Notes on Cholesky Factorization

Robert A. van de Geijn
Department of Computer Science
Institute for Computational Engineering and Sciences
The University of Texas at Austin
Austin, TX 78712
rvdg@cs.utexas.edu

March 11, 2011

1 Definition and Existence

The Cholesky factorization is only defined for symmetric or Hermitian positive definite matrices. In this note, we will restrict ourselves to the case where A is real and symmetric positive definite (SPD).

Definition 1. *A matrix $A \in \mathbb{R}^{m \times m}$ is symmetric positive definite (SPD) if and only if it is symmetric ($A^T = A$) and for all nonzero vectors $x \in \mathbb{R}^m$ it is the case that $x^T A x > 0$.*

First some exercises:

Exercise 2. *Let $B \in \mathbb{R}^{m \times n}$ have linearly independent columns. Prove that $A = B^T B$ is SPD.*

Exercise 3. *Let $A \in \mathbb{R}^{m \times m}$ be SPD. Show that its diagonal elements are positive.*

We will prove the following theorem in Section 4:

Theorem 4. Cholesky Factorization Theorem *Given a SPD matrix A there exists a lower triangular matrix L such that $A = LL^T$.*

The lower triangular matrix L is known as the Cholesky factor and LL^T 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

```
for j = 1 : n
    α_{j,j} := √α_{j,j}

    for i = j + 1 : n
        α_{i,j} := α_{i,j} / α_{j,j}
    endfor

    for k = j + 1 : n
        for i = k : n
            α_{i,k} := α_{i,k} − α_{i,j} α_{k,j}
        endfor
    endfor
endfor
```

```
for j = 1 : n
    α_{j,j} := √α_{j,j}

    α_{j+1:n,j} := α_{j+1:n,j} / α_{j,j}

    α_{j+1:n,j+1:n} :=
        α_{j+1:n,j+1:n} − TRIL(α_{j+1:n,j} α_{j+1:n,j}^T)
endfor
```

Figure 1: Formulations of the Cholesky factorization that expose indices using Matlab-like notation.

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 SPD: Substituting the factors into the equation yields $LL^Tx = y$. Letting $z = L^Tx$,

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

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 $L^Tx = z$.

3 An Algorithm (Variant 3)

The most common algorithm for computing $A := \text{CHOL}(A)$ can be derived as follows: Consider $A = LL^T$. 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 5. *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 $A = LL^T$ we find that

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

so that

$$\left(\begin{array}{c|c} \alpha_{11} = \lambda_{11}^2 & \star \\ \hline a_{21} = \lambda_{11} l_{21} & A_{22} = l_{21} l_{21} + L_{22} L_{22}^T \end{array}\right)$$

and hence

$$\left(\begin{array}{c|c} \lambda_{11} = \sqrt{\alpha_{11}} & \star \\ \hline l_{21} = a_{21}/\lambda_{11} & L_{22} = \textsc{Chol}(A_{22} - l_{21} l_{21}^T) \end{array}\right).$$

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 ensures uniqueness.)
3. Overwrite $a_{21} := l_{21} = a_{21}/\lambda_{11}$.
4. Overwrite $A_{22} := A_{22} - l_{21} l_{21}^T$ (updating only the lower triangular part of A_{22}).
5. Continue with $A = A_{22}$. (Back to Step 1.)

The algorithm is typically presented in a text using Matlab-like notation as illustrated in Fig. 1.

Remark 6. *Similar to the* `tril` *function in Matlab, we use* $\textsc{tril}(B)$ *to denote the lower triangular part of matrix B.*

4 Proof of the Cholesky Factorization Theorem

In this section, we partition A as in (4). The following lemmas, which can be found in any standard text, are key to the proof:

Lemma 7. *Let $A \in \mathbb{R}^{n \times n}$ be SPD. Then α_{11} is real and positive.*

Proof: This is special case of Exercise 1.

