A Fast Parallel Cholesky Decomposition Algorithm for Tridiagonal Symmetric Matrices*

Ilan Bar-On† Bruno Codenotti‡ Mauro Leoncini§

TR-95-006

February 1995

Abstract

In this paper we present a new fast and stable parallel algorithm for computing the Cholesky decomposition of real symmetric and positive definite tridiagonal matrices. This new algorithm is especially suited for the solution of linear systems and for computing a few eigenvalues of very large matrices. We demonstrate these results on the Connection Machine CM5, where we obtain a very satisfactory performance. We finally note that the algorithm can be generalized to block tridiagonal and band systems.

Key words. Parallel algorithm, Cholesky decomposition, LR and QR algorithms, Eigenvalues, Symmetric, Tridiagonal, and Band Matrices.

AMS(MOS) subject classifications. 65F15

* Research produced with the help of the National Science Foundation, Infrastructure Grant number CDA-8722788, and the Cooperation Agreement CNR-NCRD.

† Department of Computer Science, Technion, Haifa 32000, Israel.

‡ Istituto di Matematica Computazionale del CNR, Via S. Maria 46, 56126 Pisa, Italy. Research partially supported by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM).

§ Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy. Research supported by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM), and by Italian M.U.R.S.T. 40% funds.

1. Introduction. We consider the problem of computing the Cholesky decomposition of very large real symmetric tridiagonal matrices. In addition to being one of the most natural approaches to the solution of positive definite linear systems, this decomposition is a valuable tool in many diagonalization techniques for computing eigenvalues and singular values of matrices. Rutishauser's cubically convergent LR algorithm is based on the iterative application of Cholesky decomposition [21]. The divide and conquer approach can also be combined with it [6, 7, 8]. More recently, the Cholesky decomposition, or a variation of it, has been used in connection with the accurate computation of the singular values of bidiagonal matrices [12, 15], and of the eigenvalues of specially structured symmetric tridiagonal matrices [10]. Moreover, it has been shown that Francis' QR algorithm (see [16, 17]) can be implemented using a band Cholesky decomposition [3, 4, 5]. We also point out that, despite the amount of work devoted to the parallel solution of tridiagonal linear systems (see, for instance, [2, 13, 23, 24, 26]), the use of Cholesky decomposition in such a context has not received a great deal of attention.

The classical serial algorithms for computing the Cholesky decomposition cannot be efficiently parallelized, nor directly vectorized. It is therefore natural to seek an algorithm directly amenable for an efficient parallel implementation. In this paper we introduce a new algorithm based on a preprocessing stage that subdivides the original problem into independent smaller subproblems that can be solved in parallel. The preprocessing stage makes use of block-cycling and prefix sum computation techniques [11]. Our parallel algorithm is computationally efficient, scalable, and can be efficiently vectorized. From a theoretical viewpoint, our algorithm computes the Cholesky factors of an order N symmetric tridiagonal matrix in time $O(n)$ using p processors, where $N = np$ and $n = \Omega(\log N)$. We have also implemented our algorithm on a CM5 parallel supercomputer with 32 nodes [27], where we have obtained a speedup of three orders of magnitudes over the sequential algorithm described in Section 4.

Our parallel algorithm is usually pointwise stable. Let θ be the relative working precision. Then the relative perturbations are bounded by $\theta' = \theta\frac{\lambda_1^2}{\lambda_N^2}$ where λ_1 and λ_N are the largest and smallest eigenvalues, respectively. Also, in many practical cases we observe that $\theta' = \theta\frac{\lambda_1}{\lambda_N}$. For instance, if $\lambda_N/\lambda_1 \approx \mu$, and we want the eigenvalues with a relative precision μ, then we store the original matrix in double precision (i.e. $\theta' = \mu^2$), and we do the computation in quadruple precision (i.e. $\theta = \mu^4$). Moreover, we can verify the accuracy of the computed decomposition with no additional cost, and thus tailor the precision to the best of our needs.

This paper is organized as follows. In Section 2 we define concepts and notations used throughout the rest of the paper. In Section 3 we review the LR algorithm for computing the eigenvalues of symmetric tridiagonal matrices. This will provide a motivation to the development of a parallel algorithm for computing the Cholesky decomposition of such matrices. In Section 4 we describe a sequential algorithm that computes the Cholesky factors and discuss its implementation, computational cost, and numerical accuracy. In Section 5 we describe the parallel algorithm, providing details of the preprocessing stage, and, in Section 6, we analyze its computational cost and suitability to vectorization. In Section 7 we present the experimental results obtained on the CM5, and in Section 8 we discuss the numerical accuracy of the algorithm. Finally, we mention some applications of

the algorithm to the implementation of the QR algorithm, and state some open problems.

2. Definitions and Main Notations. We denote by $\mathcal{R}^n$ the set of real vectors of order n and by

$$e_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T,$$

where the 1 is in the ith position, $i = 1, \ldots, n$, the standard basis for this vector space. When needed, we emphasize that a particular vector e_i is in $\mathcal{R}^n$ by writing $e_i^{(n)}$.

We denote by $\mathcal{M}(n)$ the set of real matrices of order n, and by A^T the transpose of A.

We denote a tridiagonal symmetric matrix $T \in \mathcal{M}(n)$ by

$$T = \begin{pmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & b_3 & & \ddots & \\ & & \ddots & & b_n \\ & & & b_n & a_n \end{pmatrix}. \tag{1}$$

In this paper we assume that T is unreduced, that is, $b_i \neq 0,\ i = 2, \ldots, n$.

