A Fast Parallel Bisection Algorithm for Symmetric Band Matrices

by

I. Bar-On

Technical Report #726
April 1992
A FAST PARALLEL BISECTION ALGORITHM FOR SYMMETRIC BAND MATRICES.

ILAN BAR-ON*

Abstract. We present a fast and practical parallel algorithm for computing few eigenvalues of a symmetric band matrix. Our main contribution is in devising a truly parallel bisection algorithm instead of the inefficient multi-section method. Our method is based on a variation of the parallel Cholesky decomposition algorithm presented in Bar-On[2] and new insights into the structure of the inertia of band symmetric matrices.

Key words. $LDL^t$ factorization, inertia, bisection, parallel algorithm.

AMS(MOS) subject classifications. 65F15

1. Introduction. Computing few eigenvalues of a matrix usually proceeds in two stages. First, a bisection strategy is used to "isolate" the eigenvalues(or clusters of eigenvalues). Then a faster converging strategy is used to "extract" the exact eigenvalue(or eigenvalues in each cluster). For example, we can use inverse iteration as proposed by Wilkinson and Reinsch[15], the Rayleigh quotient as proposed by Scott[12], or the secant method as proposed by Brent and Dekker[4]. However, in all of these schemes bisection is an important stage, slower to converge, and harder to parallelize. In this paper we present a new fast and practical parallel bisection algorithm for band symmetric systems which can be implemented on many existing parallel machines. The algorithm is much faster than multi-section, having an approximate speed-up of order $p/3$ as compared to $O(p/\log(p))$ for multi-section. In practice, this speed-up is more likely to be about $2p/3$ as the numerical examples in Section 6 shows. Furthermore, there is no need for each processor to access the whole matrix. The matrix is subdivided between the processors, allowing for larger problems to be solved.

2. Bisection and multi-section. Let $A$ be a given real symmetric matrix for which we are seeking the $k$th smallest eigenvalue. Let $[a,b]$ be an estimated interval for that eigenvalue. For example, take the interval enclosing all the Gerschgorin’s circles of $A$, see Wilkinson[14]. Let $x = (a+b)/2$ be its middle point, and let $(neg, zer, pos)$ be the inertia of $A - xI$, i.e., the number of negative, zero, and positive eigenvalues respectively. Then, in case $neg >= k$ we proceed to the left subinterval $[a, x]$. Otherwise, unless $x$ is the $k$th eigenvalue $pos > n - k$, and we proceed to the right subinterval $[x, b]$. The process of repeatedly halving the interval is called bisection. Traditionally, bisection is applied to tridiagonal matrices as proposed by Given[6] and implemented in the EISPACK routine BISECT[13]. Here, we compute the Sturm sequence, given recursively by the formula,

$$p_i(x) = (a_i - x)p_{i-1}(x) - b_i^2p_{i-2}(x), \quad i = 1, \ldots n$$

---

* Department of Computer Science, Technion, Haifa 32000, Israel. email:baron@cs.technion.ac.il
where $p_{-1} \equiv 0$, $p_0 \equiv 1$, and $b_i, a_i$ denote the sub-diagonal and diagonal elements in row $i$, i.e., $A_i = (b_i, a_i, b_{i+1})$. The number of sign changes in the sequence is then equal to the number of eigenvalues less than $x$. A more stable method, less liable to overflows and underflows, is to compute the rational factors $q_i(x) = p_i(x)/p_{i-1}(x)$, given recursively by the formula,

$$q_i(x) = (a_i - x) - b_i^2/q_{i-1}(x).$$

However, there are several drawbacks in the above methods. Firstly, block methods like the block Lanczos algorithm, which are more efficient for vectorized machines, yield banded and not tridiagonal systems. The above methods have no counterpart for the general banded case. Secondly, even for tridiagonal systems, the more stable formula (2), is inherently sequential and can not be parallelized. Hence, it is common to use a variation of bisection known as multi-section, to achieve some parallelism. Here, given $p$ processors, we subdivide the initial interval into $p + 2$ points,

$$x_i = a + i \cdot h, \quad h = (b - a)/(p + 1), \quad i = 0, \ldots, p + 1,$$

and compute in parallel the inertia at the points $x_i, i = 1, \ldots, p$. We then proceed to one of the smaller subintervals $[x_i, x_{i+1}], i = 0, \ldots, p$, accordingly. However, since one iteration of multi-section with $p$ processors, can be simulated with one processor in $\log(p)$ iterations of bisection, the method is very inefficient, i.e., $E_p = O(\log(p)/p)$. Multi-section is therefore used only in case we have idle processors to use, and is not recommended otherwise, see for example the discussion in Lo Philippe and Sameh[9]. Furthermore, in a distributive environment of parallel processing, which is the preferred environment for systems with many processors, multi-section requires each processor to hold a copy of the whole matrix. However, the memory allocated to a single processor is limited, and in practice it is even proportional to its speed, namely, to the amount of data it can process by itself more efficiently than several processors together. In that case, multi-section can not be applied to really large problems.