Lemma 8. *Let $A \in \mathbb{R}^{m\times m}$ be SPD and $l_{21} = a_{21}/\sqrt{\alpha_{11}}$. Then $A_{22} - l_{21}l_{21}^T$ is SPD.*

Proof: Since A is symmetric so are A_{22} and $A_{22} - l_{21}l_{21}^T$. Let $x_1 \neq 0$ be an arbitrary vector of length $n-1$. Define $x = \left(\begin{array}{c} \chi_0 \\ \hline x_1 \end{array}\right)$ where $\chi_0 = -a_{21}^T x_1/\alpha_{11}$. Then, since $x \neq 0$,

$$
\begin{array}{rcl}
0 < x^T A x & = & \left(\begin{array}{c} \chi_0 \\ \hline x_1 \end{array}\right)^T \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array}\right) \left(\begin{array}{c} \chi_0 \\ \hline x_1 \end{array}\right) \\
& = & \left(\begin{array}{c} \chi_0 \\ \hline x_1 \end{array}\right)^T \left(\begin{array}{c} \alpha_{11}\chi_0 + a_{21}^T x_1 \\ \hline a_{21}\chi_0 + A_{22}x_1 \end{array}\right) \\
& = & \alpha_{11}\chi_0^2 + \chi_0 a_{21}^T x_1 + x_1^T a_{21}\chi_0 + x_1^T A_{22} x_1 \\
& = & \alpha_{11}\frac{a_{21}^T x_1}{\alpha_{11}}\frac{x_1^T a_{21}}{\alpha_{11}} - \frac{x_1^T a_{21}}{\alpha_{11}} a_{21}^T x_1 - x_1^T a_{21}\frac{a_{21}^T x_1}{\alpha_{11}} + x_1^T A_{22} x_1 \\
& = & x_1^T (A_{22} - \frac{a_{21}a_{21}^T}{\alpha_{11}}) x_1 \\
& = & x_1^T (A_{22} - l_{21}l_{21}^T) x_1.
\end{array}
$$

We conclude that $A_{22} - l_{21}l_{21}^T$ is SPD.

Proof: of **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 SPD 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 SPD matrix $A \in \mathbb{R}^{(n-1)\times(n-1)}$. We will show that it holds for $A \in \mathbb{R}^{n\times n}$. Let $A \in \mathbb{R}^{n\times n}$ be SPD. Partition A and L as in (4) and let $\lambda_{11} = \sqrt{\alpha_{11}}$ (which is well-defined by Lemma 4), $l_{21} = a_{21}/\lambda_{11}$, and $L_{22} = \textsc{Chol}(A_{22} - l_{21}l_{21}^T)$ (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 (Variant 3)

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

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

for $j = 1 : n$ **in steps of** n_b
  $b := \min(n - j + 1, n_b)$
  $A_{j:j+b-1,j:j+b-1} := \text{CHOL}(A_{j:j+b-1,j:j+b-1})$
  $A_{j+b:n,j:j+b-1} := A_{j+b:n,j:j+b-1} A^{-T}_{j:j+b-1,j:j+b-1}$
  $A_{j+b:n,j+b:n} := A_{j+b:n,j+b:n} - \text{TRIL}(A_{j+b:n,j:j+b-1} A^{T}_{j+b:n,j:j+b-1})$
endfor

Figure 2: Blocked algorithm for computing the Cholesky factorization. Here n_b is the block size used by the algorithm.

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

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

From this we conclude that

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

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 $A_{11} := L_{11} = \text{CHOL}(A_{11})$.
3. Overwrite $A_{21} := L_{21} = A_{21}L_{11}^{-T}$.
4. Overwrite $A_{22} := A_{22} - L_{21}L_{21}^T$ (updating only the lower triangular part).
5. Continue with $A = A_{22}$. (Back to Step 1.)

An algorithm that explicitly indexes into the array that stores A is given in Fig. 2.