We say that a nonsingular matrix $P \in \mathcal{M}(m)$ is a *cyclic transformation* if

$$P = \begin{pmatrix} H & 0 \\ h^T & 1 \end{pmatrix}, \qquad \begin{array}{c} H \in \mathcal{M}(m-1), \\ h \in \mathcal{R}^{(m-1)}. \end{array}$$

Note that H must be nonsingular.

We say that the computation of the Cholesky decomposition of a matrix A is *pointwise stable* if the computed Cholesky factors are the exact decomposition of a pointwise perturbation of A.

We measure the theoretical running time of a sequential algorithm by counting the number of *flops*, i.e., floating point operations. We also refer to the flop count as to the number of *arithmetic steps*. The running time of a parallel algorithm implemented on a p processor machine is the maximum, over the p processors, of the number of steps performed. We refer to this measure as to the number of *parallel steps*. It is plain that the actual running time can be highly affected, on vector pipeline computers (both sequential and parallel), by the suitability of the algorithm to vectorization.

The *speedup* of a parallel algorithm A over a sequential algorithm B is the ratio

$$S_p(n) = \frac{T_B(n)}{T_{A,p}(n)},$$

where $T_B(n)$ is the running time of B on inputs of size n while $T_{A,p}(n)$ is the running time of A on inputs of size n with p processors. It obviously holds that $S_p(n) \leq p$, for otherwise a sequential simulation of the parallel algorithm would beat the (supposedly) best known sequential one. However, the upper bound on the speedup does not necessarily holds in the presence of inner parallelism of the hardware, i.e. in case of vector and pipeline architectures.

The *work* done by a parallel algorithm running in time T on a p processor machine is defined as the product $W = Tp$. When the work done by a parallel algorithm equals the running time of the best sequential algorithm for the same problem we say that the parallel algorithm is (theoretically) *work optimal*.

3. An overview of the LR algorithm. The sequential LR algorithm developed by Rutishauser was termed by Wilkinson as "the most significant advance which has been made in connexion with the eigenvalue problem since the advent of automatic computers" (see [29], p. 485). For computing the eigenvalues of tridiagonal symmetric matrices, this algorithm is very simple and efficient, with a cubic convergence.

The LR algorithm iteratively computes a sequence of tridiagonal matrices that gradually converge to a diagonal matrix with the same eigenvalues. Starting with the original matrix $A_0 = A$ and with $eig = 0$, for $s = 0, 1, \ldots$, the sth step consists of the following stages:

- choose a shift y_s;
- find the Cholesky decomposition of $B_s = A_s - y_s I = L_s L_s^T$;
- set $A_{s+1} = L_s^T L_s$ and $eig = eig + y_s$.

As soon as the last off diagonal element becomes negligible, eig is a new exposed eigenvalue. It is easy to see that the third stage of the above algorithm can be efficiently parallelized. In addition, after a few steps, the shifts y_s in the first stage can be read off the last diagonal element of the matrix (see Rutishauser and Schwarz [22]). It follows that the Cholesky decomposition is the main difficulty in implementing the LR algorithm on a parallel machine. This is a major motivation to focus our attention on the development of an efficient parallel implementation of Cholesky decomposition. For further discussions on the LR algorithm the reader is encouraged to see Wilkinson [29], Parlett [20], Grad and Zakrajsek [19], and Bar-On [3, 5, 8].

4. Cholesky decomposition. In this section we describe a sequential algorithm to compute the Cholesky decomposition of a symmetric tridiagonal matrix which is particularly suitable for the use in the LR algorithm, and analyze its computational and numerical properties.

Consider the Cholesky decomposition stage in the LR algorithm described in Section 3, and let (1) be the matrix to be factored. We have that

$$T_0 = T = \begin{pmatrix} d_1 & & & & \\ y_2 & d_2 & & & \\ & y_3 & d_3 & & \\ & & \ddots & \ddots & \\ & & & y_n & d_n \end{pmatrix} \begin{pmatrix} d_1 & y_2 & & & \\ & d_2 & y_3 & & \\ & & \ddots & \ddots & \\ & & & d_{n-1} & y_n \\ & & & & d_n \end{pmatrix} = LL^T. \tag{2}$$

Instead of computing the decomposition (2), and considering that this process must be repeatedly applied over LR iterations, we compute the quantities x_i and z_i using the following recurrences:

$$\begin{aligned} z_1 &= 0, \\ x_1 &= a_1, \\ z_i &= b_i^2 / x_{i-1}, \quad i = 2, \ldots, n, \\ x_i &= a_i - z_i, \quad i = 2, \ldots, n. \end{aligned} \tag{3}$$

Note that in the recurrences (3) we only use the a_i's and b_i^2's (rather than the b_i's). It can

be easily proved by induction that $x_i = d_i^2$ and $z_i = y_i^2$. Now, if we set

$$T_1 = L^T L = \begin{pmatrix} f_1 & g_2 & & & \\ g_2 & f_2 & g_3 & & \\ & g_3 & & \ddots & \\ & & \ddots & & g_n \\ & & & g_n & f_n \end{pmatrix},$$

then we can efficiently compute the quantities f_i and g_i^2 as follows:

$$\begin{array}{rcll} f_1 & = & x_1 + z_2, & \\ g_i^2 & = & z_i * x_i, & i = 2, \ldots, n, \\ f_i & = & x_i + z_{i+1} & i = 2, \ldots, n, \end{array}$$

where it is assumed that $z_{n+1} = 0$.

