A NEW DIVIDE AND CONQUER PARALLEL ALGORITHM
FOR THE CHOLESKY DECOMPOSITION
OF BAND MATRICES.

by
Ilan Bar-On

Technical Report #667
March 1991
A NEW DIVIDE AND CONQUER PARALLEL ALGORITHM FOR THE
CHOLESKY DECOMPOSITION OF BAND MATRICES

ILAN BARON*

Abstract.
We present a new divide and conquer parallel algorithm for finding the Cholesky decomposition of a band symmetric positive definite matrix. This is the first time such an algorithm is presented. All previously known parallel algorithms for this problem are direct implementations of the sequential methods, which as though, offer almost no speedup. Here, for the first time a parallel oriented algorithm is presented, with an approximate speedup of order $p/3$ given $p$ processors. Moreover, the algorithm can be efficiently implemented on many existing parallel computers.

We further discuss the more theoretical aspects of the algorithm, and show that it can be implemented efficiently in $O(\log m \log n)$ time using $p = (n/m)M(m)/(\log m \log n)$ processors. Here, $M(m) = m^\beta, 2 \leq \beta \leq 3$, and $m^\beta/\log m$ denotes the least number of processors required in order to multiply two matrices of order $m$ in $O(\log m)$ time. This improves by a factor of $\log m$ the best previously known result for this problem.

We conclude with an application of the algorithm to the finding of the eigenvalues of a non-singular band symmetric matrix. We show for the first time how to implement each iteration of the QR algorithm in the same complexity as above.

Key Words. Cholesky decomposition, band matrices, parallel algorithm, QR algorithm.

1. Introduction. In this paper we study the problem of finding the Cholesky decomposition of a band symmetric positive definite (s.p.d.) matrix on large parallel MIMD computers. Such a decomposition is important in many numerical methods for solving systems of linear equations and for computing the eigenvalues and eigenvectors of a corresponding matrix.

In this work we present a new divide and conquer parallel algorithm which offers an approximate $p/3$ speedup over known sequential methods, given $p$ processors. This is the first time such a parallel oriented algorithm is presented.

Let $A$ be a band s.p.d. matrix of order $n$ and bandwidth $m$. The problem of finding the Cholesky decomposition of $A$ is reduced to that of finding $p$ independent decompositions of smaller matrices of order $n/p$ and bandwidth $m$. Each such problem is then solved independently and in parallel by the corresponding processor.

The algorithm can be viewed as a bottom up sweep followed by a top down sweep of a complete binary tree of $p = 2^{k+1}$ leaves, see Figure 1.1. In the beginning each processor computes in parallel, then the pairs of processors

$$ (j-1)2^i + 1, j2^i \quad j = 1, 2, \ldots, p/2^i, $$

for $i = 1, 2, \ldots, k, k + 1, k, \ldots, 2, 1$, communicate and compute, and in the end, each processor again computes in parallel. More precisely, in the beginning each processor performs approximately $2mn^2/p$ operations, in the intermediate steps the pairs of processors in (1) exchange $O(m^2)$ information, and perform $O(m^3)$ operations, and in the

---

* Technion - Israel Institute of Technology, Department of Computer Science, Technion City, Haifa 32 000, Israel. baron@TECHSEL.bitnet
end each processor performs approximately $nm^2/p$ operations. We conclude that the algorithm offers an approximate $p/3$ speed-up over sequential algorithms and that it can be efficiently implemented on many parallel computers.

All previously known parallel algorithm for this problem are direct implementations of the sequential methods, see for example [5, 7, 10]. As such, they suffer from the inherent sequential nature of these methods and in most interesting cases, where the bandwidth is small, they offer almost no speed-up.

In Section 2, we present the new divide and conquer parallel algorithm, and the top-down sweep. In Section 3, we describe the bottom-up preprocess sweep. In Section 4, we give a pseudo code for the whole algorithm and in Section 5, we analyze its complexity in the most general case. In Section 6, we give an application of the algorithm to the finding of the eigenvalues of a non-singular symmetric band matrix, and in the conclusion we point to some open related problems.

2. The algorithm. Let $A$ be a band s.p.d. matrix of order $n = qm$ and band $m$.

\[
A = \begin{pmatrix}
A_1 & L_1 \\
U_1 & A_2 & L_2 \\
 & \ddots & \ddots & \ddots \\
 & & U_{q-2} & A_{q-1} & L_{q-1} \\
 & & & U_{q-1} & A_q
\end{pmatrix}, \quad U_i, A_i, L_i \in M(m),
\]

where $q = \lfloor p \rfloor$, where $p = 2^{k+1}$ is the number of processors available.