Remark 9. *The Cholesky factorization $A_{11} := L_{11} = \text{CHOL}(A_{11})$ can be computed with the unblocked algorithm or by calling the blocked Cholesky factorization algorithm recursively. Operations like $L_{21} = A_{21}L_{11}^{-T}$ are computed by solving the equivalent linear system with multiple right-hand sides $L_{11}L_{21}^T = A_{21}^T$.*

6 Alternative Representation

When explaining the above algorithm in a classroom setting, invariably it is accompanied by a picture sequence like the one in Fig. 3(left) and the (verbal) explanation:

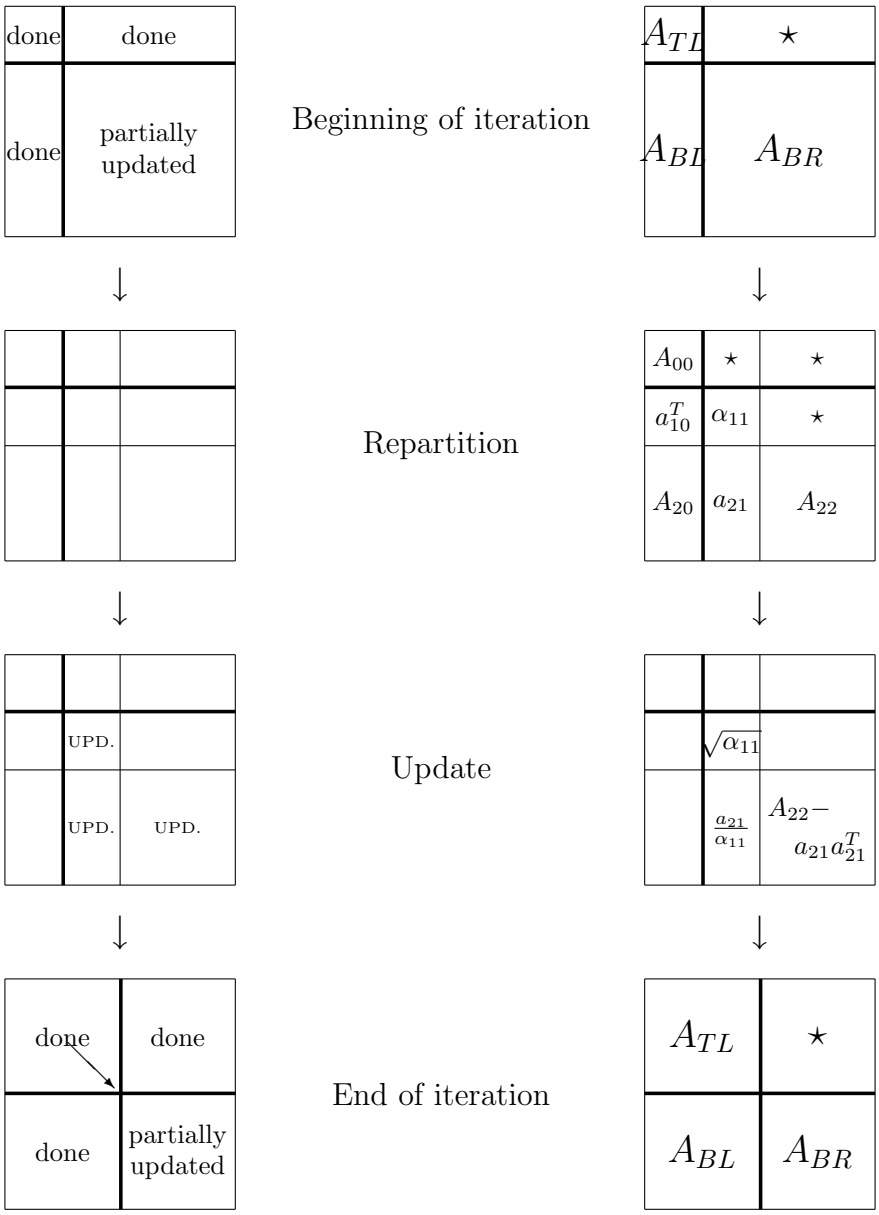

Figure 3: Left: Progression of pictures that explain Cholesky factorization algorithm. Right: Same pictures, annotated with labels and updates.

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}.

