution of $X\,A^T = B$ by Algorithm 4.2 is:

$$\mathrm{Tr}_{\,\mathrm{TRSM}}(m,p) \;=\; \sum_{k=1}^{p} 1 + m\,(1 + 2(k-1)) \;=\; m\,p^2 + p. \tag{6.3}$$

The number of tiles written $\mathrm{Tw}_{\,\mathrm{TRSM}}(m,p)$ is:

$$\mathrm{Tw}_{\,\mathrm{TRSM}}(m,p) \;=\; \sum_{k=1}^{p} m \;=\; mp. \tag{6.4}$$

6.2. Analysis of the recursive variant. This paragraph presents an analysis of the data transfers associated to the recursive variant of Cholesky factorization. We consider a matrix A, partitioned into $p \times p$ tiles, and we assume that $p = 2^l$ for some integer l. We denote by $\mathrm{Tr}_{\,\mathrm{RLLT}}(p)$ and $\mathrm{Tw}_{\,\mathrm{RLLT}}(p)$ the number of tiles respectively read and written during the factorization of A by the recursive Algorithm 2.2.
Let us compute $\mathrm{Tr}_{\,\mathrm{RLLT}}(2^l)$ and $\mathrm{Tw}_{\,\mathrm{RLLT}}(2^l)$ by induction.
If $k = 0$, the matrix A is reduced to single tile and the factorization is computed by the in-core Cholesky factorization routine. Hence:

$$\mathrm{Tr}_{\,\mathrm{RLLT}}(1) \;=\; 1, \quad \mathrm{Tw}_{\,\mathrm{RLLT}}(1) \;=\; 1.$$

Let $k \geq 1$. The number of tiles read at level k satisfies the following recurrence:

$$\mathrm{Tr}_{\,\mathrm{RLLT}}(2^k) \;=\; 2\,\mathrm{Tr}_{\,\mathrm{RLLT}}(2^{k-1}) + \mathrm{Tr}_{\,\mathrm{TRSM}}(2^{k-1},2^{k-1}) + \mathrm{Tr}_{\,\mathrm{SYRK}}(2^{k-1},2^{k-1}).$$

The number of tiles written satisfies a similar relation:

$$\mathrm{Tw}_{\,\mathrm{RLLT}}(2^k) \;=\; 2\,\mathrm{Tw}_{\,\mathrm{RLLT}}(2^{k-1}) + \mathrm{Tw}_{\,\mathrm{TRSM}}(2^{k-1},2^{k-1}) + \mathrm{Tw}_{\,\mathrm{SYRK}}(2^{k-1},2^{k-1}).$$

Hence, we have the following formulas, for the factorization of a matrix of 2^l tiles:

$$\begin{aligned}\mathrm{Tr}_{\,\mathrm{RLLT}}(2^l) \;=\; 2^l\,\mathrm{Tr}_{\,\mathrm{RLLT}}(1) &+ \sum_{k=1}^{p} 2^{k-1}\,\mathrm{Tr}_{\,\mathrm{TRSM}}(2^{l-k},2^{l-k}) \\ &+ \sum_{k=1}^{p} 2^{k-1}\,\mathrm{Tr}_{\,\mathrm{SYRK}}(2^{l-k},2^{l-k}),\end{aligned} \tag{6.5}$$

and

$$\begin{aligned}\mathrm{Tw}_{\,\mathrm{RLLT}}(2^l) \;=\; 2^l\,\mathrm{Tw}_{\,\mathrm{RLLT}}(1) &+ \sum_{k=1}^{p} 2^{k-1}\,\mathrm{Tw}_{\,\mathrm{TRSM}}(2^{l-k},2^{l-k}) \\ &+ \sum_{k=1}^{p} 2^{k-1}\,\mathrm{Tw}_{\,\mathrm{SYRK}}(2^{l-k},2^{l-k}).\end{aligned} \tag{6.6}$$

Hence, using (6.1) and (6.3) in (6.5), we obtain the number of tiles read:

$$\mathrm{Tr}_{\,\mathrm{RLLT}}(2^l) \;=\; \frac{2^{3l}}{3} + \frac{2^{2l}}{4} + 2^l\left(\frac{3l}{4} + \frac{5}{12}\right). \tag{6.7}$$