The decomposition of $A = LL^t$ can be written as

\[
A = \begin{pmatrix}
C & Q^t \\
Q & D
\end{pmatrix} = \begin{pmatrix}
\hat{L}_1 \\
X
\end{pmatrix} \begin{pmatrix}
\hat{L}_1^t & X^t
\end{pmatrix},
\]
where all submatrices are of order \( n/2 \). Moreover, \( \hat{L}_1, \hat{L}_2 \) are band lower triangular matrices of bandwidth \( m \), and \( Q^t, X^t \) are given by

\[
(4) \quad Q^t = \begin{pmatrix} 0 & 0 \\ L_{q/2} & 0 \end{pmatrix}, \quad X^t = \begin{pmatrix} 0 & 0 \\ L_x & 0 \end{pmatrix}
\]

where \( L_{q/2}, L_x \) are lower triangular matrices of order \( m \). Equating the submatrices in formula (3) we see that

\[
(5) \quad C = \hat{L}_1 \hat{L}_1^t, \quad Q = X\hat{L}_1^t, \quad X = Q\hat{L}_1^{-t}, \quad D = XX^t + \hat{L}_2\hat{L}_2^t,
\]

and therefore

\[
(6) \quad \hat{L}_2\hat{L}_2^t = D - XX^t = D - Q\hat{L}_1^{-t}\hat{L}_1^{-1}Q^t = D - QC^{-1}Q^t.
\]

Hence, the Cholesky decomposition of \( A \) can be found as follows:

- Solve in parallel

\[
(7) \quad CY = C \begin{pmatrix} \vdots \\ \mathbf{r} \end{pmatrix} = \begin{pmatrix} 0 \\ L_{q/2} \end{pmatrix}, \quad Y \in M((n/2) \times m), \quad \mathbf{r} \in M(m).
\]

- Compute \( Z = U_{q/2}\mathbf{r} \) and set

\[
(8) \quad D' = D - \begin{pmatrix} Z & 0 \\ 0 & 0 \end{pmatrix}.
\]

- Find in parallel the decompositions:

\[
(9) \quad C = \hat{L}_1 \hat{L}_1^t, \quad D' = \hat{L}_2 \hat{L}_2^t.
\]

- Solve \( \hat{L}_1 X^t = Q^t \), i.e.,

\[
(10) \quad \begin{pmatrix} \hat{L}_{1,1} & \cdots & \hat{L}_{1,2} \\ \vdots & \ddots & \vdots \\ \hat{U}_{1,q/2-2} & \cdots & \hat{U}_{1,q/2-1} \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \hat{L}_{1,x} \\ \hat{L}_{q/2} \end{pmatrix}.
\]

Here, all submatrices are of order \( m \), \( \hat{L}_{1,i}, i = 1, 2, \ldots, q/2 \) is lower triangular, and \( \hat{U}_{1,i}, i = 1, 2, \ldots, q/2 - 1 \) is upper triangular. Hence, the problem is reduced to the solution of the triangular system

\[
(11) \quad \hat{L}_{1,q/2}\hat{L}_{1,x} = L_{q/2}.
\]
We present in this work an efficient parallel implementation of the above algorithm, but we first introduce some new notations.

Let \( A^s, s = 0, 1, \ldots, k \) denote the matrix \( A \) with the substructuring

\[
A^s = \begin{pmatrix}
A_1^s & L_1^s & L_2^s \\
U_1^s & A_2^s & \ldots & \ldots \\
& U_2^s & \ddots & \ddots \\
& & \ddots & U_{v-1}^s & A_v^s
\end{pmatrix}, \quad U_1^s, A_1^s, L_1^s \in M(2^{s+1}/p),
\]

where \( v = p/2^s \) and \( s = 0, \ldots, k \).

Here, \( A_i^s, i = 1, \ldots, v \) denotes the \( i \)th principal submatrix of order \( 2^{s+1}/p \), i.e., rows and columns \((i - 1)2^{s+1}/p + 1, \ldots, i2^{s+1}/p\), of \( A \). The offdiagonal block elements \( L_i^s, U_i^s \) are given by

\[
L_i^s = \begin{pmatrix} 0 & 0 \\ L_{i+2^s}^s & 0 \end{pmatrix}, \quad U_i^s = \begin{pmatrix} 0 \\ U_{i+2^s}^s \end{pmatrix}, \quad L_i^s \in M(2^{s+1}/p \times m).
\]

In addition, let \( E_i^s, F_i^s, G_i^s, H_i^s, s = 0, 1, \ldots, k \) denote the matrices of order \( m \) given by

\[
E_i^s = \begin{pmatrix} 0 \\ F_i^s \end{pmatrix}, \quad F_i^s = \begin{pmatrix} 0 \\ E_i^s \end{pmatrix}, \quad E_i^s \in M(2^{s+1}/p \times m).
\]