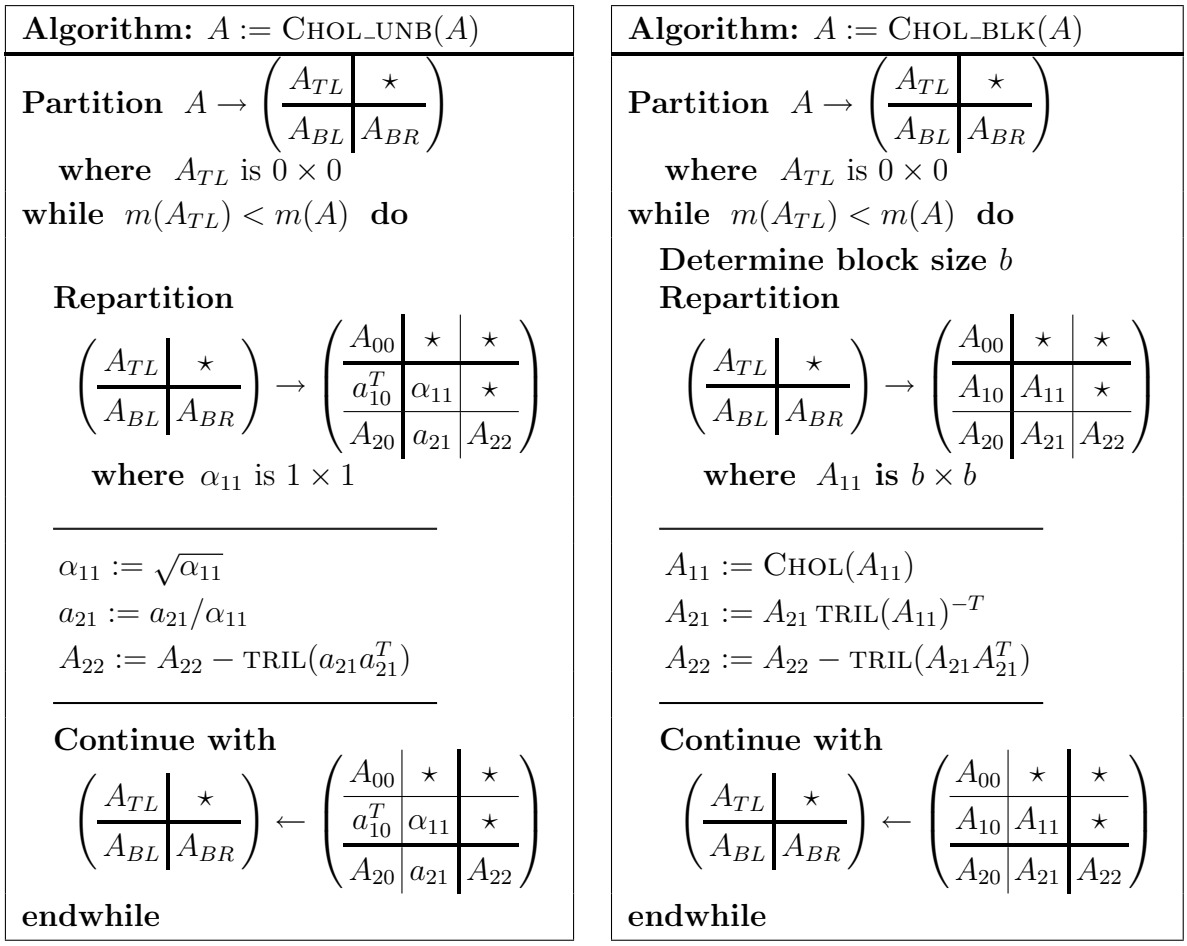

Figure 4: Unblocked and blocked algorithms for computing the Cholesky factorization.

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.

To motivate our notation, we annotate this progression of pictures as in Fig. 3 (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 Fig. 4 (left). A similar explanation can be given for the blocked algorithm, which is given in Fig. 4 (right). In the algorithms, $m(A)$ indicates the number of rows of matrix A.