Using (6.2) and (6.4) in (6.6) yields the number of written tiles:

$$\mathrm{Tw}_{\mathrm{RLLT}}(2^l) = \frac{3}{4}2^{2l} + 2^l\left(\frac{l+1}{4}\right). \tag{6.8}$$

The expressions (6.7) and (6.8) give the exact number of tiles read and written for the case $p = 2^l$. If p is not a power of two, these expressions give an approximation of this number. We replace 2^l by p and l by $\ln(p)/\ln(2)$ in (6.7) and (6.8) and obtain:

$$\mathrm{Tr}_{\mathrm{RLLT}}(p) \approx \frac{p^3}{3} + \frac{p^2}{4} + p\left(\frac{3\ln(p)}{4\ln(2)} + \frac{5}{12}\right),$$

and

$$\mathrm{Tw}_{\mathrm{RLLT}}(p) \approx \frac{3p^2}{4} + \frac{p}{4}\left(\frac{\ln(p)}{\ln(2)} + 1\right).$$

6.3. The hybrid recursive/left-looking approach. We investigate the hybrid implementation proposed in §3.3.2. We denote by $\mathrm{Tr}_{\mathrm{HLLT}}(p)$ and $\mathrm{Tw}_{\mathrm{HLLT}}(p)$ the number of tiles read and written for the factorization of a matrix of $p \times p$ tiles, where for the sake of simplicity $p = 2^l$. This hybrid variant is very close to the fully recursive variant analyzed in §6.2. Therefore, we now focus on the differences.
The recursive partitioning of the matrix is stopped at level $l-1$ (instead of l), when the matrix has 2×2 tiles.
The read operations corresponding to the level l are avoided. This results in 3×2^l tiles less to read. At level $l-1$, $3 \times 2^{l-1}$ additional tiles are read (for the factorization of the 2^{l-1} blocks of 2×2 tiles). Finally,

$$\mathrm{Tr}_{\mathrm{HLLT}}(p) = \mathrm{Tr}_{\mathrm{RLLT}}(p) - 3 \times 2^l + 3 \times 2^{l-1} = \frac{2^{3l}}{3} + \frac{2^{2l}}{4} + 2^l\left(\frac{3l}{4} - \frac{13}{12}\right).$$

This variant avoids to write 2^{l-1} tiles. Hence,

$$\mathrm{Tw}_{\mathrm{HLLT}}(p) = \mathrm{Tw}_{\mathrm{RLLT}}(p) - 2^{l-1} = \frac{3}{4}2^{2l} + 2^l\left(\frac{l-1}{4}\right)$$

When p is not necessarily a power of two, the following approximation are satisfied:

$$\mathrm{Tr}_{\mathrm{HLLT}}(p) \approx \frac{p^3}{3} + \frac{p^2}{4} + p\left(\frac{3\ln(p)}{4\ln(2)} - \frac{13}{12}\right)$$
$$\mathrm{Tw}_{\mathrm{HLLT}}(p) \approx \frac{3p^2}{4} + \frac{p}{4}\left(\frac{\ln(p)}{\ln(2)} - 1\right)$$

The hybrid variant slightly improves the number of tiles read and written. Nevertheless, it does not affect the dominant terms.

6.4. Conclusion of the analysis. We compare the number of tiles read from disk and written to the disk by the left-looking variant and by the recursive variant of Cholesky factorization. To make the comparison as fair as possible, we assume that the computation is run on a computer with M words of memory, where $M \geq 4t^2$, where t is the size of the tiles of the matrix. Both implementations are of two-tiles type. We do not take into account improvements like the narrow block technique.