The Algorithm: Let \( A \) be the given matrix as in (2) and let \( p = 2^{k+1} \) be the number of processors available. The algorithm reduces the problem of decomposing the matrix \( A \) into \( p \) independent subproblems of the same type but of order \( n/p \). Each such subproblem is then solved independently and in parallel by the corresponding processor. There are \( k + 1 = \log p \) steps in the algorithm, and in step \( s = k, k - 1, \ldots, 0 \), each decomposition problem of order \( 2^{s+1} \) is reduced into two independent decomposition problems of order \( 2^s \). Let \( D^{s+1} \equiv A \) denotes the original matrix and let \( D^s, s = k, k - 1, \ldots, 0 \), denotes the same matrix after step \( s \) above. We use similar notations for \( D^s \) as for \( A^s \) in (12), i.e.,

\[
D_i^s, \quad i = 1, 2, \ldots, p/2^s, \quad s = k, k - 1, \ldots, 0
\]

denote the corresponding principal submatrices of \( D^s \). We describe the main stages of the algorithm in the following:
• For $s = k, k - 1, \ldots, 0$ do in parallel

\begin{equation}
D_i^s = A_i^s - U_{i-1}^s (D_{i-1}^s)^{-1} L_{i-1}^s, \quad i = 2j, j = 1, 2, \ldots, p/2^{s+1}.
\end{equation}

• Find in parallel the Cholesky decomposition of $D_i^0, i = 1, 2, \ldots, p$.

Let $D_i^0 = \mathcal{L}_i \mathcal{C}_i, \ i = 1, \ldots, p$ be given by

\begin{equation}
\mathcal{L}_i = \begin{pmatrix}
L_{i,1} & L_{i,2} & \cdots & L_{i,j-1} & L_{i,j} \\
U_{i,1} & \cdots & \cdots & U_{i,j-2} & U_{i,j-1} & U_{i,j}
\end{pmatrix},
\end{equation}

where $L_{i,j}, j = 1, 2, \ldots, l$, is lower triangular, and $U_{i,j}, j = 1, 2, \ldots, l - 1$, is upper triangular.

• Solve in parallel the triangular systems

\begin{equation}
L_{i,j} L_{i,x} = \mathcal{L}_i, \quad i = 1, 2, \ldots, p - 1.
\end{equation}

The decomposition $A = LL'$ is then given by

\begin{equation}
L = \begin{pmatrix}
\mathcal{L}_1 \\
L_{1x} & \mathcal{L}_2 \\
& \ddots & \ddots & \mathcal{L}_{p-1} \\
& & L_{p-2x} & \mathcal{L}_p
\end{pmatrix}.
\end{equation}

We show in Theorem 2.1 below how to perform efficiently the first stage of the algorithm.

**Theorem 2.1.** We define $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{p-1}\}, \mathcal{R}_i \in M(m)$. Let, $\mathcal{R}_{2^s}, s = k, k - 1, \ldots, 0$ be given by the following linear systems

\begin{equation}
(D_i^{2^s})^{-1} \mathcal{L}_i = \begin{pmatrix}
\mathcal{R}_{2^s}
\end{pmatrix}, \quad i = 2j + 1, j = 0, 1, \ldots, p/2^{s+1} - 1.
\end{equation}

Then for $s = k, k - 1, \ldots, 0, \mathcal{R}_{2^s} = \mathcal{F}_{2^s}$ and

\begin{equation}
\mathcal{R}_{2^s}, \quad i = 2j + 1, j = 1, 2, \ldots, p/2^{s+1} - 1,
\end{equation}

is given recursively by

\begin{equation}
\mathcal{R}_{2^{s+1}} = \mathcal{F}_{2^s} + \mathcal{H}_{2^s} \mathcal{R}_{2^{s+1}} (I - \mathcal{G}_{2^s} \mathcal{R}_{2^{s+1}})^{-1} \mathcal{E}_{2^s}.
\end{equation}

**Proof.** We first observe that the $\mathcal{R}_i, i = 1, \ldots, p - 1$ are well defined since

\begin{equation}
i 2^s = i' 2^{s'} \rightarrow i = i', s = s'.
\end{equation}
Next, it follows from (18) that $D_i = A_i$ and hence, the first assertion follows easily from (15). To derive the second assertion we observe from (18) that

\[(D_i)^{-1}L_i = (A_i - \begin{pmatrix} U_{j_2+i} & R_{j_2+i} \\ 0 & 0 \end{pmatrix}^{-1}L_i, \]

and that

\[D_i^2 = A_i - \begin{pmatrix} U_{j_2+i} & R_{j_2+i} \\ 0 & 0 \end{pmatrix}^{-1}L_i, \]

Therefore, for $i = 2j + 1$

\[ (D_i)^{-1}L_i = (I - (A_i^{-1}U_{i-1} \begin{pmatrix} * \\ R_{j_2+i} \end{pmatrix}^{-1}(A_i^{-1})^{-1}L_i, \]

and substituting (15) and (16) we finally get

\[ (D_i)^{-1}L_i = (I - \begin{pmatrix} G_i & \cdots & 0 \\ E_i & \cdots & 0 \\ I \end{pmatrix}^{-1} \begin{pmatrix} E_i \\ F_i \\ I \end{pmatrix}^{-1}, \]

\[ (I_m - G_i R_{j_2+i})^{-1} \]

\[ (H_i R_{j_2+i} (I_m - G_i R_{j_2+i})^{-1} I_m \]

\[ (F_i + H_i R_{j_2+i} (I - G_i R_{j_2+i})^{-1} E_i \]