We present in this paper, a fast parallel bisection algorithm which is both efficient, and applicable to general banded systems. Initially, the matrix is divided evenly between the processors. Then, at the first stage of the algorithm, each processor computes a partial inertia of its sub-matrix corresponding to a partial inertia of the whole matrix, based only on local information. This stage is done completely in parallel, with no need of inter-processor communication. We then combine these partial inertias to obtain an estimate for the total inertia of the matrix. Based upon this estimate, we may be able to proceed with the next iterate of bisection. As is shown in the examples in Section 6, partial information suffice for about half of the iterates. Otherwise, we proceed to the next stage and find the complete inertia of the matrix. We present here a variation on the parallel Cholesky decomposition algorithm first presented in Bar-On[2] which proves to be very efficient and practical. We present our new parallel bisection algorithm in the next Section, analyze its theoretical properties in Section 4, describe a communication scheme for the hyper-cube architecture in Section 5, and conclude with numerical examples in Section 6.
3. A parallel bisection algorithm.

3.1. An overview. Let $A$ be a band symmetric matrix of order $n =qm$, and half bandwidth $m$. Let $M(m)$ denotes the class of matrices of order $m$, and let $A$ be presented as a $q \times q$ block matrix, with blocks of order $m$, as follows,

$$ A = \begin{pmatrix} A_1 & L_1 \\ U_1 & A_2 & L_2 \\ & \ddots & \ddots \end{pmatrix} \quad U_i, A_i, L_i \in M(m), \\ A_i \text{ symmetric, } U_i = L_i^t, \\ \text{is upper triangular.}$$

Consider the triangular decomposition of $A = LDL^t$. Here, $L$ is a unit lower triangular band matrix (with half bandwidth $m$), and $D$ is diagonal. Such a decomposition is unique and possible in case the leading minors of $A$ are non-zero[10]. Hence, from Sylvester theorem[8], the inertias of $A$ and $D$ are the same, and we can compute it from the diagonal elements of $D$. We have therefore the following Spectrum Slicing method, after Parlett[10], for implementing the "isolation" stage.

- Let $x = (a + b)/2$ be the middle point of the interval $[a, b]$.
- Compute the triangular decomposition of $A - xI = LDL^t$.
- Compute the number of negative and positive elements in $D$.
- Repeat the above for the left or right subinterval accordingly, until the eigenvalue has been isolated.

As noted in Parlett, failure of the triangular decomposition can occur in at most $n(n - 1)/2$ danger spots, in which case a small shift in $x$ will cure the problem. As such, the method is preferable to the traditional Sturm sequence method, as it is more stable, and applicable to general banded systems, see[10, page 131].

We present in this paper a parallel variation of the Spectrum Slicing algorithm. The basic algorithm is presented in the next subsection. Here we assume that both the bandwidth and the number of processors is much smaller than the order of the matrix, i.e., $m, p \ll (n/m)$. This is justified on the following grounds. The bandwidth can always be reduced by some efficient direct methods as in Schwarz[11]. Such methods are liable to parallelization in case the bandwidth is large, and the application of the Spectrum Slicing method to smaller banded systems is more efficient and stable. Existing processors which are very powerful, can then handle quite a large problem by themselves.

3.2. The basic algorithm. Let us consider the inertia of $A$ in terms of its triangular decomposition, $A = LDL^t$. Let us denote $A$ as a $2 \times 2$ block matrix, with blocks of order $n/2$, as follows,

$$ A = \begin{pmatrix} C_1 & Q^t \\ Q & C_2 \end{pmatrix} = \begin{pmatrix} \hat{L}_1 & \hat{L}_2 \\ X & \hat{L}_2^t \end{pmatrix} \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} \hat{L}_1 & X^t \\ 0 & \hat{L}_2^t \end{pmatrix} $$
Then, \( \hat{L}_1, \hat{L}_2 \) are unit lower triangular matrices of half bandwidth \( m \),

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

and \( L_x \) is a lower triangular matrix of order \( m \). Equating the sub-matrices in formula (5) we see that,

\[
C_1 = \hat{L}_1 D_1 \hat{L}_1^t, \quad C_2 = XD_1 X^t + \hat{L}_2 D_2 \hat{L}_2^t,
\]

and using \( Q = XD_1 \hat{L}_1^t \), we get \( X = Q \hat{L}_1^{-1} D_1^{-1} \hat{L}_1^{-1} Q^t = C_2 - Q C_1^{-1} Q^t = C_2' \).