This process can therefore be iterated. If needed, the elements of the matrix T_i (implicitly) generated at the ith step of the LR algorithm can be easily recovered. By using this variant of the Cholesky decomposition, which we call *revised* decomposition, we avoid the computation of square roots.

Complexity. The computation of the revised decomposition requires n additions and multiplications, and therefore $2n$ flops. In Table 1 we compare the typical running times of some computational routines on a DEC Alpha 7000 Model 660 Super Scalar machine. The BLAS routine "dgemm" performs matrix multiplication, the LAPACK routines "dpotrf" and "dpbtrf" [1] perform the Cholesky decomposition on dense and tridiagonal matrices respectively, and the private routine "trid" performs the above revised decomposition. The

Routine	n	Flops	Time	Mflops
dgemm	400	$2 * n^3$	0.95	135.48
dpotrf	600	$2 * n^3/6$	0.99	72.11
dpbtrf	200000	$2 * n$	1.01	0.39
trid	200000	$2 * n$	0.08	5.00

TABLE 1
LAPACK Computational Routines

Mflops column puts into evidence the rather inefficient use of computing power made by the sequential algorithms for tridiagonal matrices. This is mainly due to the presence of very short vectors (cfr [18], p. 156), but also to the fact that, in view of the use of shifts in the LR algorithm, it is not possible to know a priori if the decomposition will succeed, and this makes it necessary to check in (3) if $x_i > 0$.

Numerical Stability. Cholesky decomposition is pointwise stable. Usually the entries of the given matrix are known up to some perturbation so that it is very useful to investigate the "structure" of the perturbations introduced by rounding. To show this, let us denote the computed value of a by $\hat{a} = fl(a)$, and assume that the standard operations satisfy

$$fl(a \ op \ b) = (a \ op \ b)(1 + \eta), \quad |\eta| \leq \theta,$$

where *op* stands for $+, -, *$, or $/$, and θ is the machine relative precision. For example, $\theta \sim 10^{-16}$ in standard double precision. Then the actual computation of the decomposition can be formulated as follows:

$$\begin{aligned} \hat{z}_i &= c_i(1+\beta_i)/\hat{x}_{i-1} = \hat{c}_i/\hat{x}_{i-1}, \quad i = 1, \ldots, n, \\ \hat{x}_i &= (a_i - \hat{z}_i)(1+\alpha_i') = \hat{a}_i - \hat{z}_i, \quad i = 1, \ldots, n. \end{aligned}$$

where $c_i = b_i^2$, $\hat{x}_0 = x_0$, $|\beta_i| \leq \theta$ and $\hat{a}_i = a_i(1+\alpha_i)$ with $|\alpha_i| \leq |\alpha_i'| \leq \theta$. For the classical error bounds of the Cholesky decomposition one can see [28] and [14].

5. Parallel Cholesky decomposition.

5.1. Mathematical formulation. Let $T \in \mathcal{M}(n)$ be the unreduced symmetric tridiagonal matrix in (1). In block notation, T can be written as

$$T = \begin{pmatrix} T_1 & U_2^T & & & \\ U_2 & T_2 & U_3^T & & \\ & U_3 & \ddots & \ddots & \\ & & \ddots & \ddots & U_q^T \\ & & & U_q & T_q \end{pmatrix}, \qquad \begin{array}{c} T_i \in \mathcal{M}(n_i), \\ \sum_{i=1}^{q} n_i = n, \\ m_i = \sum_{j=1}^{i} n_j, \\ U_{i+1} = b_{m_i+1} e_1^{(n_{i+1})} \left(e_{n_i}^{(n_i)}\right)^T, \end{array}$$

and the Cholesky factors of the decomposition in (2) as

$$L = \begin{pmatrix} L_1 & & & & \\ R_2 & L_2 & & & \\ & R_3 & L_3 & & \\ & & \ddots & \ddots & \\ & & & R_q & L_q \end{pmatrix}, \qquad \begin{array}{c} L_i \in \mathcal{M}(n_i), \\ R_{i+1} = y_{m_i+1} e_1^{(n_{i+1})} \left(e_{n_i}^{(n_i)}\right)^T. \end{array}$$

By equating T with the product LL^T, we see that

$$T_1' = T_1 = L_1 L_1^T,$$

and

$$T_k' = T_k - R_k R_k^T = T_k - y_{m_{k-1}+1}^2 e_1^{(n_k)} (e_1^{(n_k)})^T = L_k L_k^T, \qquad k = 2, \ldots, q.$$

It is therefore clear that the preliminary computation of the "perturbations"

$$a'_{m_{k-1}+1} = a_{m_{k-1}+1} - y_{m_{k-1}+1}^2$$

reduces the original problem (computing the Cholesky decomposition of T) to q independent instances of the same problem, i.e., the computation of the Cholesky factors of $T_1', \ldots, T_q'$. We now show some preliminary facts about these perturbations that we will use to prove the correctness of our parallel algorithm.