Remark 10. *The indices in our more stylized presentation of the algorithms are subscripts rather than indices in the conventional sense.*

Remark 11. *Clearly Fig. 4 does not present the algorithm as concisely as the algorithms given in Figs. 1 and 2. However, it does capture to a large degree the verbal description of the algorithm mentioned above and therefore, in our opinion, reduces both the effort required to interpret the algorithm and the need for additional explanations. The notation also mirrors that used for the proof in Section 4.*

Remark 12. *The notation in Figs. 3 and 4 allows the contents of matrix A at the beginning of the iteration to be formally stated:*

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

where $L_{TL} = \text{CHOL}(\hat{A}_{TL})$, $L_{BL} = \hat{A}_{BL}L_{TL}^{-T}$, 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 13. *Implement the Cholesky factorization with M-script.*

7 Cost

The cost of the Cholesky factorization of $A \in \mathbb{R}^{m\times m}$ can be analyzed as follows: In Fig. 4 (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.
- $A_{22} := A_{22} - \text{TRIL}(a_{21}a_{21}^T)$: 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

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

which allows us to state that (obvious) most computation is in the update of A_{22}.

8 Other Algorithms

The algorithms that were described in Sections 3 and 5 are sometimes called the unblocked and blocked *right-looking* algorithms. The idea is that relative to the current column or block, the bulk of data with which one computes is to the right. In some of our papers we also call these Variant 3.

There are other algorithms for this operation, which we describe next.

8.1 Bordered algorithm (Variant 1)

Another algorithm for computing $A := \text{CHOL}(A)$ can be derived as follows: Consider again $A = LL^T$. Partition

$$A = \left(\begin{array}{c|c} A_{00} & \star \\ \hline a_{10}^T & \alpha_{11} \end{array}\right) \quad \text{and} \quad L = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array}\right).$$

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

$$\left(\begin{array}{c|c} A_{00} & \star \\ \hline a_{10}^T & \alpha_{11} \end{array}\right) = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array}\right)\left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array}\right)^T = \left(\begin{array}{c|c} L_{00}L_{00}^T & \star \\ \hline l_{10}^T L_{00}^T & l_{10}^T l_{10} + \lambda_{11}^2 \end{array}\right)$$

from which we conclude that

$$\left(\begin{array}{c|c} L_{00} = \text{CHOL}(A)_{00} & \star \\ \hline l_{10}^T = a_{10}^T L_{00}^{-T} & \lambda_{11} = \sqrt{\alpha_{11} - l_{10}^T l_{10}} \end{array}\right).$$

These equalities motivate the algorithm

1. Partition $A \rightarrow \left(\begin{array}{c|c} A_{00} & \star \\ \hline a_{10}^T & \alpha_{11} \end{array}\right)$.
2. Assume that $A_{00} := L_{00} = \text{CHOL}(A_{00})$ has been computed by previous iterations of the loop-based algorithm.
3. Overwrite $a_{10}^T := l_{10}^T = a_{10}^T L_{00}^{-T}$.
4. Overwrite $\alpha_{11} := \sqrt{\alpha_{11} - l_{10}^T l_{10}}$.

This justifies the unblocked Variant 1 in Figure 5.

A blocked algorithm can be similarly derived: Partition

$$A = \left(\begin{array}{c|c} A_{00} & \star \\ \hline A_{10} & A_{11} \end{array}\right) \quad \text{and} \quad L = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline L_{10} & L_{11} \end{array}\right) \tag{2}$$

Algorithm: $A := \text{CHOL\_UNB}(A)$

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**

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

Variant 1 (Bordered Algorithm)

$a_{10}^T := a_{10}^T \text{TRIL}(A_{00})^{-T}$

$\alpha_{11} := \alpha_{11} - a_{10}^T a_{10}$

$\alpha_{11} := \sqrt{\alpha_{11}}$

Variant 2 (Left-looking Algorithm)

$\alpha_{11} := \alpha_{11} - a_{10}^T a_{10}$

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21} - A_{20} a_{10}$

$a_{21} := a_{21}/\alpha_{11}$

Variant 3 (Right-looking Algorithm)

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - \text{TRIL}(a_{21} a_{21}^T)$

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

Algorithm: $A := \text{CHOL\_BLK}(A)$

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** $b \times b$

Variant 1 (Bordered Algorithm)