The result now follows by comparing (32) with (22). \(\square\)

**COROLLARY 2.2.** Suppose that the submatrices

\[ E_i, F_i, G_i, H_i \quad i = 1, 3, \ldots, p/2^s - 1 \]

\[ s = 0, 1, \ldots, k, \]

are known. Then we can implement each step of the first stage of the algorithm in \(O(m^3)\) operations.

In the next Section, we present an efficient unite and conquer parallel algorithm for computing the submatrices in (33).
3. Computing the E's, F's, G's, H's.. The algorithm we present may be viewed as a parallel version of the Gaussian elimination procedure. In the sequential case, the Gaussian elimination procedure allows us to obtain the final solution by substitution. Similarly, in the parallel elimination procedure the final solution is obtained by parallel substitution. The algorithm was first described in the author paper "A practical parallel algorithm for solving band symmetric positive definite systems of linear equations" [1], section 2.1, and we will review here its main new ideas.

Let us denote the \( i \)th row of \( A^s, s = 0, 1, \ldots, k \), given in (12), by

\[
T_i^s = \begin{pmatrix} U_i^{s-1} & A_i^s & L_i^s & 0 \end{pmatrix} \quad i = 1, 2, \ldots, p/2^s.
\]

Then the algorithm proceeds as follows:

- **Step 0.** We diagonalize the submatrices \( A_i^0, i = 1, 2, \ldots, p \), by Gaussian elimination. We denote by

\[
T_i^0 = \begin{pmatrix} \mathcal{G}_i^0 & \mathcal{V}_i^0 & 0 & 0 & \mathcal{E}_i^0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \mathcal{H}_i^0 & 0 & \mathcal{W}_i^0 & \mathcal{F}_i^0 \end{pmatrix}, \quad i = 1, 2, \ldots, p,
\]

the corresponding rows of \( A^0 \) after step 0.

- **Step \( s = 1, 2, \ldots, k \).** The \( i \)th row of \( A_i^s \) has now the form

\[
T_i^s = \begin{pmatrix} \mathcal{G}_i^{s-1} & \mathcal{V}_i^{s-1} & 0 & 0 & \mathcal{E}_i^{s-1} \\ \mathcal{H}_i^{s-1} & 0 & \mathcal{W}_i^{s-1} & \mathcal{F}_i^{s-1} & 0 \\ \mathcal{H}_i^{s-1} & 0 & \mathcal{W}_i^{s-1} & \mathcal{F}_i^{s-1} \end{pmatrix},
\]

where \( \mathcal{V}_j^s = \mathcal{V}_{(j-1)2^s+1}^0 \), and \( \mathcal{W}_j^s = \mathcal{W}_{j2^s}^0 \). We now eliminate \( \mathcal{G}_i^{s-1}, \mathcal{F}_i^{s-1} \) in parallel as follows: We subtract \( \mathcal{G}_i^{s-1}(\mathcal{W}_j^{s-1})^{-1} \) times the last row of \( T_i^{s-1} \) from the first row of \( T_i^{s-1} \), and \( \mathcal{F}_i^{s-1}(\mathcal{V}_j^{s-1})^{-1} \) times the first row of \( T_i^{s-1} \) from the
last row of $T_{2i-1}$. As a result we obtain

$$
T_{i}^* = \begin{pmatrix}
G_{2i-1}^* & V_{2i-1}^* & 0 & 0 & \epsilon_{2i-1}^*
\end{pmatrix}
$$

(38)

We next eliminate $\mathcal{H}_{2i}^* I_{2i}^*$, $A_{2i}^*$ in parallel in a similar way, i.e., we subtract $\mathcal{H}_{2i}^* I_{2i}^*$ times the last row of $T_{2i-1}^*$ from the last row of $T_{2i}^*$, and $\epsilon_{2i}^*$ times the first row of $T_{2i}^*$ from the first row of $T_{2i-1}^*$. The $i$th row of $A^*$ finally receives a form similar to $T_{2i-1}^*$ and $T_{2i}^*$ in (37), i.e.,

$$
T_{i}^* = \begin{pmatrix}
G_{i}^* & V_{i}^* & 0 & 0 & \epsilon_{i}^*
\end{pmatrix}
$$

(39)