For $k = 1, \ldots, q-1$, let

$$T_{(m_k)} = \begin{pmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & b_3 & & \ddots & \\ & & \ddots & & b_{m_k} \\ & & & b_{m_k} & a_{m_k} \end{pmatrix},$$

and let $L_{(k)}$ denote the Cholesky factor of $T_{(m_k)}$. We write $L_{(k+1)}$ in block notation as follows

$$L_{(k+1)} = \begin{pmatrix} L_{(k)} & \\ R_{(k+1)} & L_{k+1} \end{pmatrix}.$$

Then it easily follows that

$$\begin{aligned} y_{m_k+1}^2 &= \left(e_1^{(n_{k+1})}\right)^T R_{(k+1)} R_{(k+1)}^T e_1^{(n_{k+1})} \\ &= b_{m_k+1}^2 e_{m_k}^T T_{(m_k)}^{-1} e_{m_k}. \end{aligned}$$

In our parallel algorithm we will actually compute

$$x_{m_k} = a'_{m_k} = a_{m_k} - b_{m_k}^2 e_{m_k-1}^T T_{(m_k-1)}^{-1} e_{m_k-1}, \tag{4}$$

and then obtain

$$a'_{m_k+1} = a_{m_k+1} - b_{m_k+1}^2 / x_{m_k},$$

which is consistent with recurrences (3), from which we have $y_{m_k+1}^2 = b_{m_k+1}^2 / x_{m_k}$ and $x_{m_k} = a_{m_k} - y_{m_k}^2$.

LEMMA 5.1. *Let $P_i, i = 1, \ldots, j$, be a sequence of cyclic transformations. Then*

$$P = P_j \cdots P_2 P_1 = \prod_{i=1}^{j} \begin{pmatrix} H_i & 0 \\ h_i^T & 1 \end{pmatrix} = \begin{pmatrix} H & 0 \\ h^T & 1 \end{pmatrix},$$

is a cyclic transformation.

LEMMA 5.2. *Let $P_{(m_k)}$ be a cyclic transformation such that*

$$P_{(m_k)} T_{(m_k)} = \begin{pmatrix} H_k & 0 \\ h_k^T & 1 \end{pmatrix} \begin{pmatrix} T_{(m_k-1)} & b_{m_k} e_{m_{k-1}} \\ b_{m_k} e_{m_{k-1}}^T & a_{m_k} \end{pmatrix} = \begin{pmatrix} \tilde{T}_{(m_k-1)} & * \\ 0 & \tilde{a}_{m_k} \end{pmatrix}. \tag{5}$$

Then we have

$$\tilde{a}_{m_k} = a_{m_k} + b_{m_k} h_k^T e_{m_k-1}^{(m_k-1)} = x_{m_k}. \tag{6}$$

Proof. From the second equality in (5), we have

$$T_{(m_k-1)} h_k = -b_{m_k} e_{m_k-1}^{(m_k-1)}, \quad h_k = -b_{m_k} T_{(m_k-1)}^{-1} e_{m_k-1},$$

so that (4) and (6) are equal. □

In the preprocessing stage of our parallel algorithm we apply a sequence of parallel cyclic transformations to obtain the so called pivot values x_{m_k}, for $k = 1, \ldots, q-1$.

5.2. The algorithm. We assume for simplicity that the order N of the matrix T is such that $N = nq$, with $q = p = 2^r$, and that p is the number of the available processors. We initially distribute the entries of the matrix between the processors, so that each processor stores n consecutive rows. We denote these blocks of rows by

$$B_i = \begin{pmatrix} b_{(i-1)*n+1} & a_{(i-1)*n+1} & b_{(i-1)*n+2} & & \\ \ddots & \ddots & \ddots & & \\ & b_{i*n} & a_{i*n} & b_{i*n+1} \end{pmatrix} \in \mathcal{M}(n, n+2),$$

for $i = 1, \ldots, p$.

The algorithm, that we call *Parallel Cholesky*, consists of three stages.
(i) Diagonalization,
(ii) Bottom-Up and Top-Down Sweeps,
(iii) Factorization.

Stage (i) consists of $O(n)$ parallel steps, that are performed by each processor in an independent way, i.e. no interprocessor communication is required. In stage (ii) the processors perform $O(\log p)$ operations each which do require interprocessor communication. Finally, in stage (iii) each of them perform $O(n)$ operations independently and in parallel. Altogether, the number of parallel steps is $O(n + \log p)$. If $n = \Omega(\log N)$ we have that the overall work done is $O(np)$ and, since this coincides with the sequential complexity of the problem, the algorithm is work optimal.

Stage (i): Diagonalization. For simplicity of notation we denote a given block B by

$$B = \begin{pmatrix} b_1 & a_1 & b_2 & & \\ & b_2 & A & b_n & \\ & & b_n & a_n & b_{n+1} \end{pmatrix} \in \mathcal{M}(n, n+2). \tag{7}$$