Hence, the inertia of \( A \) is equivalent to the sum of the inertias of \( C_1, C_2' \). We can proceed in the same way recursively, equating the inertia of \( A \) in terms of the inertia of smaller and smaller sub-matrices, until we get \( p \) sub-matrices of order \( n/p \), \( p \) being the number of processors available. We outline the basic algorithm in the following, assuming for simplicity that the number of processors \( p \) satisfies, \( p = 2^{k+1}, q = lp \).

Let \( A^* \) denotes the matrix \( A \) with block structure

\[
A^* = \begin{pmatrix} A_1^* & L_1^* \\ U_1^* & A_2^* & L_2^* \\ \vdots & \vdots & \vdots \\ U_{v-2}^* & A_{v-1}^* & L_{v-1}^* \\ U_{v-1}^* & A_v^* \\ \end{pmatrix}, \quad v = p/2^* \quad s = 0, 1, \ldots, k + 1,
\]

\[U_i^*, A_i^*, L_i^* \in M(2^*n/p),\]

\[A_i^* \text{ symmetric band}.\]

Here, \( A_i^*, i = 1, \ldots, v \) denotes the \( i \)th principal sub-matrix of \( A^* \), namely, rows and columns \( ((i-1)2^*n/p + 1, \ldots, i2^*n/p) \) of \( A \). The off-diagonal blocks \( L_i^*, U_i^* \) are,

\[
L_i^* = \begin{pmatrix} 0 & 0 \\ L_{ii2^*} & 0 \end{pmatrix} = ( \bar{L}_i^* \ 0 ), \quad \bar{L}_i^* = \begin{pmatrix} 0 \\ L_{ii2^*} \end{pmatrix}, \quad \bar{L}_i^* \in M(2^*n/p \times m).
\]

\[
U_i^* = \begin{pmatrix} 0 & U_{ii2^*} \\ 0 & 0 \end{pmatrix} = ( 0 \ \bar{U}_i^* ), \quad \bar{U}_i^* = \begin{pmatrix} U_{ii2^*} \\ 0 \end{pmatrix}, \quad \bar{U}_i^* \in M(2^*n/p \times m).
\]

where \( M(n' \times m) \) denote the class of matrices of order \( n' \times m \).

Let, \( A^* \), like \( A^* \), denotes originally a similar block matrix. Let its principal block sub-matrices be denoted by,

\[
A_i^*, \quad i = 1, 2, \ldots, p/2^*, \quad s = k + 1, k, \ldots, 0.
\]

In what follows, these sub-matrices will become the resulting sub-matrices after \( k + 1 - s \) applications of the recursive algorithm above. Their combined inertia will give the total inertia of \( A \). We first state an important lemma.
Lemma 3.1. In step \( s = k, \ldots, 0 \) of the reduction algorithm, \( A_i^k = A_i^k \), and \( A_i^k, i = 2, \ldots, p/2^* \) is obtained from \( A_i^k \), by the modification of its first principal block of order \( m \).

*Proof.* The first claim is obviously correct. Hence,

\[
\begin{align*}
A_2^k &= A_2^k - U_2^k(A_1^k)^{-1}L_1^k \\
&= A_2^k - \left( \begin{array}{cccc}
0 & U_{q/2} & \vdots & 0 \\
0 & 0 & \ddots & 0 \\
\vdots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{array} \right) \left( \begin{array}{cccc}
0 & 0 & \cdots & 0 \\
L_{q/2} & 0 & \cdots & 0
\end{array} \right) \\
&= A_2^k - \left( \begin{array}{cccc}
0 & U_{q/2} & \cdots & 0 \\
0 & 0 & \cdots & 0
\end{array} \right) \left( \begin{array}{cccc}
* & 0 & \cdots & 0 \\
* & 0 & \cdots & \cdots
\end{array} \right) = A_2^k - \left( \begin{array}{cccc}
* & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0
\end{array} \right),
\end{align*}
\]

and the second claim is correct for \( s = k \). By induction, and the fact that each submatrix \( A_i^k \) above is of order at least \( m \), the second claim is correct for all \( s = k, \ldots, 0 \).

\[\Box\]

We outline the basic algorithm in the following.

- For \( s = k, k - 1, \ldots, 0 \) do in parallel

\[
A_i^s = A_i^s - U_i^{s-1}(A_{i-1}^s)^{-1}L_{i-1}^s, \quad i = 2j, j = 1, 2, \ldots, p/2^{s+1}.
\]

- Compute in parallel the inertia of \( A_i^0, i = 1, 2, \ldots, p \).
- Compute the inertia of \( A \) by parallel summation.

In the next section we show that in some cases partial inertia suffice, so a more simple algorithm may do as well.