**Theorem 3.1.** Let $T_{i}^*, s = 0, 1, \ldots, k$ be as in (34). Then at the end of step $s$ of the algorithm

$$
E_i^* = (V_i^*)^{-1} \epsilon_i^*, \quad F_i^* = (W_i^*)^{-1} \epsilon_i^*, \quad i = 1, 3, \ldots, \log p
$$

and

$$
G_i^* = (V_i^*)^{-1} G_i^*, \quad H_i^* = (W_i^*)^{-1} H_i^*, \quad i = 3, 5, \ldots, \log p
$$

(40) \quad (41)

**Proof.** The first $s$ steps of the algorithm may be viewed as an elimination procedure applied to the linear systems

$$
A_i^* X_i^* = \bar{E}_i, \quad A_i^* Y_i^* = \bar{F}_i, \quad i = 1, 2, \ldots, p/2^s
$$

in (15) and (16). The result in (40) and (41) now follows easily from (39). \qed

**Complexity and Processor assignments.** In Step 0, we assign processor number $i = 1, 2, \ldots, p$, to the $i$th row of $A^0$. Each processor then diagonalize its corresponding submatrix in parallel in approximately $2nm^2/p$ operations. In Step $s = 1, 2, \ldots, k$ the pair of processors

$$
(i - 1)2^s + 1, i2^s, \quad i = 1, 2, \ldots, p/2^s,
$$

is assigned to rows $2i - 1, 2i$ of $A^{s-1}$. Each such pair, performs the corresponding elimination step, exchanging $O(m^2)$ information and performing $O(m^3)$ operations. Hence, for $m, p \ll n$, the algorithm performs approximately

$$
2nm^2/p + O(m^3 \log p) \sim 2nm^2/p
$$

(43) \quad (44)

We conclude that the algorithm can be efficiently implemented on many parallel computers.

8
4. A pseudo code. Let $A$ be a band s.p.d matrix of order $n = qm$ and bandwidth $m$, and let $A = LL^T$ be the Cholesky decomposition of $A$ as in Section 2. We assume for simplicity that $q = lp$ where $p = 2^{k+1}$ is the number of processors available. We give in this Section a pseudo code for the whole algorithm consisting of the bottom-up, and top-down sweeps as in Sections 2 and 3. We further include an example for the case where $q = p = 2^3$, $k = 2$.

**Bottom-up sweep:** We compute the $E$'s, $F$'s, $G$'s, $H$'s.

- **Step $s=0$.** We compute in parallel:

\begin{align}
\begin{bmatrix}
V_1 & 0 & \xi_1^1 \\
0 & W_1 & L_2 \\
U_2 & V_2 & 0 & \xi_2^1 \\
\mathcal{H}_3 & 0 & W_2 & L_4 \\
U_4 & V_3 & 0 & \xi_3^1 \\
\mathcal{H}_4 & 0 & W_3 & L_6 \\
& & & \mathcal{H}_4 \\
& & & 0 \\
& & & W_4
\end{bmatrix}
&= F_1^0 = W_i^{-1} L_2, \\
&= G_1^0 = W_i^{-1} \xi_3^1, \\
&= F_3^0 = W_i^{-1} \xi_3^1, \\
&= G_3^0 = W_i^{-1} U_4, \\
&= H_3^0 = W_i^{-1} \mathcal{H}_4.
\end{align}

- **Step $s=1,2,\ldots,k-1$.** We perform in parallel step $s$ of the elimination procedure as in Section 3, and then compute in parallel:

\begin{align}
E_i^s &= (V_i^s)^{-1} \xi_i^s, \\
F_i^s &= (W_i^s)^{-1} F_i^s, \\
G_i^s &= (V_i^s)^{-1} G_i^s, \\
H_i^s &= (W_i^s)^{-1} H_i^s
\end{align}

$\quad i = 1,3,\ldots,p/2^s - 1.$

$\quad i = 3,5,\ldots,p/2^s - 1.$
\[ A = \begin{pmatrix}
V_1 & * & * \\
* & W_2 & F_1^2 \\
* & * & 0 \\
0 & * & V_3 \\
* & * & * \\
* & * & W_4
\end{pmatrix} \quad F_1^2 = W_2^{-1} F_1^2 \]

FIG. 4.3. Bottom-up sweep, Step 2.

- **Step \( s = k \).** We perform step \( k \) of the elimination procedure as in Section 3, and then compute:

\[ F_1^k = (W_2^k)^{-1} F_1^k. \]

**Top-down sweep:** We compute the decomposition \( A = LL^t \).

- **Step \( s = k, k-1, \ldots, 0 \).** We compute in parallel

\[ \mathcal{K}_{2^s} = F_1^s + H_2^s \mathcal{K}_{2^{s+1}} (I - G_2^s \mathcal{K}_{2^{s+1}})^{-1} E_1^s, \quad \mathcal{K}_0 = 0, \]

for \( i = 2j + 1, j = 0, 1, \ldots, p/2^{s+1} - 1. \)

- **Step \( s = -1 \).** We compute in parallel

\[ A_{i+1} = A_{i+1} - U_i \mathcal{K}_i, \]

for \( i = 1, 2, \ldots, p - 1. \) Then, we find in parallel the decompositions

\[ A_i = L_i L_i^t \quad \text{for} \quad i = 1, 2, \ldots, p. \]

Finally, we solve in parallel the linear systems

\[ L_{i,1} L_{i,2} = L_{i,1} \quad \text{for} \quad i = 1, 2, \ldots, p - 1. \]

The triangular factor \( L \) of \( A = LL^t \) is then given in in (21), see also Figure 4.7.