Therefore, we assume that 4 tiles can reside in the memory simultaneously.
We summarize the results of this section in the following theorem:

THEOREM 1. *Let A be an out-of-core matrix partitioned into $p \times p$ tiles, of size $t \times t$. We assume that the computation is run on a computer with M words of memory, where $M \geq 4t^2$.*
With the left-looking variant, the number of tiles read from the disk, $\mathrm{Tr}_{\mathrm{LLT}}^{(2\mathrm{T})}(p)$, and the number of tiles written to the disk, $\mathrm{Tw}_{\mathrm{LLT}}^{(2\mathrm{T})}(p)$ are given by:

$$\mathrm{Tr}_{\mathrm{LLT}}^{(2\mathrm{T})}(p) = \frac{p}{3}\left(\frac{p^2}{2} + 2\right),$$
$$\mathrm{Tw}_{\mathrm{LLT}}^{(2\mathrm{T})}(p) = \frac{p}{2}(p+1).$$

During the factorization of A by a recursive variant of Cholesky algorithm, the number of tiles read from the disk, $\mathrm{Tr}_{\mathrm{RLLT}}(p)$, and the number of tiles written to the disk, $\mathrm{Tw}_{\mathrm{RLLT}}(p)$, are given by:

$$\mathrm{Tr}_{\mathrm{RLLT}}(p) \approx \frac{p^3}{3} + \frac{p^2}{4} + p\left(\frac{3\ln(p)}{4\ln(2)} + \frac{5}{12}\right)$$
$$\mathrm{Tw}_{\mathrm{RLLT}}(p) \approx \frac{3p^2}{4} + \frac{p}{4}\left(\frac{\ln(p)}{\ln(2)} + 1\right).$$

Hence, the recursive variant reads and writes more tiles. Therefore, when one tries to minimize the number of tiles read and written, the left-looking variant is more appropriate.

7. Numerical experiments. We present numerical experiments for the different variants of the out-of-core Cholesky factorization. All variants were implemented in Fortran 90, compiled with Intel's Fortran compiler `ifort` with `-tpp7 -xW` optimization and run on a Pentium 4 XEON based bi-processor running RedHat Linux 3.2.2-5. The 2 CPUs have a clock cycle of 3.05 GHz, 2GB of Ram and a cache size of 512 KB. We use a SCSI disk for the out-of-core storage of matrices. We use Intel's optimized implementation of BLAS, the Math Kernel Libray. Experiments are performed in simple precision complex arithmetic with matrices generated by the discretization of boundary element formulation of the electromagnetic scattering by an object.

7.1. Narrow blocks. We consider the factorization of a matrix A of size $n = 12000$, arithmetically partitioned into tiles of size $t \times t$, by the two-tiles implementation of the Cholesky factorization. The size of the memory is fixed to M words. The memory is divided into 4 buffers, 2 for the storage of full tiles of size $t \times t$ and 2 for the storage of narrow blocks of size $t \times b$, with $b = \alpha\, t$. We investigate different values of the parameter α, to find out an optimal distribution of the memory. The results are shown in Table 7.1.

We verify that the amount of words read from the disk generally decreases with the width of the narrow blocks b. Nevertheless, this decrease is not regular. The amount of words read actually depends on the uniformity of the (arithmetic) matrix partitioning. For $\alpha = 0.17$, the predefined tile size is $t = 655$, but the actual size of the last row and column is 210. This situation is less favorable than taking $\alpha = 0.25$. With $\alpha = 0.25$, the predefined tile size is equal to 633, and the size of the last tiles, 606 is very close to this value. This partitioning generates less I/O operations.
We also verify the estimate (5.2). We compare the ratio $\mathrm{Nr}^{(b)}/\mathrm{Nr}^{(a)}$ to the predicted

	$M = 1.10^6$ words					$M = 4.10^6$ words				
α	t	b	words read ($\times 10^6$)	read time (s)	total time (s)	t	b	words read ($\times 10^6$)	read time (s)	total time (s)
1	500	500	1156	115	398	1000	1000	584	58	321
0.5	578	289	1017	101	400	1155	578	548	55	331
0.33	613	205	973	98	452	1225	409	493	49	341
0.25	633	159	918	91	489	1265	317	498	50	358
0.17	655	110	931	93	541	1310	219	504	50	384
0.10	675	68	874	85	518	1349	135	444	44	392
0.014	707	1	824	103	1858	1413	2	496	48	1336

