Notes on Cholesky Factorization

Robert A. van de Geijn
Department of Computer Science
The University of Texas
Austin, TX 78712
rvdg@cs.utexas.edu

October 24, 2014

1 Definition and Existence

This operation is only defined for Hermitian positive definite matrices:

Definition 1. *A matrix $A \in \mathbb{C}^{m\times m}$ is Hermitian positive definite (HPD) if and only if it is Hermitian ($A^H = A$) and for all nonzero vectors $x \in \mathbb{C}^m$ it is the case that $x^H Ax > 0$. If in addition $A \in \mathbb{R}^{m\times m}$ then A is said to be symmetric positive definite (SPD).*

(If you feel uncomfortable with complex arithmetic, just replace the word "Hermitian" with "Symmetric" in this document and the Hermitian transpose operation, ^{H}, with the transpose operation, ^{T}.

Example 2. *Consider the case where $m = 1$ so that A is a real scalar, α. Notice that then A is SPD if and only if $\alpha > 0$. This is because then for all nonzero $\chi \in \mathbb{R}$ it is the case that $\alpha\chi^2 > 0$.*

First some exercises:

Exercise 3. *Let $B \in \mathbb{C}^{m\times n}$ have linearly independent columns. Prove that $A = B^H B$ is HPD.*

Exercise 4. *Let $A \in \mathbb{C}^{m\times m}$ be HPD. Show that its diagonal elements are real and positive.*

We will prove the following theorem in Section 4:

Theorem 5. (Cholesky Factorization Theorem) *Given a HPD matrix A there exists a lower triangular matrix L such that $A = LL^H$.*

Obviously, there similarly exists an upper triangular matrix U such that $A = U^H U$ since we can choose $U^H = L$.

The lower triangular matrix L is known as the Cholesky factor and LL^H is known as the Cholesky factorization of A. It is unique if the diagonal elements of L are restricted to be positive.

The operation that overwrites the lower triangular part of matrix A with its Cholesky factor will be denoted by $A := \text{CHOL}(A)$, which should be read as "A becomes its Cholesky factor." Typically, only the lower (or upper) triangular part of A is stored, and it is that part that is then overwritten with the result. In this discussion, we will assume that the lower triangular part of A is stored and overwritten.

2 Application

The Cholesky factorization is used to solve the linear system $Ax = y$ when A is HPD: Substituting the factors into the equation yields $LL^H x = y$. Letting $z = L^H x$,

$$Ax = L \underbrace{(L^H x)}_{z} = Lz = y.$$

Thus, z can be computed by solving the triangular system of equations $Lz = y$ and subsequently the desired solution x can be computed by solving the triangular linear system $L^H x = z$.

3 An Algorithm

The most common algorithm for computing $A := \text{CHOL}(A)$ can be derived as follows: Consider $A = LL^H$. Partition

$$A = \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right) \quad \text{and} \quad L = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right). \tag{1}$$

Remark 6. *We adopt the commonly used notation where Greek lower case letters refer to scalars, lower case letters refer to (column) vectors, and upper case letters refer to matrices. The $\star$ refers to a part of A that is neither stored nor updated.*

By substituting these partitioned matrices into $A = LL^H$ we find that

$$\begin{aligned} \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right) &= \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)^H = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c} \bar{\lambda}_{11} & l_{21}^H \\ \hline 0 & L_{22}^H \end{array}\right) \\ &= \left(\begin{array}{c|c} |\lambda_{11}|^2 & \star \\ \hline \bar{\lambda}_{11} l_{21} & l_{21} l_{21}^H + L_{22} L_{22}^H \end{array}\right), \end{aligned}$$

from which we conclude that

$$\left(\begin{array}{c|c} |\lambda_{11}| = \sqrt{\alpha_{11}} & \star \\ \hline l_{21} = a_{21}/\bar{\lambda}_{11} & L_{22} = \text{CHOL}(A_{22} - l_{21} l_{21}^H) \end{array}\right).$$

These equalities motivate the algorithm

1. Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right)$.
2. Overwrite $\alpha_{11} := \lambda_{11} = \sqrt{\alpha_{11}}$. (Picking $\lambda_{11} = \sqrt{\alpha_{11}}$ makes it positive and real, and ensures uniqueness.)
3. Overwrite $a_{21} := l_{21} = a_{21}/\lambda_{11}$.
4. Overwrite $A_{22} := A_{22} - l_{21} l_{21}^H$ (updating only the lower triangular part of A_{22}). This operation is called a *symmetric rank-1 update.*
5. Continue with $A = A_{22}$. (Back to Step 1.)

Remark 7. *Similar to the* `tril` *function in Matlab, we use* $\text{TRIL}(B)$ *to denote the lower triangular part of matrix B.*

4 Proof of the Cholesky Factorization Theorem

In this section, we partition A as in (1):

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array}\right).$$

The following lemmas are key to the proof:

Lemma 8. *Let $A \in \mathbb{C}^{nxn}$ be HPD. Then α_{11} is real and positive.*

Proof: This is special case of Exercise 1.

Lemma 9. *Let $A \in \mathbb{C}^{m\times m}$ be HPD and $l_{21} = a_{21}/\sqrt{\alpha_{11}}$. Then $A_{22} - l_{21}l_{21}^H$ is HPD.*

Proof: Since A is Hermitian so are A_{22} and $A_{22} - l_{21}l_{21}^H$.