3.3. Partial inertia evaluation. We consider again the bisection algorithm applied to the finding of the \( k \)th smallest eigenvalue of a matrix. Let, \([a, b]\) be the initial interval, and let \( x = (a + b)/2 \) be its middle point. Let the inertia of \( A - xI \) be given by \((neg, zer, pos)\), the number of negative, zero, and positive eigenvalues, respectively. Then, we proceed to the left subinterval if \( neg \geq k \) and to the right if \( pos > n - k \). Hence, suffice to know that there are \( neg \geq k \) negative eigenvalues, or \( pos > n - k \) positive eigenvalues in \( A - xI \), in order to decide where to proceed. The complete inertia is not always needed.

Corollary 3.2. We may denote \( A_i^0, A_i^0, i = 2, \ldots, p \) by

\[
A_i^0 = \begin{pmatrix} A_{r_i+1} & L_{r_i+1} \\ U_{r_i+1} & C_i \end{pmatrix}, \quad A_i^0 = \begin{pmatrix} * & L_{r_i+1} \\ U_{r_i+1} & C_i \end{pmatrix}, \quad r_i = \frac{q}{p}(i - 1).
\]

Then, the inertia of \( C_i \) corresponds to a partial inertia of the whole matrix \( A \).

*Proof.* By the above Lemma. \[\Box\]

Let the half bandwidth \( m \), and the number of processors \( p \), be much smaller than \( n \),
3.4. Complete inertia evaluation. We present in this subsection an efficient parallel algorithm for the reduction stage in (16). The algorithm is based on a new divide and conquer parallel algorithm for computing the Cholesky decomposition of a symmetric positive definite matrix, first presented in Bar-On[2]. The reduction stage of the algorithm can be viewed as a bottom up sweep followed by a top down sweep of a complete binary tree with $p = 2^k$ leaves, see Figure 3.1. First, in level $i = 0$, each processor in parallel diagonalizes its corresponding sub-matrix of order $O(n/p)$ in $O(nm^2/p)$ time. Then, a bottom up sweep of levels $i, i = 1, 2 \ldots k$, followed by a top down sweep of levels $i = k + 1, k \ldots 1$, is performed where the corresponding pairs of processors

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

perform $O(m^3)$ operations and interchange $O(m^2)$ information of matrices of order $m$. At this stage, the problem has been decoupled, and each processor at level $i = 0$, finds the inertia of its corresponding sub-matrix in $O(nm^2/p)$ time. In what follows, we combine the partial inertia evaluation stage of the previous subsection, with the first diagonalization step, so that the last step of computing the inertia is done in $O(m^3)$ time.

3.4.1. A bottom up sweep. Let $A$ be a symmetric matrix of order $n = qm$, and half-bandwidth $m$, and let $p = 2^{k+1}, q = l/p$, be the number of processors. We
assume, that initially, the matrix is divided evenly between the processors, each having a sub-matrix of order \( m/p \). We denote the \( i \)th row of the block matrix \( A^0 \) by,

\[
T_i^0 = \begin{pmatrix}
U_{r_i} & A_{r_i+1} & L_{r_i+1} \\
U_{r_i+1} & A_{r_i+2} & L_{r_i+2} \\
& \ddots & \ddots \\
& & U_{r_i+l-2} & A_{r_i+l-1} & L_{r_i+l-1} \\
& & & U_{r_i+l-1} & A_{r_i+l} & L_{r_i+l}
\end{pmatrix}, \quad r_i = \frac{m}{p}(i-1)
\]

and assign it to processor \( i, i = 1, \ldots, p \). We further use the following notations,

\[
\begin{align*}
A_i^sX_i^s &= \widetilde{L}_i^s, \\
X_i^s &= \begin{pmatrix}
E_i^s \\
F_i^s
\end{pmatrix}, \\
A_i^sY_i^s &= \widetilde{U}_{i-1}^s, \\
Y_i^s &= \begin{pmatrix}
G_i^s \\
H_i^s
\end{pmatrix},
\end{align*}
\]

In the following we compute the sub-matrices \( E_i^s, F_i^s, G_i^s, H_i^s \), as above.

- **Step 0.** We let processor \( i, i = 2, \ldots, p \), eliminates \( L_{r_i+1} \), by upward block Gaussian elimination, while evaluating the inertia of \( C_i \) in (17), as follows,

1. Let, \((\text{neg}_i, \text{zer}_i, \text{pos}_i) = (0, 0, 0)\).
2. Let, \( V_i^0 = A_{r_i+l} \), and \( E_i^0 = L_{r_i+l} \).
3. For \( j = l-1, \ldots, 1 \) compute,

\[
\begin{align*}
(neg_i, zer_i, pos_i) &= (neg_i, zer_i, pos_i) + \text{Inertia}(V_i^0), \\
E_i^0 &= -L_{r_i+j}(V_i^0)^{-1}E_i^0, \\
V_i^0 &= A_{r_i+j} - L_{r_i+j}(V_i^0)^{-1}U_{r_i+j}.
\end{align*}
\]