TABLE 7.1

Impact of the narrow blocks technique on the performance characteristics of the two-tiles implementation of Cholesky factorization. The matrix size is $n = 12000$. The memory of M words is divided into 4 buffers, 2 for the storage of tiles of size $t \times t$ and 2 for the storage of narrow blocks of size $t \times b$. Several values of $\alpha = b/t$ are experienced.

value $1/\sqrt{2}$ for a small α. When $M = 1 \times 10^6$ words, and $b = 1$, we obtain:

$$\frac{\mathrm{Nr}^{(b)}(12000)}{\mathrm{Nr}^{(a)}(12000)}\sqrt{2} = \frac{824 \times \sqrt{2}}{1156} = 1.0080,$$

When $M = 4 \times 10^6$ words, the choice $\alpha = 0.014$ leads to a non-uniform partitioning (the size of the last tiles is 688 whereas the tile size is 1414), and to a high amount of I/O operations. Therefore, for $M = 4 \times 10^6$ words, we compute $\mathrm{Nr}^{(b)}/\mathrm{Nr}^{(a)}$ for $\alpha = 0.10$:

$$\frac{\mathrm{Nr}^{(b)}(12000)}{\mathrm{Nr}^{(a)}(12000)}\sqrt{2} = \frac{444 \times \sqrt{2}}{584} = 1.075,$$

which again confirms prediction (5.2).
We observe an unexpected behavior: despite of the decrease of the amount of read words, the total time taken by the factorization increases as α diminishes. The optimal value for α, from the point of view of elapsed time, is $\alpha = 1$, that is when narrow blocks are not used. A possible explanation for this behavior is that the computation is better scheduled when performed on square matrices. Thus, narrow blocks do not allow to achieve better performances. Therefore, they are no more used in our numerical experiments.

7.2. One-tile versus two-tiles approach for the left-looking variant. We consider the factorization of a matrix A of size $n = 12000$, arithmetically partitioned into tiles of size $t \times t$ by the left-looking out-of-core Cholesky algorithm. The amount of available memory is limited to M words. We show on Figure 7.1 the number of terms read with respect to the size of the memory available. We verify our theoretical prediction: if the memory available is larger than M_0 words, with

$$M_0 = (n/5.08)^2,$$

that is for simple precision complexes, 42.5 MB, the two-tiles approach requires less I/O operations. Nevertheless, though there is a difference in the amount of I/O operations, none of the variant is significantly faster than the other, as shown on the right graphic of Figure 7.1. The difference in the elapsed times is about 1 second for a total elapsed time of 270 seconds.

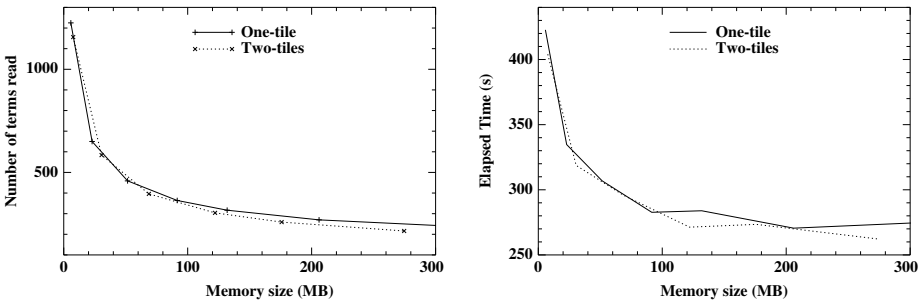

FIGURE 7.1. *Factorization of a simple precision complex matrix of size* 12000*, by the one-tile and two-tiles implementation of the left-looking variant. Different sizes of tiles t are experienced, corresponding to different memory size requirements. The memory size in MB is equal to $M \times 8/(1024)^2$, where M is the memory size in words. The left part shows the measured number of elements read during the factorization ($\times 10^6$). The right part shows the elapsed time for the factorization.*