Let $x_2 \in \mathbb{C}^{(n-1)\times(n-1)}$ be an arbitrary nonzero vector. Define $x = \left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array}\right)$ where $\chi_1 = -a_{21}^H x_2/\alpha_{11}$. Then, since $x \neq 0$,

$$\begin{array}{rcl} 0 < x^H Ax & = & \left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array}\right)^H \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array}\right) \left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array}\right) \\ & = & \left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array}\right)^H \left(\begin{array}{c} \alpha_{11}\chi_1 + a_{21}^H x_2 \\ \hline a_{21}\chi_1 + A_{22}x_2 \end{array}\right) \\ & = & \alpha_{11}|\chi_1|^2 + \bar{\chi}_1 a_{21}^H x_2 + x_2^H a_{21}\chi_1 + x_2^H A_{22}x_2 \\ & = & \alpha_{11}\frac{a_{21}^H x_2}{\alpha_{11}}\frac{x_2^H a_{21}}{\alpha_{11}} - \frac{x_2^H a_{21}}{\alpha_{11}}a_{21}^H x_2 - x_2^H a_{21}\frac{a_{21}^H x_2}{\alpha_{11}} + x_2^H A_{22}x_2 \\ & = & x_2^H(A_{22} - \frac{a_{21}a_{21}^H}{\alpha_{11}})x_2 \quad \text{(since } x_2^H a_{21}a_{21}^H x_2 \text{ is real and hence equals } a_{21}^H x_2 x_2^H a_{21}\text{)} \\ & = & x_2^H(A_{22} - l_{21}l_{21}^H)x_2. \end{array}$$

We conclude that $A_{22} - l_{21}l_{21}^H$ is HPD.

Proof: of the **Cholesky Factorization Theorem**
Proof by induction.

Base case: $n = 1$. Clearly the result is true for a 1×1 matrix $A = \alpha_{11}$: In this case, the fact that A is HPD means that α_{11} is real and positive and a Cholesky factor is then given by $\lambda_{11} = \sqrt{\alpha_{11}}$, with uniqueness if we insist that λ_{11} is positive.

Inductive step: Assume the result is true for HPD matrix $A \in \mathbb{C}^{(n-1)\times(n-1)}$. We will show that it holds for $A \in \mathbb{C}^{n\times n}$. Let $A \in \mathbb{C}^{n\times n}$ be HPD. Partition A and L as in (1) and let $\lambda_{11} = \sqrt{\alpha_{11}}$ (which is well-defined by Lemma 4), $l_{21} = a_{21}/\lambda_{11}$, and $L_{22} = \text{CHOL}(A_{22} - l_{21}l_{21}^H)$ (which exists as a consequence of the Inductive Hypothesis and Lemma 4). Then L is the desired Cholesky factor of A.

By the principle of mathematical induction, the theorem holds.

5 Blocked Algorithm

In order to attain high performance, the computation is cast in terms of matrix-matrix multiplication by so-called blocked algorithms. For the Cholesky factorization a blocked version of the algorithm can be derived

by partitioning

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right) \quad \text{and} \quad L \rightarrow \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right),$$

where A_{11} and L_{11} are $b \times b$. By substituting these partitioned matrices into $A = LL^H$ we find that

$$\left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right)^H = \left(\begin{array}{c|c} L_{11}L_{11}^H & \star \\ \hline L_{21}L_{11}^H & L_{21}L_{21}^H + L_{22}L_{22}^H \end{array}\right).$$

From this we conclude that

$$\left(\begin{array}{c|c} L_{11} = \textsc{Chol}(A_{11}) & \star \\ \hline L_{21} = A_{21}L_{11}^{-H} & L_{22} = \textsc{Chol}(A_{22} - L_{21}L_{21}^H) \end{array}\right).$$

An algorithm is then described by the steps

1. Partition $A \rightarrow \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array}\right)$, where A_{11} is $b \times b$.
2. Overwrite $A_{11} := L_{11} = \textsc{Chol}(A_{11})$.
3. Overwrite $A_{21} := L_{21} = A_{21}L_{11}^{-H}$.
4. Overwrite $A_{22} := A_{22} - L_{21}L_{21}^H$ (updating only the lower triangular part).
5. Continue with $A = A_{22}$. (Back to Step 1.)

Remark 10. *The Cholesky factorization $A_{11} := L_{11} = \textsc{Chol}(A_{11})$ can be computed with the unblocked algorithm or by calling the blocked Cholesky factorization algorithm recursively.*

Remark 11. *Operations like $L_{21} = A_{21}L_{11}^{-H}$ are computed by solving the equivalent linear system with multiple right-hand sides $L_{11}L_{21}^H = A_{21}^H$.*

6 Alternative Representation

When explaining the above algorithm in a classroom setting, invariably it is accompanied by a picture sequence like the one in Figure 1(left) and the (verbal) explanation:

Beginning of iteration: At some stage of the algorithm (Top of the loop), the computation has moved through the matrix to the point indicated by the thick lines. Notice that we have finished with the parts of the matrix that are in the top-left, top-right (which is not to be touched), and bottom-left quadrants. The bottom-right quadrant has been updated to the point where we only need to perform a Cholesky factorization of it.

Repartition: We now repartition the bottom-right submatrix to expose α_{11}, a_{21}, and A_{22}.

Update: α_{11}, a_{21}, and A_{22} are updated as discussed before.

End of iteration: The thick lines are moved, since we now have completed more of the computation, and only a factorization of A_{22} (which becomes the new bottom-right quadrant) remains to be performed.

Continue: The above steps are repeated until the submatrix A_{BR} is empty.