Note that A is a tridiagonal matrix of order $n-2$, and that b_2 and b_n in the middle row (column) of B should be actually read as $b_2 e_1^{(n-2)}$ and $b_n e_{n-2}^{(n-2)}$ ($b_2 \left(e_1^{(n-2)}\right)^T$ and $b_n \left(e_{n-2}^{(n-2)}\right)^T$), respectively. Each processor i, $1 < i < p$, performs a forward Gaussian elimination procedure to eliminate b_n in the last row, and then a backward Gaussian elimination procedure to eliminate b_2 in the first row. In matrix notations this amounts to applying a cyclic transformation,

$$B' = PB = \begin{pmatrix} 1 & -b_2 f^T & \\ & I & \\ & -b_n g^T & 1 \end{pmatrix} B, \qquad \begin{aligned} f &= A^{-1} e_1^{(n-2)}, \\ g &= A^{-1} e_{n-2}^{(n-2)}, \end{aligned}$$

so that

$$B' = \begin{pmatrix} b_1 & a_1' & & b_2' & \\ & b_2 & A & b_n & \\ & b_n' & & a_n' & b_{n+1} \end{pmatrix} = \begin{pmatrix} b_1 & v & & y & \\ & b_2 & A & b_n & \\ & y & & w & b_{n+1} \end{pmatrix}, \tag{8}$$

since $y = b'_2 = b'_n$ for symmetry. Processor 1 performs only a forward Gaussian elimination procedure to eliminate b_n in the last row, obtaining

$$B' = \begin{pmatrix} A & b_n & \\ & w & b_{n+1} \end{pmatrix} \in \mathcal{M}(n, n+1),$$

and processor p remains idle. By the end of stage (i), the in-between rows do not further contribute to the search for the pivots x_{m_k}, so that they can be ignored. The original matrix has now been reduced into a $(2p-3) \times (2p-3)$ tridiagonal symmetric matrix. Note that processor 1 contributes one row, while processor i, $i > 1$ contributes two rows, i.e.,

$$T^{(0)} = \begin{pmatrix} w_1^{(0)} & b_2^{(0)} & & & \\ b_2^{(0)} & & \ddots & & \\ & \ddots & \ddots & b_{p-1}^{(0)} & \\ & & b_{p-1}^{(0)} & v_{p-1}^{(0)} & y_{p-1}^{(0)} \\ & & & y_{p-1}^{(0)} & w_{p-1}^{(0)} \end{pmatrix},$$

where $b_{i+1}^{(0)} = b_{in+1}$. Processor $i = 2, \ldots, p-1$ stores the submatrix

$$T_i^{(0)} = \begin{pmatrix} b_i^{(0)} & v_i^{(0)} & y_i^{(0)} & \\ & y_i^{(0)} & w_i^{(0)} & b_{i+1}^{(0)} \end{pmatrix},$$

while processor 1 stores

$$X_1^{(0)} = \begin{pmatrix} w_1^{(0)} & b_2^{(0)} \end{pmatrix} \equiv \begin{pmatrix} x_1 & b_2^{(0)} \end{pmatrix}.$$

Stage (ii): Bottom-Up and Top-Down Sweeps. We apply cyclic transformations to the reduced matrix $T^{(0)}$. We first perform a Bottom-Up sweep as follows:
For $s = 1, \ldots, \log_2 p - 1$, let

$$T_{i,0}^{(s)} = \begin{pmatrix} T_{2i-1}^{(s-1)} \\ T_{2i}^{(s-1)} \end{pmatrix} = \begin{pmatrix} b_{2i-1}^{(s-1)} & v_{2i-1}^{(s-1)} & y_{2i-1}^{(s-1)} & & & \\ & y_{2i-1}^{(s-1)} & w_{2i-1}^{(s-1)} & b_{2i}^{(s-1)} & & \\ & & b_{2i}^{(s-1)} & v_{2i}^{(s-1)} & y_{2i}^{(s-1)} & \\ & & & y_{2i}^{(s-1)} & w_{2i}^{(s-1)} & b_{2i+1}^{(s-1)} \end{pmatrix},$$

for $i = 2, \ldots, p/2^s - 1$. As in Stage (i), we eliminate $y_{2i-1}^{(s-1)}$ in the top row, and $y_{2i}^{(s-1)}$ in the bottom row, by applying a cyclic transformation, i.e.

$$T_{i,1}^{(s)} = P_i^{(s)} T_{i,0}^{(s)} = \begin{pmatrix} b_i^{(s)} & v_i^{(s)} & & & y_i^{(s)} & \\ & y_{2i-1}^{(s-1)} & w_{2i-1}^{(s-1)} & b_{2i}^{(s-1)} & & \\ & & b_{2i}^{(s-1)} & v_{2i}^{(s-1)} & y_{2i}^{(s-1)} & \\ & y_i^{(s)} & & & w_i^{(s)} & b_{i+1}^{(s)} \end{pmatrix}.$$

We then form a matrix $T_i^{(s)}$ using the extreme rows

$$T_i^{(s)} = \begin{pmatrix} b_i^{(s)} & v_i^{(s)} & y_i^{(s)} & \\ & y_i^{(s)} & w_i^{(s)} & b_{i+1}^{(s)} \end{pmatrix}.$$

Similarly, for $i = 1$, let

$$T_{1,0}^{(s)} = \begin{pmatrix} X_1^{(s-1)} \\ T_2^{(s-1)} \end{pmatrix} = \begin{pmatrix} x_{2^{s-1}} & b_2^{(s-1)} & & \\ b_2^{(s-1)} & v_2^{(s-1)} & y_2^{(s-1)} & \\ & y_2^{(s-1)} & w_2^{(s-1)} & b_3^{(s-1)} \end{pmatrix}.$$