	Left-looking algorithm							
	One-tile implementation				Two-tiles implementation			
t	Tiles read	Tiles written	I/O time (s)	Total time (s)	Tiles read	Tiles written	I/O time (s)	Total time (s)
375	11440	528	175	495	10944	528	166	483
750	1496	136	97	389	1376	136	90	381
1500	204	36	60	325	176	36	53	318
3000	30	10	46	306	24	10	42	300
	Recursive algorithm							
	Standard implementation				Hybrid implementation			
t	Tiles read	Tiles written	I/O time (s)	Total time (s)	Tiles read	Tiles written	I/O time (s)	Total time (s)
375	11312	816	183	501	11264	800	205	523
750	1484	212	106	396	1460	204	115	405
1500	208	56	70	335	196	52	71	337
3000	33	15	59	319	27	13	53	311

TABLE 7.2

Factorization of a matrix of simple precision complex matrix of size $N = 12000$, partitioned into tiles of size $t \times t$, for different values of t. We compare two recursive variants (standard and hybrid), and two left-looking variants (one-tile and two-tiles). The recursive variants and the two-tiles left-looking variant use 4 buffers of $t \times t$ words in-core, whereas the one-tile variant of the left-looking algorithm uses 3 buffers of $t \times t$ words in-core. We measure the number of tiles read and written during the factorization, the total time elapsed, and the time taken by the I/Os (read+write). Times are reported in seconds.

7.3. Comparison of the loop-based and recursive implementations. In this paragraph, we compare the performance characteristics of the one-tile and two-tiles implementation of the left-looking variant, and the fully recursive and hybrid implementations of the recursive variant of Cholesky algorithm. We factorize a simple precision complex matrices of size $n = 12000$ and $n = 48000$, and the results are respectively reported in Table 7.2 ans Table 7.3. For the tile sizes we consider, arithmetic and recursive partitioning of the matrices are identical.

We observe a quite surprising result: the fully recursive implementation is faster

	Left-looking algorithm							
	One-tile implementation				Two-tiles implementation			
t	Tiles read	Tiles written	I/O time (s)	Total time (s)	Tiles read	Tiles written	I/O time (s)	Total time (s)
750	89440	2080	5645	23655	87424	2080	5462	23489
1500	11440	528	3039	19239	10944	528	2903	19077
3000	1496	136	1898	17608	1376	136	1781	17365
	Recursive algorithm							
	Standard implementation				Hybrid implementation			
t	Tiles read	Tiles written	I/O time (s)	Total time (s)	Tiles read	Tiles written	I/O time (s)	Total time (s)
750	88720	3184	5732	23746	88624	3152	6411	24467
1500	11312	816	3100	19298	11264	800	3466	19671
3000	1484	212	1952	17583	1460	204	2164	17724

TABLE 7.3

Factorization of a matrix of simple precision complex matrix of size $N = 48000$, partitioned into tiles of size $t \times t$, for different values of t. We compare two recursive variants (standard and hybrid), and two left-looking variants (one-tile and two-tiles). The recursive variants and the two-tiles left-looking variant use 4 buffers of $t \times t$ words in-core, whereas the one-tile variant of the left-looking algorithm uses 3 buffers of $t \times t$ words in-core. We measure the number of tiles read and written during the factorization, the total time elapsed, and the time taken by the I/Os (read+write). Times are reported in seconds.

than the hybrid implementation, although the amount of I/O operations is higher.

8. Conclusion and perspectives. This paper analyses and compares the out-of-core implementation of different variants of the partitioned Cholesky factorization algorithm. The same partitioning of the matrix to factorize is used for all variants, to allow a fair comparison. Our theoretical analysis shows that the amount of I/O operations is lower when a left-looking algorithm is used. Our numerical experiments confirm this result. Moreover, in our tests, the two-tiles implementation of the left-looking algorithm is the fastest variant. It is the equivalent in complex arithmetic of the (sequential) POOCLAPACK routine for the Cholesky decomposition of real positive definite matrices [13]. Some improvements could still be added to this solver, as prefetching the tiles, to overlap I/O operations and computation, as is performed in [21]. Finally, in order to tackle larger size problems, we are currently working on the parallelization of the solver on a distributed architecture.