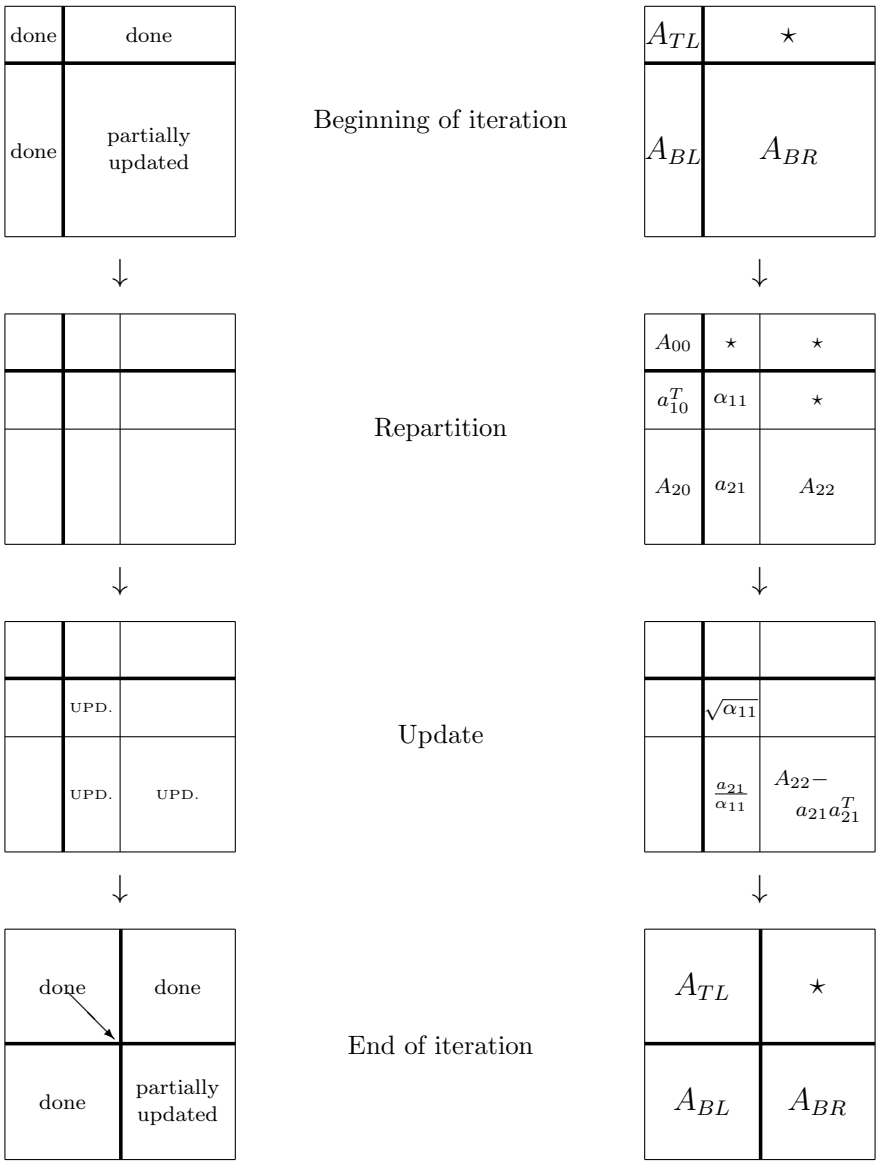

Figure 1: Left: Progression of pictures that explain Cholesky factorization algorithm. Right: Same pictures, annotated with labels and updates. (Picture modified from a similar one in [9].

To motivate our notation, we annotate this progression of pictures as in Figure 1 (right). In those pictures, "T", "B", "L", and "R" stand for "Top", "Bottom", "Left", and "Right", respectively. This then motivates the format of the algorithm in Figure 2 (left). It uses what we call the FLAME notation for representing algorithms [9, 8, 12]. A similar explanation can be given for the blocked algorithm, which is given in Figure 2 (right). In the algorithms, $m(A)$ indicates the number of rows of matrix A.

Remark 12. *The indices in our more stylized presentation of the algorithms are subscripts rather than indices in the conventional sense.*