**Complexity and Processor assignments.** We have considered the complexity and processor assignments of the bottom-up sweep in Section 3, and we will consider here that of the Top-down sweep. In **Step \( s = k, k-1, \ldots, 0 \),** processor numbers

\[ (i - 1)2^{s+2} + 1, i2^{s+2}, \quad i = 1, 2, \ldots, p/2^{s+2} \]

compute \( \mathcal{K}_{4^i-2^{3s}}, \mathcal{K}_{4^i-12^s} \), respectively, performing \( O(m^3) \) operations and exchanging \( O(m^2) \) information in parallel. Then, in **Step \( s = -1 \),** processor \( i = 1, 2, \ldots, p \), performs approximately \( nm^2/p \) operations. Hence, for \( m, p \ll n \), the complexity is

\[ O(m^3 \log p) + nm^2/p \sim nm^2/p \]
\[ R_4 = P_1^2. \]

**Fig. 4.4. Top-down sweep, Step 2.**

\[ R_2 = P_1^1, \quad R_0 = P_0^1 + H_2^1 R_4 (I - G_2^1 R_4)^{-1} E_3^1. \]

**Fig. 4.5. Top-down sweep, Step 1.**

\[ R_{2j+1} = (I - G_{2j+1} R_{2j})^{-1} P_{2j+1}^0, \quad j = 0, 1, 2, 3. \]

**Fig. 4.6. Top-down sweep, Step 0.**

\[
L = \begin{pmatrix}
L_1 \\
L_{1,2} & L_2 \\
& \ddots & \ddots \\
& & L_{2,3} & L_3
\end{pmatrix}.
\]

**Fig. 4.7. Top-down sweep, Step -1.**
for a total of $\sim 3nm^2/p$ operations. We conclude that the speed-up is $S_p \sim p/3$ and that the algorithm is efficient, i.e. $E_p = O(1)$. Moreover, since the communication overhead is negligible, it can be implemented efficiently on many existing parallel computers.

5. An $O(\log m \log n)$ time algorithm. Let $A$ be a band s.p.d. matrix of order $n = q m$ and bandwidth $m$, and let $A = LL'$ be the Cholesky decomposition of $A$. We discuss in this section the more theoretical aspects of the algorithm, such as speed-up and efficiency for $p > q/\log q$ processors. We show that the algorithm can be implemented efficiently in $O(\log m \log n)$ time using $p \leq qM(m)/(\log m \log n)$ processors. Here, $M(m) = m^\beta, 2 \leq \beta \leq 3$, and $m^\beta/\log m$ denotes the least number of processors required in order to multiply matrices of order $m$ in $O(\log m)$ time. For example, using the standard multiplication algorithm $\beta = 3$, parallel implementation of Strassen’s method gives $\beta = \log 7 \sim 2.8$ see Chandra[6], and for huge size matrices $\beta$ can be reduced even further, see Pan and Reif[11].

We show in this section how to implement the algorithm for different numbers of processors. Given $p > q/\log q$ processors, the algorithm proceeds as for the case where there are $\min(q, p)$ processors, only now matrix operations of order $m$ are done in parallel. We show how to implement efficiently each matrix operation such as add, multiply, invert, and find the Cholesky decomposition in parallel. Let $T_{seq}$ denotes the running time of the sequential algorithm, and let $T_p$ denotes the running time of the parallel algorithm with $p$ processors. We distinguish between three cases as follows:

- $p \leq q/\log q$: We apply the algorithm as given before. The total number of operations is therefore approximately

$$3nm^2/p + O(m^3\log p) \sim 3T_{seq}/p.$$  

Hence, the speed-up is $S_p = T_{seq}/T_p \sim p/3$, and the algorithm is efficient, $E_p = S_p/p = O(1)$. Moreover, there are only about $2\log p$ communication steps, where pairs of processors exchange information of order $O(m^2)$. We conclude that the algorithm can be efficiently implemented on many parallel computers.

- $q/\log q < p \leq qM^2/\log q$: We apply the algorithm as in the case of $\min(p, q)$ processors, only now doing the corresponding matrix operations of order $m$ using as many processors as available. For example, with $q$ processors, we implement each operation in Step 0 of the bottom-up sweep with one processor, and Step $s = 1, 2, \ldots, k$, with $2^s$ processors. We further assume that each such operation is implemented using standard parallel methods, which require at least $O(m)$ time per operation. Since there are only $\sim 2\log q$ steps in the algorithm the complexity is

$$O(nm^2/p) + O(m\log q) = O(nm^2/p) \text{ time}.$$  

The total operation count is the same as before, as the standard parallel methods implement the known sequential methods. Hence, $S_p \sim p/3$ and $E_p = O(1)$.  