At the same time we let processor \( i = 1 \), eliminate \( U_{r_i+l-1} \) by downward block Gaussian elimination, while evaluating the inertia of \( A_i^0 \) in (9). The partial inertias \((\text{neg}_i, \text{zer}_i, \text{pos}_i), i = 1, \ldots, p \), so computed, are then combined to give the partial inertia of \( A \). In case this suffice, we proceed with the next iterate of bisection. Otherwise, we proceed, with the current algorithm, letting processors \( i = 2, \ldots, p-1 \) eliminate \( U_{r_i+l-1} \) in a similar way. We then obtain the following transformed matrix,

\[
T_i^0 = \begin{pmatrix}
G_i^0 & V_i^0 & 0 & 0 & E_i^0 \\
0 & * & 0 & 0 & \mathcal{F}_i^0 \\
H_i^0 & 0 & 0 & \mathcal{V}_i^0 & \mathcal{F}_i^0 \end{pmatrix}, \quad \mathcal{F}_i^0 = L_{r_i+1}, \quad G_i^0 = U_{r_i}, \\
& i = 1, \ldots, p,
\]

in which the star denotes the original in between rows. Note that \( \mathcal{G}_i^0, \mathcal{H}_i^0 \) are missing, and that \( E_i^0 \equiv 0 \) in \( T_i^0 \). Similar observations hold for \( T_p^0 \).
We conclude with the computation of the $E_i$'s, $F_i$'s, $G_i$'s, $H_i$'s.

**Theorem 3.3.** Let $T_i^s, s = 0, 1, \ldots, k$ be as in (28). Then $E_i^s, F_i^s, G_i^s, H_i^s$ can be calculated at the end of step $s$ of the algorithm as follows,

\[
E_i^s = (V_i^s)^{-1} E_i^s, \quad F_i^s = (W_i^s)^{-1} F_i^s, \quad s = 0, 1, \ldots, k, \quad i = 1, 3, \ldots, p/2^s - 1
\]
and

\[ G_s^i = (\mathcal{V}_s^i)^{-1}G_s^i, \quad H_s^i = (\mathcal{W}_s^i)^{-1}H_s^i, \quad s = 0, 1, \ldots, k - 1, \quad i = 3, 5, \ldots, p/2^s - 1. \]

**Proof.** See Bar-On[2]. \(\Box\)

We note that the inverses above were already calculated at the first step of the bottom-up sweep, so that only matrix product are involved here.

### 3.4.2. A top down sweep

The \(E's, F's, G's, H's\) computed in the bottom up sweep serve in devising a simple recurrence formula for evaluating the complete inertia of the matrix. This algorithm is a variation on the algorithm presented in Bar-On[2], and more details and proofs can be found there.

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

\[ N_{i2^s} = F_s^i + H_s^i N_{i2^s+1} (I - G_s^i N_{i2^s+1})^{-1} E_s^i, \quad N_0 \equiv 0, \]

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

- We compute in parallel,

\[ \mathcal{V}_i^0 = \mathcal{V}_i^0 - U_r N_{i-1}, \quad i = 2, \ldots, p. \]

- We then compute in parallel,

\[ (\text{neg}_i, \text{zer}_i, \text{pos}_i) = (\text{neg}_i, \text{zer}_i, \text{pos}_i) + \text{Inertia}(\mathcal{V}_i^0), \]

for \(i = 2, \ldots, p\), and finally, obtain the complete inertia by parallel summation.

### 4. Theoretical Analysis

#### 4.1. Complexity

We assume for simplicity that matrix inversion is much more time consuming than multiplication, that inertia evaluation can be computed during inversion, and ignore the other basic matrix operations. This is quite realistic given current vectorize processors, see for example[3]. Let \(c_1, c_2\) denote the running time for inversion and multiplication of matrices of order \(m\), respectively. Let \(T_{seq}\) denotes the running time of the block sequential Spectrum Slicing algorithm, and let \(T'_p, T_p\) denote the running time of the partial and complete parallel algorithm with \(p\) processors. Then, from (24) we get,

\[ T_{seq} \sim q(2c_1 + c_2) = qc_1(2 + d), \]

where \(d = c_2/c_1\). Similarly, from (23)-(24) we get,

\[ T'_p \sim (q/p)(3c_1 + c_2) = (q/p)c_1(3 + d), \]
so that the speed-up for the partial inertia evaluation algorithm is,

\[ S_p' \sim \frac{2 + d}{3 + d} \geq 90p, \]

for \( c_2 \geq 7c_1 \), which is almost optimal. For the complete inertia algorithm, we get from (29)-(36), that

\[ T_p \sim 2(q/p)c_1(3 + d) + \log(p)(8c_1 + c_2) + \log(p)(4c_1 + c_2) \]

\[ = 2c_1[(q/p)(3 + d) + \log(p)(6 + d)] \leq 3c_1(q/p)(2 + d), \]

provided \( p \log(p) \leq q/4 \), \( d \geq 6 \). Hence, for \( m, p \ll q \), we get an approximate speed-up of order at least \( p/3 \). Communication overhead depends on the architecture at hand, but as is shown in the next Section, given an underlying tree structure, communication can proceed with computation and the additional overhead could be consumed by the above formulas.