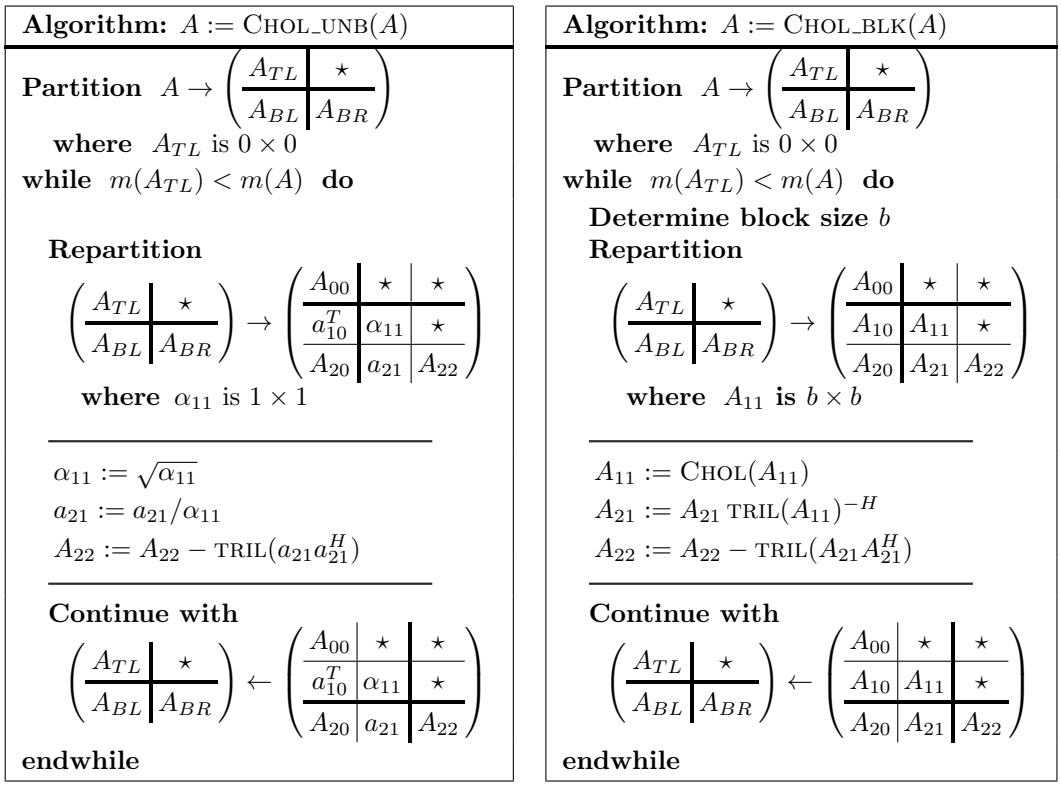

Figure 2: Unblocked and blocked algorithms for computing the Cholesky factorization in FLAME notation.

Remark 13. *The notation in Figs. 1 and 2 allows the contents of matrix A at the beginning of the iteration to be formally stated:*

$$A = \left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L_{TL} & \star \\ \hline L_{BL} & \hat{A}_{BR} - \text{TRIL}(L_{BL}L_{BL}^H) \end{array}\right),$$

where $L_{TL} = \text{CHOL}(\hat{A}_{TL})$, $L_{BL} = \hat{A}_{BL}L_{TL}^{-H}$, *and* $\hat{A}_{TL}, \hat{A}_{BL}$ *and* $\hat{A}_{BR}$ *denote the original contents of the quadrants* A_{TL}, A_{BL} *and* A_{BR}, *respectively.*

Exercise 14. *Implement the Cholesky factorization with M-script.*

7 Cost

The cost of the Cholesky factorization of $A \in \mathbb{C}^{m\times m}$ can be analyzed as follows: In Figure 2 (left) during the kth iteration (starting k at zero) A_{00} is $k \times k$. Thus, the operations in that iteration cost

- $\alpha_{11} := \sqrt{\alpha_{11}}$: negligible when k is large.
- $a_{21} := a_{21}/\alpha_{11}$: approximately $(m-k-1)$ flops.
- $A_{22} := A_{22} - \text{TRIL}(a_{21}a_{21}^H)$: approximately $(m-k-1)^2$ flops. (A rank-1 update of all of A_{22} would have cost $2(m-k-1)^2$ flops. Approximately half the entries of A_{22} are updated.)

Thus, the total cost in flops is given by

$$\begin{aligned} C_{\text{Chol}}(m) &\approx \underbrace{\sum_{k=0}^{m-1}(m-k-1)^2}_{\text{(Due to update of } A_{22})} + \underbrace{\sum_{k=0}^{m-1}(m-k-1)}_{\text{(Due to update of } a_{21})} \\ &= \sum_{j=0}^{m-1} j^2 + \sum_{j=0}^{m-1} j \approx \frac{1}{3}m^3 + \frac{1}{2}m^2 \approx \frac{1}{3}m^3 \end{aligned}$$

which allows us to state that (obvious) most computation is in the update of A_{22}. It can be shown that the blocked Cholesky factorization algorithm performs exactly the same number of floating point operations.

Comparing the cost of the Cholesky factorization to that of the LU factorization from "Notes on LU Factorization" we see that taking advantage of symmetry cuts the cost approximately in half.

8 Solving the Linear Least-Squares Problem via the Cholesky Factorization

Recall that if $B \in \mathbb{C}^{m\times n}$ has linearly independent columns, then $A = B^H B$ is HPD. Also, recall from "Notes on Linear Least-Squares" that the solution, $x \in \mathbb{C}^n$ to the linear least-squares (LLS) problem

$$\|Bx - y\|_2 = \min_{z\in\mathbb{C}^n} \|Bz - y\|_2$$

equals the solution to the normal equations

$$\underbrace{B^H B}_{A}\, x = \underbrace{B^H y}_{\hat{y}} .$$

This makes it obvious how the Cholesky factorization can (and often is) used to solve the LLS problem.

Exercise 15. *Consider $B \in \mathbb{C}^{m\times n}$ with linearly independent columns. Recall that B has a QR factorizaiton, $B = QR$ where Q has orthonormal columns and R is an upper triangular matrix with positive diagonal elements. How are the Cholesky factorization of $B^H B$ and the QR factorization of B related?*

Proof:

$$B^H B = (QR)^H QR = R^H \underbrace{Q^H Q}_{I} R = \underbrace{R^H}_{L}\ \underbrace{R}_{L^H} .$$

9 Other Cholesky Factorization Algorithms

There are actually three different unblocked and three different blocked algorithms for computing the Cholesky factorization. The algorithms we discussed in this note are sometimes called the *right-looking* algorithm. Systematic derivation of all these algorithms, as well as their blocked counterparts, are given in Chapter 6 of [12].