$A_{10} := A_{10} \text{TRIL}(A_{00})^{-T}$

$A_{11} := A_{11} - A_{10} A_{10}^T$

$A_{11} := \text{CHOL}(A_{11})$

Variant 2 (Left-looking Algorithm)

$A_{11} := A_{11} - A_{10} A_{10}^T$

$A_{11} := \text{CHOL}(A_{11})$

$A_{21} := A_{21} - A_{20} A_{10}^T$

$A_{21} := A_{21} \text{TRIL}(A_{11})^{-T}$

Variant 3 (Right-looking Algorithm)

$A_{11} := \text{CHOL}(A_{11})$

$A_{21} := A_{21}\, \text{TRIL}(A_{11})^{-T}$

$A_{22} := A_{22} - \text{TRIL}(A_{21} A_{21}^T)$

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

Figure 5: Multiple variants for computing the Cholesky factorization.

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

$$\left(\begin{array}{c|c} A_{00} & \star \\ \hline A_{10} & A_{11} \end{array}\right) = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline L_{10} & L_{11} \end{array}\right)\left(\begin{array}{c|c} L_{00} & 0 \\ \hline L_{10} & L_{11} \end{array}\right)^T = \left(\begin{array}{c|c} L_{00}L_{00}^T & \star \\ \hline L_{10}L_{00}^T & L_{10}L_{10}^T + L_{11}L_{11}^T \end{array}\right)$$

from which we conclude that

$$\left(\begin{array}{c|c} L_{00} = \textsc{Chol}(A)_{00} & \star \\ \hline L_{10} = A_{10}L_{00}^{-T} & L_{11} = \textsc{Chol}(A_{11} - L_{10}L_{10})^T \end{array}\right).$$

These equalities motivate the algorithm

1. Partition $A \rightarrow \left(\begin{array}{c|c} A_{00} & \star \\ \hline A_{10} & A_{11} \end{array}\right)$.
2. Assume that $A_{00} := L_{00} = \textsc{Chol}(A_{00})$ has been computed by previous iterations of the loop-based algorithm.
3. Overwrite $A_{10} := L_{10} = A_{10}L_{00}^{-T}$.
4. Overwrite $A_{11} := A_{11} - L_{10}L_{10}^T$.
5. Overwrite $A_{11} := L_{11} = \textsc{Chol}(A_{11})$ (by calling, for example, the unblocked algorithm).

This justifies the blocked Variant 1 in Figure 5.

8.2 Left-looking algorithm (Variant 2)

Yet another algorithm for computing $A := \textsc{Chol}(A)$ can be derived as follows: Consider again $A = LL^T$. This time partition

$$A = \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) \quad \text{and} \quad L = \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right) \tag{3}$$

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

$$\begin{array}{rcl} \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) & = & \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right)^T \\ & = & \left(\begin{array}{c|c|c} L_{00}L_{00}^T & \star & \star \\ \hline l_{10}^T L_{00}^T & l_{10}^T l_{10} + \lambda_{11}^2 & \star \\ \hline L_{20}L_{00}^T & L_{20}l_{10} + \lambda_{11}l_{21} & L_{20}L_{20}^T + l_{21}l_{21}^T + L_{22}L_{22}^T \end{array}\right) \end{array}$$

from which we conclude that

$$\left(\begin{array}{c|c|c} A_{00} = L_{00}L_{00}^T & \star & \star \\ \hline a_{10}^T = l_{10}^T L_{00}^T & \alpha_{11} = l_{10}^T l_{10} + \lambda_{11}^2 & \star \\ \hline A_{20} = L_{20}L_{00}^T & a_{21} = L_{20}l_{10} + \lambda_{11} l_{21} & A_{22} = L_{20}L_{20}^T + l_{21}l_{21}^T + L_{22}L_{22}^T \end{array}\right).$$

Now, let us assume that in previous iterations of the loop the computation has proceeded so that A contains

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \alpha_{11} & 0 \\ \hline L_{20} & a_{21} & A_{22} \end{array}\right).$$

The purpose of the current iteration is to make it so that A contains

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & A_{22} \end{array}\right).$$

The following algorithm accomplishes this:

1. Partition

$$A \rightarrow \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

   and assume that due to computation from previous iterations it contains

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \alpha_{11} & 0 \\ \hline L_{20} & a_{21} & A_{22} \end{array}\right).$$

2. Overwrite $\alpha_{11} := \lambda_{11} = \sqrt{\alpha_{11} - l_{10}^T l_{10}}$.

3. Overwrite $a_{21} := a_{21} - L_{20}l_{10}$.

4. Overwrite $a_{21} := l_{21} = a_{21}/\lambda_{11}$.

This justifies the unblocked Variant 2 in Figure 5.