"
Considering the communication overhead, we note that each parallel matrix operation of order \( m \) requires approximately \( O(m) \) communication steps. This may be quite significant as \( p \) increases from \( q / \log q \) towards \( q m^2 / \log q \), and in that case we may consider the iterative method suggested below.

- \( q m^2 / \log q < p \leq q M(m) / (\log m \log n) \): We proceed as in the previous case, only now we perform complex matrix operations using only the basic matrix operations such as add, subtract, and multiply. We consider the bottom-up and the top-down sweeps separately. In the bottom-up sweep we implement the fast parallel elimination procedure described in [1, 2] whose complexity is \( O(q M(m)/p) \). Furthermore, there are only \( n^2 \) communication steps whose complexity is dominated by the time to multiply matrices of order \( m \) in parallel. In the top-down sweep we have the following:

1. We multiply matrices of order \( m \) using standard or fast parallel algorithms. The parallel complexity for this operation is

\[
(55) \quad O(M(m)/p) \text{ using } p \leq M(m)/\log m \text{ processors.}
\]

2. Consider the inverses

\[
(56) \quad (I - GR)^{-1} = (I - Z)^{-1}, \quad Z = GR,
\]

in (47), where we denote for simplicity \( G_i \) by \( G \), and \( G_{i+1} \) by \( G \). Here, we observe from [3] that \( 0 \leq \lambda(Z) < 1 \), and therefore

\[
(57) \quad (I - Z)^{-1} = \sum_{i=0}^{\infty} Z^i, \quad \| Z \| = \alpha < 1.
\]

Using the notation

\[
(58) \quad S_M = \sum_{i=0}^{M-1} Z^i = \prod_{i=0}^{\log M/2} (I + Z^{2^i}),
\]

we get that

\[
(59) \quad \| R_M \| = \| (I - Z)^{-1} - S_M \| \leq \sum_{i=M}^{\infty} \| Z \|^i \leq \alpha^M / (1 - \alpha),
\]

and the solution converges quadratically. For example, let \( \alpha = 2^{-1/210} \) and let the precision used be \( \epsilon = 2^{-t} \). Then for \( t \geq 8 \), eleven products suffices to get an accurate inverse. The actual convergence rate is clearly \( \alpha \) dependent and not \( m \) dependent. However, even for \( \alpha = 1 - 1/m^{O(1)} \) the series converges in \( O(\log m) \) steps, and the number of operations required is of order \( O(\log m M(m)/p) \). We will assume for simplicity that \( \alpha \) is independent of \( m \). Hence, the complexity and communication overhead for the above operation is the same as for matrix multiplication. The actual properties of this new iterative method needs further research.
3. We finally find the Cholesky decomposition of the dense s.p.d. matrices $A^2$ in (49), using the author new divide and conquer parallel algorithm described in[4]. A similar iterative scheme used there gives a complexity of order

$$O(M(m)/p) \quad \text{time using } p \leq M(m)/\log^2 m \quad \text{processors}.$$  

Furthermore, there are only $\sim 2\log m$ communication steps, and the overhead in Step $i = 1, 2, \ldots, \log m, \ldots, 2, 1$ is dominated by the time to multiply matrices of order $2^{i-1}$ in parallel. As a by product we obtain the inverses of the triangular factors and the triangular systems in (50) can be solved by multiplication.

We conclude that the total complexity of the algorithm, in this last case is

$$O(qM(m)/p \quad \text{using } p \leq qM(m)/\log m \log n) \quad \text{processors},$$

with $O((\log q + \log m)) = O(\log n)$ interprocessor communication steps, dominated by the time to do matrix multiplication of order $m$ in parallel. Hence, the algorithm offers $S_p = O(p)$ speed-up over sequential algorithms, and it is therefore efficient, i.e. $E_p = O(1)$. The problem of multiplying matrices was extensively studied, and many fast and efficient algorithms were suggested in the literature, see a recent survey in N.J.Higham[8]. As these methods are well adapted for parallel implementations[6, 9], we conclude that the algorithm is very suitable for parallel computations.

6. An application to the QR algorithm. We can apply the LR algorithm[14] for finding the eigenvalues of a symmetric band matrix using our algorithm straightforwardly. We further show that we can apply also the QR algorithm. The QR algorithm finds the eigenvalues of $A$ by repeatedly performing:

- Find the QR factorization $A = QR$.
- Set $A = RQ$.

until the offdiagonal elements become negligible. The resulting diagonal matrix is similar to the original matrix and therefore has the same eigenvalues. We show in this section how to implement efficiently each such step of the QR algorithm using the algorithm of the previous sections.