10 Implementing the Cholesky Factorization with the (Traditional) BLAS

The Basic Linear Algebra Subprograms (BLAS) are an interface to commonly used fundamental linear algebra operations. In this section, we illustrate how the unblocked and blocked Cholesky factorization

	Algorithm	Code

Simple	**for** $j = 1 : n$
	$\alpha_{j,j} := \sqrt{\alpha_{j,j}}$
	
	for $i = j + 1 : n$
	$\alpha_{i,j} := \alpha_{i,j}/\alpha_{j,j}$
	endfor
	
	for $k = j + 1 : n$
	for $i = k : n$
	$\alpha_{i,k} := \alpha_{i,k} - \alpha_{i,j}\alpha_{k,j}$
	endfor
	endfor
	endfor
		`do j=1, n`
		`  A( j,j ) = sqrt( A( j,j ) )`
		
		`  do i=j+1,n`
		`  A( i,j ) = A( i,j ) / A( j,j )`
		`  enddo`
		
		`  do k=j+1,n`
		`  do i=k,n`
		`  A( i,k ) = A( i,k ) - A( i,j ) * A( k,j )`
		`  enddo`
		`  enddo`
		`enddo`

Vector-vector	**for** $j = 1 : n$
	$\alpha_{j,j} := \sqrt{\alpha_{j,j}}$
	
	$\alpha_{j+1:n,j} := \alpha_{j+1:n,j}/\alpha_{j,j}$
	
	for $k = j + 1 : n$
	$\alpha_{k:n,k} := -\alpha_{k,j}\alpha_{k:n,j} + \alpha_{k:n,k}$
	endfor
	endfor
		`do j=1, n`
		`  A( j,j ) = sqrt( A( j,j ) )`
		
		`  call dscal( n-j, 1.0d00 / A( j,j ), A( j+1, j ), 1 )`
		
		`  do k=j+1,n`
		`  call daxpy( n-k+1, -A( k,j ), A(k,j), 1, A( k, k ), 1 )`
		`  enddo`
		`enddo`

Matrix-vector	**for** $j = 1 : n$
	$\alpha_{j,j} := \sqrt{\alpha_{j,j}}$
	
	$\alpha_{j+1:n,j} := \alpha_{j+1:n,j}/\alpha_{j,j}$
	
	$\alpha_{j+1:n,j+1:n} :=$
	$-\texttt{tril}(\alpha_{j+1:n,j}\ \alpha_{j+1:n,j}^T) + \alpha_{j+1:n,j+1:n}$
	endfor
		`do j=1, n`
		`  A( j,j ) = sqrt( A( j,j ) )`
		
		`  call dscal( n-j, 1.0d00 / A( j,j ), A( j+1, j ), 1 )`
		
		`  call dsyr( 'lower triangular',`
		`  n-j, -1.0, A(j+1,j), 1, A( j+1, j+1 ), lda )`
		`enddo`

FLAME Notation	**Partition** $A \rightarrow \left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right)$
	where A_{TL} is 0×0
	
	while $m(A_{TL}) < m(A)$ **do**
	Determine block size *blank*
	Repartition
	$\left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
	where α_{11} **is** 1×1
	
	$\alpha_{11} := \sqrt{\alpha_{11}}$
	$a_{21} := a_{21}/\alpha_{11}$
	$A_{22} := A_{22} - \text{TRIL}(a_{21}a_{21}^H)$
	
	Continue with
	$\left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
	endwhile
		`int Chol_unb_var3( FLA_Obj A )`
		`{`
		`  FLA_Obj ATL,  ATR,  A00,  a01,  A02,`
		`  ABL,  ABR,  a10t, alpha11, a12t,`
		`  A20,  a21,  A22;`
		
		`  FLA_Part_2x2( A,  &ATL, &ATR,`
		`  &ABL, &ABR,  0, 0, FLA_TL );`
		
		`  while ( FLA_Obj_length( ATL ) < FLA_Obj_length( A ) ){`
		`  FLA_Repart_2x2_to_3x3(`
		`  ATL, /**/ ATR,  &A00,  /**/ &a01,  &A02,`
		`  /* ************ */  /* ********** */`
		`  &a10t, /**/ &alpha11, &a12t,`
		`  ABL, /**/ ABR,  &A20,  /**/ &a21,  &A22,`
		`  1, 1, FLA_BR );`
		`  /*----------*/`
		`  FLA_Sqrt( alpha11 );`
		`  FLA_Invscal( alpha11, a21 );`
		`  FLA_Syr( FLA_LOWER_TRIANGULAR, FLA_MINUS_ONE, A21, A22 );`
		`  /*----------*/`
		`  FLA_Cont_with_3x3_to_2x2(`
		`  &ATL, /**/ &ATR,  A00,  a01,  /**/ A02,`
		`  a10t, alpha11, /**/ a12t,`
		`  /* ************ */  /* ********** */`
		`  &ABL, /**/ &ABR,  A20,  a21,  /**/ A22,`
		`  FLA_TL );`
		`  }`
		`  return FLA_SUCCESS;`
		`}`

Figure 3: Various representations of the right-looking algorithm.

algorithms can be implemented in terms of the BLAS. The explanation draws from the entry we wrote for the BLAS in the Encyclopedia of Parallel Computing [11].

10.1 What are the BLAS?

The BLAS interface [10, 6, 5] was proposed to support portable high-performance implementation of applications that are matrix and/or vector computation intensive. The idea is that one casts computation in terms of the BLAS interface, leaving the architecture-specific optimization of that interface to an expert.