Then we eliminate $y_2^{(s-1)}$ in the bottom row by applying the cyclic transformation

$$T_{1,1}^{(s)} = P_1^{(s)} T_{1,0}^{(s)} = \begin{pmatrix} x_{2^{s-1}} & b_2^{(s-1)} & & \\ b_2^{(s-1)} & v_2^{(s-1)} & y_2^{(s-1)} & \\ & & w_1^{(s)} & b_2^{(s)} \end{pmatrix}, \tag{9}$$

and we form the matrix $X_1^{(s)}$ with the new bottom row

$$X_1^{(s)} = \begin{pmatrix} w_1^{(s)} & b_2^{(s)} \end{pmatrix} \equiv \begin{pmatrix} x_{2^s} & b_2^{(s)} \end{pmatrix}.$$

We then proceed with a Top-Down sweep as follows.
For $s = \log_2 p - 2, \log_2 p - 3, \ldots, 0$ let,

$$T_{i,0}^{(s)} = \begin{pmatrix} X_j^{(s+l)} \\ T_i^{(s)} \end{pmatrix} = \begin{pmatrix} x_{j2^{s+l}} & b_i^{(s)} & & \\ b_i^{(s)} & v_i^{(s)} & y_i^{(s)} & \\ & y_i^{(s)} & w_i^{(s)} & b_{i+1}^{(s)} \end{pmatrix}, \qquad \begin{matrix} i = 3, 5, \ldots, p/2^s - 1, \\ i = 2^l j + 1, j \text{ is odd.} \end{matrix}$$

We then eliminate $y_i^{(s)}$ in the bottom row by applying a cyclic transformation as in (9), and we form the matrix $X_i^{(s)} = \begin{pmatrix} x_{i2^s} & b_{i+1}^{(s)} \end{pmatrix}$.

THEOREM 5.3. *Stages (i) and (ii) of the Parallel Cholesky algorithm correctly compute the pivots* $x_k, k = 1, \ldots, p-1$.

Proof. With respect to a given processor $k, 1 \le k \le (p-1)$, the sequence of parallel transformations applied in stages (i) and (ii) of the algorithm consist of a sequence of cyclic transformations applied to the submatrix $T_{(m_k)}$, and so, from corollary 5.1, to some given cyclic transformation. Since this annihilates the off diagonal element b_{m_k}, the proof follows from Lemma 5.2. □

Stage (iii): Factorization. The parallel factorization of the independent blocks is straightforward. Processor 1 computes the Cholesky decomposition of its original block T_1, while processors 2 through q modify their blocks according to the rule

$$a'_{in+1} = a_{in+1} - b_{in+1}^2 / x_{i-1},$$

before computing their decompositions.

Figure 5.1 contains a flowchart of the algorithm.

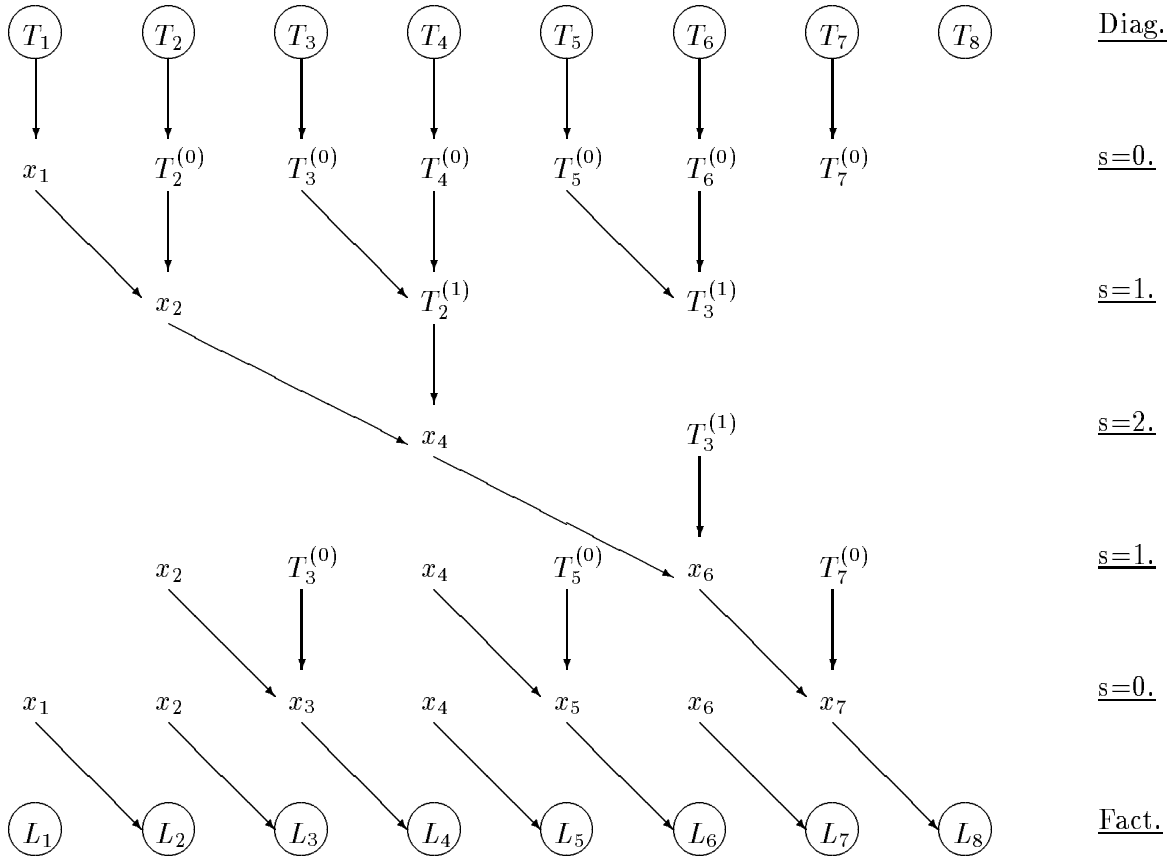

FIG. 5.1. *A flowchart for $p = 2^3$ processors.*

6. Parallel computational cost. In this section we study the computational cost of the Parallel Cholesky algorithm of Section 5.2.