Let $A$ be a band symmetric non-singular matrix of order $n = qm$ and bandwidth $m$, and let $Q = QR$ be the QR decomposition of $A$. Then it is well known, see Parlett[12], that $Q$ is lower Hessenberg of bandwidth $m$, and $R$ is upper triangular of bandwidth $2m$, i.e.,

$$q_{ij} = 0, \ j < i - m, \quad \text{and} \quad r_{ij} = 0, \ j < i, \ j > i + 2m.$$
We denote $RQ$ by

$$
\begin{pmatrix}
   R_{1,1} & R_{1,2} & R_{1,3} \\
   R_{2,1} & R_{2,2} & R_{2,3} \\
   & & \ddots \\
   & & & R_{q,q}
\end{pmatrix}
\begin{pmatrix}
   Q_1 \\
   W_1 \\
   \vdots \\
   0
\end{pmatrix}
\begin{pmatrix}
   Q_2 \\
   \vdots \\
   Q_q
\end{pmatrix}
$$

(63)

where all submatrices are of order $m$, and $R_{ii}, W_i$, are upper triangular and $R_{i+2}$ is lower triangular. Hence, $RQ$ is lower Hessenberg of bandwidth $m$, but since $RQ = Q^t A Q$ is symmetric, it is again a band matrix of bandwidth $m$. We have therefore the following algorithm:

- Set $B = A^t A = A^2$, a band s.p.d. matrix.
- Find the Cholesky decomposition $B = LL'$.
- Solve the triangular system $LX = AL$.

Now, since $A = QR$ it follows that $L = R^t$ and therefore

$$
X = L^{-1} AL = R^{-1} AR^t = Q^t R^t = RQ,
$$

(64)

is the transformed matrix sought. Let us denote $X$ by

$$
X = \begin{pmatrix}
   X_1 & Y_1 \\
   Z_1 & X_2 & Y_2 \\
   & \ddots & \ddots \\
   & & Z_{q-2} & X_{q-1} & Y_{q-1} \\
   & & & \ddots & \ddots \\
   & & & & Z_{q-1} & X_q
\end{pmatrix},
$$

(65)

$Z_i, X_i, Y_i \in M(m)$, $Z_q = Y_q^t$ is upper triangular.

Next, observing that $A$ is of bandwidth $m$, and $L = R^t$ is lower triangular of bandwidth $2m$ we get

$$
\begin{pmatrix}
   E_1 & F_1 \\
   * & E_2 & F_2 \\
   * & * & E_3 & F_3 \\
   * & * & * & E_4 & F_4 \\
   & & \ddots & \ddots & \ddots \\
   & & & \ddots & \ddots \\
   & & & & \ddots & \ddots \\
   & & & & & E_{q-1} & F_{q-1} \\
   & & & & & * & E_q
\end{pmatrix}
$$

(66)

$E_i, F_i \in M(m)$, $F_i$ is lower triangular,

and therefore $X$ in $LX = AL$ can be solved in parallel as follows:

- Solve in parallel the triangular systems

$$
R_i^t Y_i = F_i, \quad \text{for } i = 1, 2, \ldots, q - 1.
$$

(67)

7. Conclusion. We have presented a practical parallel algorithm for finding the Cholesky decomposition of a general band s.p.d. matrix that does not implement the known sequential method. We have suggested a divide and conquer approach whose inherent tree structure make it simple to implement on many parallel computers. Moreover, in most practical cases the operation count is approximately only as thrice as the sequential method and the overhead due to interprocessor communication is negligible.

Cholesky decomposition is useful in solving s.p.d. systems of linear equations where the coefficient matrix is fixed and the right hand sides are changing. Once the decomposition is known the problem reduces to that of solving triangular systems for which fast systolic and parallel algorithms exist[2]. Such systems arises also in least squares solutions of overdetermined systems where the corresponding matrix is of full rank. We have also seen that the algorithm is useful in finding the eigenvalues of a symmetric band matrix where we can apply the LR algorithm directly or the QR algorithm implicitly as in the previous section. However, there are still many open problems remaining, such as:

- Counting eigenvalues - Here, we have to find the triangular factors of an indefinite symmetric matrix[12]. We therefore look for an efficient parallel algorithm for computing the $LL^T$ decomposition of a general band symmetric matrix, if it exists. Moreover, we may ask whether such an algorithm exists for the general $LU$ decomposition.

- QR with shifts - Shifts are used in order to accelerate the convergence of the algorithm. We then look for shifts that decompose the original problem into independent smaller problems.

Other numerical algorithms such as the conjugate gradient algorithm for solving sparse systems of linear equations, and Lanczos methods for finding the eigenvalues of symmetric sparse matrices are also sequential in nature and we believe that more parallel oriented algorithms for these problems can be found.

REFERENCES