4.2. Correctness. Let \( A = A' - xI \) be indefinite, and assume we have control on the shift \( x \). The following Lemma holds.

**Lemma 4.1.** Let the sub-matrices, \( A_i^s, i = 1, \ldots, p/2^s, s = 1, \ldots, k \), as in (9), be non-singular. Consider, \( A_i^0, i = 1, \ldots, p \), as a block matrix with blocks of order \( m \). Let each of its leading block sub-matrices and each of its trailing block sub-matrices be non-singular, i.e.,

\[ A_i^0(1, j) = \begin{pmatrix} A_{r_i+1} & L_{r_i+1} \\ U_{r_i+1} & A_{r_i+2} & L_{r_i+2} \\ & \ddots & \ddots & \ddots \\ U_{r_i+j-2} & A_{r_i+j-3} & L_{r_i+j-2} \\ & \ddots & \ddots & \ddots \\ U_{r_i+j-1} & A_{r_i+j-1} \\ & \ddots & \ddots & \ddots \\ & & U_{r_i+l-2} & A_{r_i+l-3} & L_{r_i+l-2} \\ & & \ddots & \ddots & \ddots \\ & & & U_{r_i+l-1} & A_{r_i+l-1} \end{pmatrix}, \]

\[ A_i^0(j, l) = \begin{pmatrix} A_{r_i+j} & L_{r_i+j} \\ U_{r_i+j} & A_{r_i+j+1} & L_{r_i+j+1} \\ & \ddots & \ddots & \ddots \\ U_{r_i+l-2} & A_{r_i+l-3} & L_{r_i+l-2} \\ & \ddots & \ddots & \ddots \\ & & U_{r_i+l-1} & A_{r_i+l-1} \end{pmatrix} \]

where, \( r_i = \frac{q}{p}(i - 1) \), and \( j = 1, \ldots, l \). Furthermore, let the leading principal block sub-matrices of \( A^0 \) in (9) be non-singular, i.e.,

\[ A^0(1, j) = \begin{pmatrix} A^0 & L^0 \\ U^0 & A^0 \end{pmatrix}, \]

\[ \cdots \]

\[ \begin{pmatrix} U_{j-2} & A^0_{j-1} & L^0_{j-1} \\ U_{j-2} & A^0_{j-1} & L^0_{j-1} \end{pmatrix} \]
for \( j = 1, \ldots, p - 1 \). Then, the algorithm is mathematically correct.

**Proof.** Consider Step 0 of the bottom-up sweep. Here, the non-singularity of \( A^0_0(j+1, l), j = l-1, \ldots, 1 \), insures the existence of the inverse \( (\mathcal{V}_0^0)^{-1} \) and the correctness of the upward block Gaussian elimination algorithm. Similarly, the non-singularity of \( A^0_0(1, j - 1), j = 2, \ldots, l \), insures the correctness of the downward block Gaussian elimination algorithm. The non-singularity of \( A^0_0 \) insures the existence of the inverses \( (\mathcal{V}_s^0)^{-1} = (\mathcal{V}_{2s-1}^0)^{-1} \), and similarly, \( (\mathcal{W}_s^0)^{-1} = (\mathcal{W}_{2s-1}^0)^{-1} \), in Step \( s = 1, \ldots, k \). Finally, the non-singularity of \( A^*_i \), insures the existence of the inverses of \( \mathcal{R}_i \), and its counterpart in these steps. Next, consider the recurrence formula in (35), for the top-down sweep. Here, we imply from Theorem 2.1 in Bar-on[2] that the existence of inverses \( A^*_0(r) \) insures the existence of the inverse \( \mathcal{V}_1^0 \), and the correctness of the algorithm.

**Corollary 4.2.** There are at most,

\[
(46) \quad n \frac{q}{p} + n \log(p) + \frac{1}{2} np < nq,
\]

"dangerous spots" for which the algorithm may fail.