A blocked algorithm can be similarly derived. Partition

$$A = \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right) \quad \text{and} \quad L = \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right) \tag{4}$$

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

$$\left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right) \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right)^T$$

$$= \left(\begin{array}{c|c|c} L_{00}L_{00}^T & \star & \star \\ \hline L_{10}L_{00}^T & L_{10}L_{10}^T + L_{11}L_{11}^T & \star \\ \hline L_{20}L_{00}^T & L_{20}L_{10}^T + L_{11}L_{21} & L_{20}L_{20}^T + L_{21}L_{21}^T + L_{22}L_{22}^T \end{array}\right)$$

from which we conclude that

$$\left(\begin{array}{c|c|c} A_{00} = L_{00}L_{00}^T & \star & \star \\ \hline A_{10} = L_{10}L_{00}^T & A_{11} = L_{10}L_{10}^T + L_{11}^2 & \star \\ \hline A_{20} = L_{20}L_{00}^T & A_{21} = L_{20}L_{10}^T + L_{11}L_{21} & A_{22} = L_{20}L_{20}^T + L_{21}L_{21}^T + L_{22}L_{22}^T \end{array}\right).$$

Now, let us assume that in previous iterations of the loop the computation has proceeded so that A contains

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & A_{11} & 0 \\ \hline L_{20} & A_{21} & A_{22} \end{array}\right).$$

The purpose of the current iteration is to make it so that A contains

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & A_{22} \end{array}\right).$$

The following algorithm accomplishes this:

1. Partition

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

   and assume that due to computation from previous iterations it contains

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & A_{11} & 0 \\ \hline L_{20} & A_{21} & A_{22} \end{array}\right).$$

2. Overwrite $A_{11} := L_{11} = \text{CHOL}(A_{11} - L_{10}L_{10}^T)$.

3. Overwrite $A_{21} := A_{21} = A_{21} - L_{20}L_{10}^T$.

4. Overwrite $A_{21} := L_{21} = A_{21}L_{11}^{-T}$.

This justifies the blocked Variant 2 in Figure 5.

9 Coding the algorithms

As part of the FLAME project at The University of Texas at Austin, Universidad Jaume I, Spain, and RWTH Aachen University, Germany, we have developed APIs for coding these algorithms that make it so that the code closely resembles the algorithms.

The demonstrate how these codes are written, we created a video titled *High-Performance Implementation of Cholesky Factorization* that can be found at

`http://www.cs.utexas.edu/users/flame/Movies.html#Chol`.

Exercise 14. *Implement all the algorithms in Figure 5 in M-script (the scripting language of Matlab and Octave).*

Exercise 15. *Implement all the algorithms in Figure 5 using the FLAME/C API mentioned in the video.*

10 Performance

In Figure 6 we show the performance attained on a multicore architecture.

All experiments were performed using double-precision floating-point arithmetic on a Dell/PowerEdge-1435 powered by a Intel Xeon X5355 (2.666 GHz) processor. This particular architecture has four cores, each of which has a clock rate of about 2.7GHz and each core can perform four floating point operations per clock cycle. Thus, the peak of this machine is

$$4 \text{ cores } \times \frac{4 \text{ FLOPS}}{\text{cycle}} \times \frac{2.7 \times 10^9 \text{ cycles}}{\text{core}} \approx 43 \times 10^9 FLOPS = 43 GFLOPS.$$