10.2 A simple implementation in Fortran

We start with a simple implementation in Fortran. A simple algorithm that does not use BLAS and the corresponding code in given in the row labeled "Simple" in Figure 3. This sets the stage for our explaination of how the algorithm and code can be represented with vector-vector, matrix-vector, and matrix-matrix operations, and the corresponding calls to BLAS routines.

10.3 Implemention with calls to level-1 BLAS

The first BLAS interface [10] was proposed in the 1970s when vector supercomputers like the early Cray architectures reigned. On such computers, it sufficed to cast computation in terms of vector operations. As long as memory was accessed mostly contiguously, near-peak performance could be achieved. This interface is now referred to as the Level-1 BLAS. It was used for the implementation of the first widely used dense linear algebra package, LINPACK [4].

Let x and y be vectors of appropriate length and α be scalar. In this and other notes we have *vector-vector operations* such as scaling of a vector ($x := \alpha x$), inner (dot) product ($\alpha := x^T y$), and scaled vector addition ($y := \alpha x + y$). This last operation is known as an `axpy`, which stands for alpha times x plus y.

Our Cholesky factorization algorithm expressed in terms of such vector-vector operations and the corresponding code are given in Figure 3 in the row labeled "Vector-vector". *If* the operations supported by `dscal` and `daxpy` achieve high performance on a target archecture (as it was in the days of vector supercomputers) *then* so will the implementation of the Cholesky factorization, since it casts most computation in terms of those operations. Unfortunately, vector-vector operations perform $O(n)$ computation on $O(n)$ data, meaning that these days the bandwidth to memory typically limits performance, since retrieving a data item from memory is often more than an order of magnitude more costly than a floating point operation with that data item.

We summarize information about level-1 BLAS in Figure 4.

10.4 Matrix-vector operations (level-2 BLAS)

The next level of BLAS supports operations with matrices and vectors. The simplest example of such an operation is the matrix-vector product: $y := Ax$ where x and y are vectors and A is a matrix. Another example is the computation $A_{22} = -a_{21}a_{21}^T + A_{22}$ (symmetric rank-1 update) in the Cholesky factorization. Here only the lower (or upper) triangular part of the matrix is updated, taking advantage of symmetry.

The use of symmetric rank-1 update is illustrated in Figure 3, in the row labeled "Matrix-vector". There `dsyr` is the routine that implements a double precision symmetric rank-1 update. Readability of the code is improved by casting computation in terms of routines that implement the operations that appear in the algorithm: `dscal` for $a_{21} = a_{21}/\alpha_{11}$ and `dsyr` for $A_{22} = -a_{21}a_{21}^T + A_{22}$.

If the operation supported by `dsyr` achieves high performance on a target archecture (as it was in the days of vector supercomputers) *then* so will this implementation of the Cholesky factorization, since it casts most computation in terms of that operation. Unfortunately, matrix-vector operations perform $O(n^2)$ computation on $O(n^2)$ data, meaning that these days the bandwidth to memory typically limits performance.

We summarize information about level-2 BLAS in Figure 5.

A proto-typical calling sequence for a level-1 BLAS routine is

`□axpy( n, alpha, x, incx, y, incy )`,

which implements the scaled vector addition operation $y = \alpha x + y$. Here

- The "□" indicates the data type. The choices for this first letter are

`s`	single precision
`d`	double precision
`c`	single precision complex
`z`	double precision complex

- The operation is identified as `axpy`: alpha times x plus y.
- `n` indicates the number of elements in the vectors x and y.
- `alpha` is the scalar α.
- `x` and `y` indicate the memory locations where the first elements of x and y are stored, respectively.
- `incx` and `incy` equal the increment by which one has to stride through memory for the elements of vectors x and y, respectively

The following are the most frequently used level-1 BLAS:

routine/ function	operation
`□swap`	$x \leftrightarrow y$
`□scal`	$x \leftarrow \alpha x$
`□copy`	$y \leftarrow x$
`□axpy`	$y \leftarrow \alpha x + y$
`□dot`	$x^T y$
`□nrm2`	$\|x\|_2$
`□asum`	$\|\text{re}(x)\|_1 + \|\text{im}(x)\|_1$
`i□max`	$\min(k) : \|\text{re}(x_k)\| + \|\text{im}(x_k)\| = \max(\|\text{re}(x_i)\| + \|\text{im}(x_i)\|)$

Figure 4: Summary of the most commonly used level-1 BLAS.

10.5 Matrix-matrix operations (level-3 BLAS)

Finally, we turn to the implementation of the blocked Cholesky factorization algorithm from Section 5. The algorithm is expressed with FLAME notation and Matlab-like notation in Figure 6.

The routines `dtrsm` and `dsyrk` are level-3 BLAS routines:

- The call to `dtrsm` implements $A_{21} := L_{21}$ where $L_{21} L_{11}^T = A_{21}$.
- The call to `dsyrk` implements $A_{22} := -L_{21} L_{21}^T + A_{22}$.

The bulk of the computation is now cast in terms of matrix-matrix operations which can achieve high performance.

We summarize information about level-3 BLAS in Figure 7.

The naming convention for level-2 BLAS routines is given by

`□XXYY,`

where

- "□" can take on the values `s`, `d`, `c`, `z`.
- `XX` indicates the shape of the matrix:

`XX`	matrix shape
`ge`	general (rectangular)
`sy`	symmetric
`he`	Hermitian
`tr`	triangular

  In addition, operations with banded matrices are supported, which we do not discuss here.