**Proof.** The statement will be satisfied unless \( x \) in \( A = A' - xI \) is equal to one of the eigenvalues of the corresponding sub-matrices in Lemma 4.1. For the second requirement in (43) and (44), there are

\[
(47) \quad p(2m(1 + 2 + \cdots \left( \frac{q}{p} - 1 \right)) + \frac{n}{p}) = n\left( \frac{q}{p} - 1 \right) + n = n \frac{q}{p},
\]

"dangerous spots". For the first requirement on \( A^*_i \), there are

\[
(48) \quad \frac{n}{p} \left( \frac{2p^2}{2} + \frac{4p^2}{4} + \cdots + \frac{p^2}{2} \right) \leq n \log(p)
\]

"dangerous spots". For the final requirement in (45), there are more

\[
(49) \quad \sum_{s=1}^{k} \sum_{j=1}^{2s-1} \left( 2^{\frac{s}{p}} \frac{n}{p} + j \frac{n}{p} \right) = 3 \sum_{s=1}^{k} \frac{n}{p} \sum_{j=1}^{2s-1} \left( 2^s - 1 \right) = \frac{1}{2} \frac{n}{p} (p^2 - 3p + 2) < \frac{1}{2} np
\]

additional "dangerous spots". Hence, the total number of "dangerous spots" is no more than,

\[
(50) \quad n \frac{q}{p} + n \log(p) + \frac{1}{2} np = nq\left( \frac{1}{p} + \frac{p + 2 \log(p)}{2q} \right) \leq nq.
\]

\[\square\]

4.3. Stability. Consider the sequential Spectrum Slicing method as discussed in Parlett[10, pages 48-49]. As long as the element growth in Gaussian elimination is tolerable, the algorithm is reasonably stable. Hence, as long as \( x \) is not too close to one
of the finite number of "dangerous spots" in our case, the algorithm is stable. Since these "dangerous" eigenvalue usually differ from the eigenvalue of the whole matrix, we may restart the current iteration with a slightly modified shift \( x \), as soon as we detect an extraordinary element growth. We can insure better stability using pivoting. For example, consider Step 0 of the bottom-up sweep. Here, we may compute the Bunch-Parlett factorization of the symmetric indefinite matrix \( C_i \) in (17), see Golub and Van Loan[7, page 166], that is,

\[
P_iC_iP_i^t = L_iD_iL_i^t, \quad C_i = P_i^tL_iD_iP_i.
\]

The partial inertia of \( A_i^0 \) is then calculated stably from the diagonal blocks of order \( 1 \times 1 \), and \( 2 \times 2 \) of \( D_i \). Similarly, we obtain the top row in (25) by,

\[
\nu_i^0 = A_{r_i+1} - \left( \begin{array}{ccc} L_{r_i+1} & 0 & \cdots \\
0 & 0 & \cdots \\
\vdots & \ddots & \ddots \\
0 & \cdots & 0
\end{array} \right) P_i^tL_i^{-t}D_i^{-1}L_i^{-1}P_i \left( \begin{array}{ccc} L_{r_i+1} & 0 & \cdots \\
0 & 0 & \cdots \\
\vdots & \ddots & \ddots \\
0 & \cdots & 0
\end{array} \right)
\]

(52) \[ E_i^0 = - \left( \begin{array}{ccc} L_{r_i+1} & 0 & \cdots \\
0 & 0 & \cdots \\
\vdots & \ddots & \ddots \\
0 & \cdots & 0
\end{array} \right) P_i^tL_i^{-t}D_i^{-1}L_i^{-1}P_i \left( \begin{array}{ccc} 0 & 0 & \cdots \\
0 & 0 & \cdots \\
\vdots & \ddots & \ddots \\
0 & \cdots & 0
\end{array} \right)
\]

(53) where we implicitly use \( L_i^{-1} \) by solving the corresponding triangular system. In a similar way we can obtain the bottom row, and Step 0 is stable. We can use pivoting, or even orthogonal transformations in the other elimination steps of the algorithm, but this seems to be unnecessary. As is shown in the examples of Section 6, the simple algorithm with no pivoting at all, gives the correct "isolating" interval. Moreover, the method can give very accurate results if desired.

5. Communication on the Hyper-cube. Let \( \overrightarrow{S_i} \) be a sequence of \( 2^i \) numbers defined as follows:

\[
\overrightarrow{S_i} \overset{\text{def}}{=} \begin{cases} 
\{0,1\} & \text{if } i = 1 \\
\{0[\overrightarrow{S_{i-1}}],1[\overrightarrow{S_{i-1}}]\} & \text{if } i > 1
\end{cases}
\]

(54) Here \( \overrightarrow{S_i} \) (with left arrow) has \( S_i \)'s numbers in a reversed order. \( 0[\overrightarrow{S_i}] \) (or \( 1[\overrightarrow{S_i}] \)) is a sequence obtained by appending bit 0 (or 1) before each number in \( \overrightarrow{S_i} \). Every number in \( \overrightarrow{S_i} \) is represented by \( i \) bits. For example: \( \overrightarrow{S_2} = \{00,01,11,10\} \) and \( \overrightarrow{S_3} = \{000,001,011,010,110,111,101,100\} \). Let us divide \( \overrightarrow{S_{k+1}} \) to any \( 2^s \), \( s \leq k + 1 \), equal sized subsequences, then it is clear that within each subsequence the two extreme and the two middle numbers differ in exactly one bit. By the \( k \)-cube definition [1] a cube's corner, with \( k \) bits address, is connected to \( k \) other corners whose binary address differ from its own in one bit. Let us assign the ith logical processor to the ith number in the sequence \( \overrightarrow{S_{k+1}} \).