What the graph shows is the problem size for which performance was measured along the x-axis and the rate at which the algorithm computed along the y-axis. The rate was computed by, for a given time, measuring the time required to complete the computation, $t(n)$ (in seconds), and then dividing the number of floating point operations by this time. This gives the rate, in floating point operations per second (FLOPS), at which the computation executed. Since this is a number measured in billions, we then divide this number by a billion to get the rate in GFLOPS (GigaFLOPS, or billions of floating point operations per second):

$$\text{rate in GFLOPS} = \frac{n^3}{t(n) \times 10^9}.$$

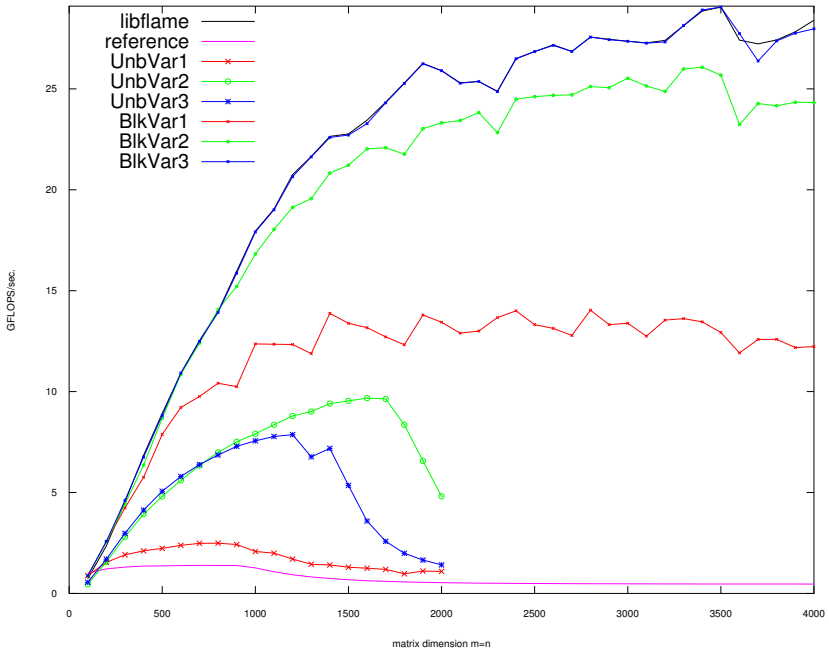

Figure 6: Performance of the various algorithms on a 16 core architecture. The peak of this architecture is around 40 GFLOPS.

Thus, our implementations reach about 2/3 of the peak of the processor, which is very good for this operation. It would be worthwhile to play with the block size (which we chose to be 128) to see if changing it improves performance.

In the graph we do not show the performance of the LAPACK routine for Cholesky factorization, `DPOTRF`. In other experiments we did compare against a version of that routine that we changed so that we could control the block size that it uses. It then attains exactly the same performance as our blocked Variant 2 (and hence less than our blocked Variant 3 and the `libflame` routine `FLA_Chol`).

11 Additional Reading

The interested reader may be interested in the following documents:

1. A paper that illustrates how different algorithmic variants for Cholesky factorization and related operations should be chosen for different kinds of architectures, ranging from sequential processors to multithreaded (multicore) architectures to distributed memory parallel computers.

   Paolo Bientinesi, Brian Gunter, and Robert A. van de Geijn. Families of

algorithms related to the inversion of a symmetric positive definite matrix. *ACM Transactions on Mathematical Software*, 35(1).

2. Two papers that show how matrix-matrix operations can achieve near-peak performance:

   Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Soft.*, 34(3: Article 12, 25 pages), May 2008.

   and

   Kazushige Goto and Robert van de Geijn. High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Softw.*, 35(1):1–14, 2008.

3. A book that shows how to systematically derive all loop-based algorithms for a broad range of linear algebra operations, including Cholesky factorization:

   Robert A. van de Geijn and Enrique S. Quintana-Ortí. *The Science of Programming Matrix Computations*. `http://www.lulu.com/content/1911788`, 2008.

4. The User's Guide for the `libflame` library, a modern replacement for the LAPACK library that is coded using the APIs mentioned in the movie:

   Field G. Van Zee. `libflame`*: The Complete Reference*. `www.lulu.com`, 2009.

5. An overview of the FLAME project:

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