Stage (i). To determine the cost of this stage, we give the details of the forward and backward Gaussian Elimination procedures. We denote the blocks in each processor as in (7), and the computed transformations as in (8). Since we compute the revised decomposition introduced in Section 4, in what follows we actually use the squares c_i of the off diagonal elements b_i of the matrix T.

- Forward Gaussian elimination:
  1. Set $z = c_2$ and $w = a_2$.
  2. For $i = 3, \ldots, n$ set

$$\begin{aligned} t &= c_i/w, \\ z &= z * t/w, \end{aligned}$$

$$w = a_i - t.$$

- Backward Gauss elimination:
  1. Set $v = a_{n-1}$.
  2. For $i = n-1, \ldots, 2$ set

$$\begin{aligned} t &= c_i/v \\ v &= a_{i-1} - t \end{aligned}$$

The flop count for Stage (i) is therefore $\sim 6n$, This is a pretty accurate measure of the number of parallel steps performed by the p processors (no communication is involved).

Stage (ii). Let

$$T = \begin{pmatrix} b_1 & v_1 & y_1 & & & \\ & y_1 & w_1 & b_2 & & \\ & & b_2 & v_2 & y_2 & \\ & & & y_2 & w_2 & b_3 \end{pmatrix} \Rightarrow \begin{pmatrix} b_1 & v & & & y & \\ & y_1 & w_1 & b_2 & & \\ & & b_2 & v_2 & y_2 & \\ & y & & & w & b_3 \end{pmatrix}$$

denote a typical transformation in the Bottom-Up sweep. Again, we consider the squares of the off diagonal elements, $c_2 = b_2^2$ and $z_i = y_i^2$, $i = 1, 2$, and compute $z = y^2$. Therefore, we perform the following calculation

$$\begin{aligned} \alpha &= w_1 * v_2, \\ \beta &= c_2/\alpha, \\ \gamma &= 1 - \beta, \\ t_1 &= z_1/(w_1 * \gamma), \\ t_2 &= z_2/(v_2 * \gamma), \\ v &= v_1 - t_1, \\ w &= w_2 - t_2, \\ z &= \beta * t_1 * t_2. \end{aligned}$$

This calculation takes 11 parallel steps. Similarly, let

$$T = \begin{pmatrix} x_1 & b_2 & & \\ b_2 & v_2 & y_2 & \\ & y_2 & w_2 & b_3 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 & b_2 & & \\ b_2 & v_2 & y_2 & \\ & & x_2 & b_3 \end{pmatrix}$$

denote a typical transformation in the Top-Down sweep. Then we compute $t = v_2 - c_2/x_1$ and $x_2 = w_2 - z_2/t$ in 4 parallel steps.

Stage (iii). The number of parallel steps is $\sim 2n$, as was shown in Section 4.

The total number T_p of parallel steps is therefore:

$$T_p \sim 6n + 11r + 4r + 2n = 8n + 15r,$$

where $r = \log_2 p$. Assuming that $r \ll n$ we conclude that the cost of the parallel algorithm is governed by the factor $8n$. Hence, the parallel algorithm requires about 4 times the number of flops of the sequential algorithm (see Section 4). The theoretical speedup is thus $p/4$. However, on vector, pipelined, and super-scalar machines, the flops count determines the true performance of an algorithm only to within a constant factor. Actually, an algorithm with a worse flops count might perform better in the case it can be vectorized. We show now that our parallel algorithm can be vectorized.

Vectorization: Let $N = p * n$ where $p = 2^r$ is the number of "physical" processors. Suppose that each of such processors has the availability of one or more vector units. One possibility to exploit the additional power given by the latter rests on employing some "parallel slackness". In other words, we assume that the number of available processors is larger than p and let each physical processor simulate many such "logical" processors. More precisely, let $n = P * m$, where $P = 2^t$, so that $N = q * m$, with $q = p * P = 2^{r+t}$. We let each physical processor performs the tasks of P corresponding logical processors. The number of flops in stages (i) and (iii) is still $\sim 8n$. The number of flops in stage (ii) increases to $\sim 15(r+P)$, which is still negligible for $(r+P) \ll n$. However, the main stages of the algorithm, namely stages (i) and (iii), can now be vectorized, with each processor working on vectors of length $P = 2^t$. We provide an example in this direction with the implementation of the algorithm on the CM5. The results obtained are reported in the next section.