Consider Step \( s = 1, \ldots, k \) of the bottom-up sweep in Subsection 3.4. Let, \( p_{11}, p_{12}, p_{13}, p_{14} \) denote the corresponding logical processors involved in the computation of \( T_s^* \), i.e., the logical processors assigned to the top and bottom rows of \( T_{2i-1}^* \), and \( T_{2i}^* \), respectively. Then, \( p_{11} \) is connected to \( p_{12}, p_{13} \) to \( p_{14} \), and \( p_{12} \) to \( p_{13} \), by the above assignment. Hence, the computation in (29)-(32) can proceed as required. For the top down sweep, we state
the following Lemma.

**LEMMA 5.1.** We can let logical processor \( p_i^s = i2^s \) computes \( R_{i2^s} \) in (35), and then send it to processor \( p_i^s + 1 \), to which it has a direct link.

**Proof.** We prove that this is possible by induction. First, it is certainly possible for \( s = k \), since \( R_{k/2} = F_{k}^k \) was computed by \( p_i^k = p/2 \) in Step \( k \) of the bottom-up sweep. This is further possible for \( R_{2^s} = F_{2^s}^s \), \( s = k - 1, \ldots, 0 \) for the same reasons. Assume this is possible for Step \( s + 1 \). Then in Step \( s \), processor \( p_i^s \) has already computed \( H_i^s, F_i^s \), and processor \( p_i^{s+1} \) has already computed \( G_i^s, E_i^s \), for \( i = 2j - 1, j = 2, \ldots, p/2^{s+1} \), Moreover, \( p_i^{s} + 1 \), has already received \( R_{j2^{s+1}} = R_{(i-1)2^s} \), from \( p_i^{s-1} \). But \( p_i^s \), and \( p_i^{s+1} + 1 \), have direct communication link. \( \square \)

6. Numerical examples. We present in this Section some test results obtained by simulating the parallel algorithm. In each case, we have searched for the middle eigenvalue, using the parallel algorithm as presented in Section 3, using no pivoting at all. We have run three tests, the tridiagonal matrix \( A = (-1,2,-1) \) with \( n = 64000 \), and random matrices of bandwidth \( m = 2, 4, 8 \). In each case we have run the sequential Spectrum Slicing algorithm to obtain an estimate for the eigenvalue and its close neighbors. These are recorded on the first row and column of each Table. We then ran the simulation with \( p = 2, 4, 8, 16 \) processors. The number of iterations recorded are for the first interval to isolate the eigenvalue, and then for the first interval of length less than \( 10^{-6} \). The number of iterations for which full inertia was computed is recorded in the corresponding column of each processor.

<table>
<thead>
<tr>
<th>Closest Eigenvalues: 1.999853, 2.000049</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>(1.999923,2.000045)</td>
</tr>
<tr>
<td>1.999951</td>
</tr>
</tbody>
</table>

**Table 1**

**Tridiagonal matrix** \( A = (-1,2,-1), n = 64000 \).

<table>
<thead>
<tr>
<th>Closest Eigenvalues: 1.209446, 1.209844</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>(1.209515,1.209596)</td>
</tr>
<tr>
<td>1.209592</td>
</tr>
</tbody>
</table>

**Table 2**

**Random matrix** \( n = 32000, m = 2 \).

Considering the results obtained we first note that an "isolating" interval could be found with less than half the number of full iterates. Moreover, a very close estimate can be found, as accurate as the one for the sequential Spectrum Slicing stable algorithm, using more iterates. Here too, the total number of full iterates is about half. We may
therefore estimate the speed-up of the whole algorithm by

\[ S_p \sim \frac{1}{2} p + \frac{1}{2} p = 2p/3, \]

so that the efficiency is about \( E_p \sim 66\% \).

7. Conclusion. We have presented a new application to the divide and conquer parallel algorithm for computing the Cholesky decomposition of a band symmetric matrix. The parallel bisection algorithm so presented is very fast, efficient, and can be implemented on many existing parallel computers. Hence, it may be used for finding few eigenvalues, or all eigenvalues in a given interval, of a matrix with a small bandwidth. Traditionally, bisection is applied to a band matrix by first reducing it to a tridiagonal form, a process with almost no speed-up for a small bandwidth. The current paper suggests that better speed-up can be obtained by working directly with the original matrix. Other applications of the divide and conquer approach presented in Bar-On[2], should further be studied.

REFERENCES