- `YY` indicates the operation to be performed:

`YY`	matrix shape
`mv`	matrix vector multiplication
`sv`	solve vector
`r`	rank-1 update
`r2`	rank-2 update

A representative call to a level-2 BLAS operation is given by

```
dsyr( uplo, n, alpha, x, incx, A, lda )
```

which implements the operation $A = \alpha x x^T + A$, updating the lower or upper triangular part of A by choosing `uplo` as `'Lower triangular'` or `'Upper triangular'`, respectively. The parameter `lda` (the leading dimension of matrix A) indicates the increment by which memory has to be traversed in order to address successive elements in a row of matrix `A`.
The following table gives the most commonly used level-2 BLAS operations:

routine/ function	operation
`□gemv`	general matrix-vector multiplication
`□symv`	symmetric matrix-vector multiplication
`□trmv`	triangular matrix-vector multiplication
`□trsv`	triangular solve vector
`□ger`	general rank-1 update
`□syr`	symmetric rank-1 update
`□syr2`	symmetric rank-2 update

Figure 5: Summary of the most commonly used level-2 BLAS.

	Algorithm	Code

Matrix-matrix	**for** $j = 1 : n$ **in steps of** n_b
	$b := \min(n - j + 1, n_b)$
	$A_{j:j+b-1,j:j+b-1} := \text{CHOL}(A_{j:j+b-1,j:j+b-1})$
	$A_{j+b:n,j:j+b-1} :=$
	$A_{j+b:n,j:j+b-1} A^{-H}_{j:j+b-1,j:j+b-1}$
	$A_{j+b:n,j+b:n} := A_{j+b:n,j+b:n}$
	$- \text{TRIL}(A_{j+b:n,j:j+b-1} A^{H}_{j+b:n,j:j+b-1})$
	endfor
		`do j=1, n, nb`
		`  jb = min( nb, n-j+1 )`
		
		`  call chol( jb, A( j, j ), lda )`
		
		`  call dtrsm( 'Right', 'Lower triangular',`
		`  'Transpose', 'Nonunit diag',`
		`  J-JB+1, JB,  1.0d00, A( j, j ),  lda,`
		`  A( j+jb, j ), lda )`
		
		`  call dsyrk( 'Lower triangular', 'No transpose',`
		`  J-JB+1, JB, -1.0d00, A( j+jb, j ),  lda,`
		`  1.0d00, A( j+jb, j+jb ), lda )`
		`enddo`

FLAME Notation	**Partition** $A \rightarrow \left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right)$
	where A_{TL} is 0×0
	while $m(A_{TL}) < m(A)$ **do**
	Determine block size b
	Repartition
	$\left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
	where A_{11} **is** $b \times b$
	$A_{11} := \text{CHOL}(A_{11})$
	$A_{21} := A_{21}\, \text{TRIL}(A_{11})^{-H}$
	$A_{22} := A_{22} - \text{TRIL}(A_{21} A_{21}^H)$
	Continue with
	$\left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
	endwhile
		`int Chol_unb_var3( FLA_Obj A )`
		`{`
		`  FLA_Obj ATL,  ATR,  A00,  a01,  A02,`
		`  ABL,  ABR,  a10t, alpha11, a12t,`
		`  A20,  a21,  A22;`
		
		`  FLA_Part_2x2( A,  &ATL, &ATR,`
		`  &ABL, &ABR,  0, 0, FLA_TL );`
		
		`  while ( FLA_Obj_length( ATL ) < FLA_Obj_length( A ) ){`
		`  FLA_Repart_2x2_to_3x3(`
		`  ATL, /**/ ATR,  &A00,  /**/ &a01,  &A02,`
		`  /* ************* */  /* ************************* */`
		`  &a10t, /**/ &alpha11, &a12t,`
		`  ABL, /**/ ABR,  &A20,  /**/ &a21,  &A22,`
		`  1, 1, FLA_BR );`
		`  /*-----------------------------------------------*/`
		`  FLA_Sqrt( alpha11 );`
		`  FLA_Invscal( alpha11, a21 );`
		`  FLA_Syr( FLA_LOWER_TRIANGULAR, FLA_MINUS_ONE, A21, A22 );`
		`  /*-----------------------------------------------*/`
		`  FLA_Cont_with_3x3_to_2x2(`
		`  &ATL, /**/ &ATR,  A00,  a01,  /**/ A02,`
		`  a10t, alpha11, /**/ a12t,`
		`  /* ************** */  /* ********************** */`
		`  &ABL, /**/ &ABR,  A20,  a21,  /**/ A22,`
		`  FLA_TL );`
		`  }`
		`  return FLA_SUCCESS;`
		`}`

Figure 6: Blocked algorithm and implementation with level-3 BLAS.

10.6 Impact on performance

Figure 8 illustrates the performance benefits that come from using the different levels of BLAS on a typical architecture.

11 Alternatives to the BLAS

11.1 The FLAME/C API

In a number of places in these notes we presented algorithms in FLAME notation. Clearly, there is a disconnect between this notation and how the algorithms are then encoded with the BLAS interface. In Figures 3 and 6 we also show how the FLAME API for the C programming language [2] allows the algorithms to be more naturally translated into code. While the traditional BLAS interface underlies the implementation of Cholesky factorization and other algorithms in the widely used LAPACK library [1], the FLAME/C API is used in our `libflame` library [9, 13, 14].

11.2 BLIS

The implementations that call BLAS in this paper are coded in Fortran. More recently, the languages of choice for scientific computing have become C and C++. While there is a C interface to the traditional BLAS called the CBLAS [3], we believe a more elegant such interface is the BLAS-like Library Instantiation Software (BLIS) interface [7]. BLIS is not only a framework for rapid implementation of the traditional BLAS, but also presents an alternative interface for C and C++ users.