REFERENCES

[1] B. S. Andersen, J. A. Gunnels, F. G. Gustavson, J. K. Reid, and J. Wasniewski. A fully portable high performance minimal storage hybrid format Cholesky algorithm. Technical Report RAL-TR-2004-017, CCLRC Rutherford Appleton Laboratory, May 2004.

[2] B. S. Andersen, J. Wasniewski, and F. G. Gustavson. A recursive formulation of Cholesky factorization of a matrix in packed storage. *ACM Trans. Math. Softw.*, 27(2):214–244, 2001.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, third edition, 1999.

[4] M. Baboulin, L. Giraud, and S. Gratton. A parallel distributed solver for large dense symmetric systems: applications to geodesy and electromagnetism problems. Technical Report TR/PA/04/16, CERFACS, Toulouse, France, 2004.

[5] A. Bendali. Numerical analysis of the exterior boundary value problem for the time-harmonic Maxwell equations by a boundary finite element method, part 1: The continuous problem, part 2: The discrete problem. *Math. Comp.*, 43(167):29–68, 1984.

[6] N. Béreux. A Cholesky algorithm for some complex symmetric systems. Technical Report 515, CMAP, École Polytechnique, October 2003.

[7] S. H. Christiansen. Discrete Fredholm properties and convergence estimates for the electric field integral equation. *Math. Comp.*, 73(245):143–167, 2004.

[8] S. H. Christiansen and J.-C. Nédélec. A Preconditioner for the Electric Field Integral Equation based on Calderon formulas. *SIAM J. Numer. Anal.*, 40(3):1100–1135, 2002.

[9] E. Darve. The Fast Multipole Method (I) : Error Analysis and Asymptotic Complexity. *SIAM J. Numer. Anal.*, 38(1):98–128, 2000.

[10] E. F. D'Azevedo and J. Dongarra". "the Design and Implementation of the Parallel Out-of-Core ScaLAPACK LU, QR and Cholesky Factorization Routines". *Concurrency: Practice and Experience*, 12(15):1481–1493, 2000.

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

[12] J. A. Gunnels and R. A. Van de Geijn. Developping linear algebra algorithms: A collection of class projects. FLAME Working Note 3, Department of Computer Sciences, The University of Texas, May 2001.

[13] B. C. Gunter, W. C. Reiley, and R. A. Van De Geijn. Parallel out-of-core Cholesky and QR factorizations with POOCLAPACK. PLAPACK Working Note 12, Department of Computer Sciences, The University of Texas at Austin, September 2000.

[14] F. G. Gustavson and I. Jonsson. Minimal storage high performance cholesky factorization via blocking and recursion. *IBM J. Res. Develop.*, 44:823–849, 2000.

[15] D. Irony, G. Shklarski, and S. Toledo. Parallel and fully recursive multifrontal sparse Cholesky. *Future Generation Computer Systems*, 20:425–440, 2004.

[16] T. Joffrain, E. S. Quintana-Ortí, and R. A. van de Geijn. Rapid development of high-performance out-of-core solvers. In *PARA*, pages 413–422, 2004.

[17] J.-C. Nédélec. *Acoustic and electromagnetic equations: Integral representations for harmonic problems*, volume 144 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2001.

[18] W. C. Reiley and R. A. van de Geijn. POOCLAPACK: Parallel out-of-core linear algebra package. Technical Report CS-TR-99-33, Department of Computer Sciences, The University of Texas at Austin, 1, 1999.

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

[20] S. Toledo. A survey of out-of-core algorithms in numerical linear algebra. In J. Abello and J. S. Vitter, editors, *External Memory Algorithms and Visualization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 161–180. American Mathematical Society Press, Providence, RI, 1999.

[21] S. Toledo and F. G. Gustavson. The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computations. In *IOPADS '96: Proceedings of the fourth workshop on I/O in parallel and distributed systems*, pages 28–40, New York, NY, USA, 1996. ACM Press.

[22] R. Van de Geijn and E. Quintana-Ortí. The science of programming matrix computations, 2005.

[23] R. Wesley. Efficient Parallel Out-of-core Implementation of the Cholesky Factorization. PLAPACK Working Note 11, Department of Computer Sciences, The University of Texas at Austin, December 1999.