References

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide (third ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[2] Paolo Bientinesi, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. Representing linear algebra algorithms in code: The FLAME application programming interfaces. *ACM Trans. Math. Soft.*, 31(1):27–59, March 2005.

[3] Basic linear algebra subprograms technical forum standard. *International Journal of High Performance Applications and Supercomputing*, 16(1), Spring 2002.

[4] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.

[5] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[6] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[7] Robert A. van de Geijn Field G. Van Zee. Blis: A framework for rapid instantiation of blas functionality. *ACM Transactions on Mathematical Software*.

[8] John A. Gunnels. *A Systematic Approach to the Design and Analysis of Parallel Dense Linear Algebra Algorithms*. PhD thesis, Department of Computer Sciences, The University of Texas, December 2001.

[9] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. Flame: Formal linear algebra methods environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.

The naming convention for level-3 BLAS routines are similar to those for the level-2 BLAS. A representative call to a level-3 BLAS operation is given by

```
dsyrk( uplo, trans, n, k, alpha, A, lda, beta, C, ldc )
```

which implements the operation $C := \alpha AA^T + \beta C$ or $C := \alpha A^T A + \beta C$ depending on whether `trans` is chosen as `'No transpose'` or `'Transpose'`, respectively. It updates the lower or upper triangular part of C depending on whether `uplo` equal `'Lower triangular'` or `'Upper triangular'`, respectively. The parameters `lda` and `ldc` are the leading dimensions of arrays `A` and `C`, respectively.
The following table gives the most commonly used Level-3 BLAS operationsx

routine/ function	operation
☐`gemm`	general matrix-matrix multiplication
☐`symm`	symmetric matrix-matrix multiplication
☐`trmm`	triangular matrix-matrix multiplication
☐`trsm`	triangular solve with multiple right-hand sides
☐`syrk`	symmetric rank-k update
☐`syr2k`	symmetric rank-2k update

Figure 7: Summary of the most commonly used level-3 BLAS.

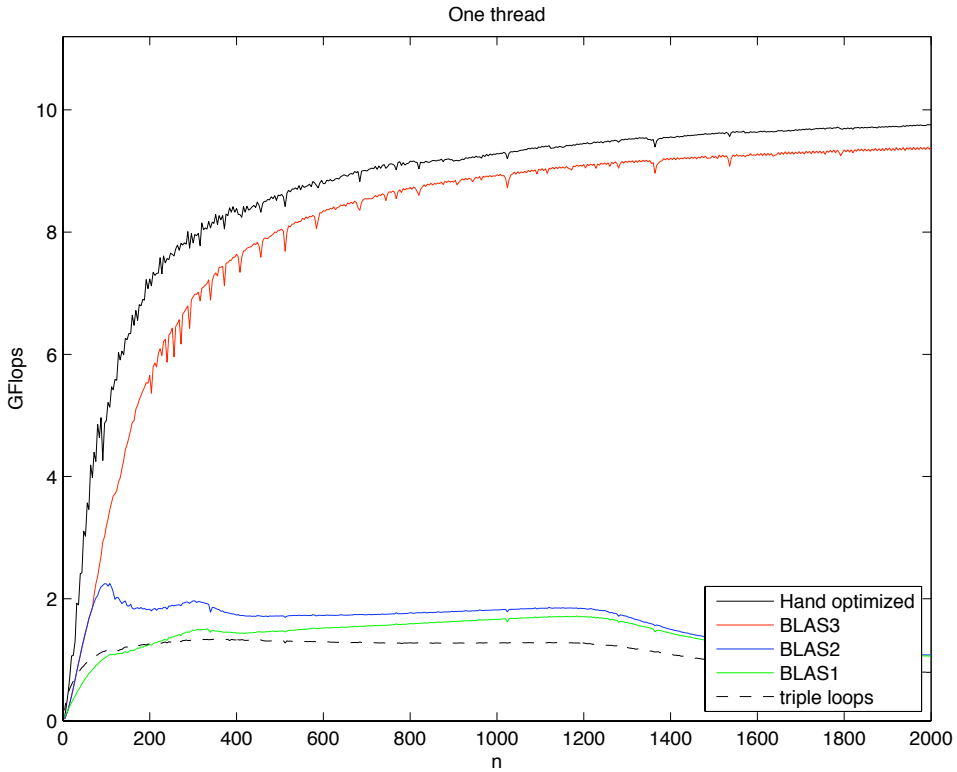

Figure 8: Performance of the different implementations of Cholesky factorization that use different levels of BLAS. The target processor has a peak of 11.2 Gflops (billions of floating point operations per second). BLAS1, BLAS2, and BLAS3 indicate that the bulk of computation was cast in terms of level-1, -2, or -3 BLAS, respectively.

[10] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, Sept. 1979.

[11] Robert van de Geijn and Kazushige Goto. *Encyclopedia of Parallel Computing*, chapter BLAS (Basic Linear Algebra Subprograms), pages Part 2, 157–164. Springer, 2011.

[12] Robert A. van de Geijn and Enrique S. Quintana-Ortí. *The Science of Programming Matrix Computations*. `www.lulu.com/contents/contents/1911788/`, 2008.

[13] Field G. Van Zee. `libflame`*: The Complete Reference*. `www.lulu.com`, 2012.

[14] Field G. Van Zee, Ernie Chan, Robert van de Geijn, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. The libflame library for dense matrix computations. *IEEE Computation in Science & Engineering*, 11(6):56–62, 2009.