7. Numerical examples. In this section we present some experimental results obtained on a CM5 parallel supercomputer with $p = 32$ nodes. Each node is in turn composed of 4 vector units, controlled by a SPARC microprocessor, and 32 Mbytes of memory. The running time and speedup for the largest problems we could experiment on are shown in Table 2. We have computed the Cholesky factorization of several classes of tridiagonal matrices, including (a) the matrix

$$T = \begin{pmatrix} 2 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & 1 & & \ddots & & \\ & & \ddots & & 1 & \\ & & & 1 & 2 & \end{pmatrix},$$

(b) matrices obtained from T above by varying the diagonal elements, and (c) random tridiagonal matrices. The order of the test matrices is $N = p * n = q * m$, where $p = 32$ is the number of the "physical" processors actually available, and q is the number of "logical" processors (cf. Section 6). Table 2 gives the running times for each of the following stages of the algorithm.

1. D - Logical Diagonalization.
2. I - Bottom-Up and Top-Down stages performed by the logical processors within any physical processor.
3. E - Bottom-Up and Top-Down stages performed by the physical processors.
4. C - Logical Factorization.

5. S - Sequential algorithm.

N	$3 * 2^{24}$	2^{25}	2^{24}
n	$3 * 2^{19}$	2^{20}	2^{19}
m	$3 * 2^{8}$	2^{9}	2^{8}
D	1.444	0.963	0.480
I	0.033	0.033	0.035
E	0.005	0.008	0.009
C	0.562	0.372	0.186
total	2.045	1.376	0.710
S	2704.121	1791.729	880.850
speed-up	1322	1302	1240

TABLE 2
Computational examples on the CM5, $q = 2^{16}$.

The results shown in Table 2 are those obtained for the matrix T above. Very similar results have been obtained with the other test cases.

8. Error Analysis. We first present a simple a-posteriori estimate, and then proceed to give an a priori analysis.

8.1. A-posteriori error analysis. Consider stage (iii) of the parallel algorithm, which is the actual transformation applied to the matrix. As in (7), let B denote the block assigned to a processor, and let x_0 be the pivot computed by the previous processor. We compute the following:

$$\begin{aligned} z_i &= c_i/x_{i-1}, \quad i = 1, , \ldots, , n, \\ x_i &= a_i - z_i, \quad i = 1, , \ldots, , n. \end{aligned}$$

Due to rounding errors, we get

$$\begin{aligned} \hat{z}_i = \hat{c}_i/\hat{x}_{i-1}, \quad \hat{c}_i = c_i(1+\epsilon_i), \quad |\epsilon_i| \le \theta, \quad i = 1, , \ldots, , n, \\ \hat{x}_i = \hat{a}_i - \hat{z}_i, \quad \hat{a}_i = a_i(1+\eta_i), \quad |\eta_i| \le \theta, \quad i = 1, , \ldots, , n, \end{aligned}$$

where $\hat{x}_0$ is the computed pivot. In this analysis there is a small discrepancy because $\hat{x}_n$ is not the same as the pivot transmitted to the next processor. To fix this problem, we define $\hat{x}'_0$ as this new kind of pivot. Then, in the first step above,

$$\hat{z}_1 = (c_1/\hat{x}'_0)(1+\epsilon_1)(\hat{x}'_0/\hat{x}_0) = \hat{c}_1/\hat{x}'_0,$$

where

$$\hat{c}_1 = (1+\epsilon'_1)c_1, \qquad (1+\epsilon'_1) = (1+\epsilon_1)(\hat{x}'_0/\hat{x}_0).$$

The perturbation in the first off diagonal element is further influenced by the factor $\hat{x}'_0/\hat{x}_0$.

If we assume that $|(\hat{x}'_0 - \hat{x}_0)/\hat{x}_0| = O(\theta)$, we can see that the algorithm is pointwise stable. We found that practically, even for very ill conditioned matrices, this factor is

relatively small (see Table 3). Moreover, for some applications like eigenvalue computation, we can sometimes allow for more than pointwise perturbations in the off diagonal elements without loosing accuracy in the final results.

In Table 3 we compare the number of correct computed digits with the estimate given by the a priori error analysis. We consider three different tests: Test 1 - for random diagonally dominant matrices. Test 2 - for the tridiagonal matrix (1,2,1). Test 3 - for random tridiagonal matrices. We have added an appropriate shift to the diagonal elements to monitor the size of smallest eigenvalue of the matrix, which governs the number of a priori correct digits (enclosed in parenthesis). Each Test column gives the number of correct digits in the computed factorization as produced by the a posteriori error bound. The '-' stands for no correct digits at all.

Shift	Test 1	Test 2	Test 3
$10^{-4}(8)$	14	15	16
$10^{-8}(-)$	12	14	16
$10^{-12}(-)$	12	14	8
$10^{-14}(-)$	12	14	6

TABLE 3
A posteriori as compared to a priori results, $N = 2^{23}, m = 2^8$.

Finally note that the a posteriori bound requires no additional cpu time or memory space, so that it can be used to tailor the precision of the computed results with no additional cost. We next perform an a priori error analysis and show that the parallel algorithm is pointwise stable provided some extra precision is used.

8.2. A-priori error analysis. Let T be a tridiagonal symmetric positive definite matrix of order $N = pn$, where p is the number of processors available, each processor being identified by the label i_p, for $i_p = 1, \ldots, p$. We denote by λ_i, $i = 1, \ldots, N$, the eigenvalues of the tridiagonal matrix, so that

$$\lambda_1 > \lambda_2 > \cdots \lambda_{N-1} > \lambda_N > 0.$$

We further denote by $\hat{\theta}$ the relative precision to which we know the matrix coefficients. We then assume that pointwise $\hat{\theta}$-perturbations in the matrix coefficients may lead up to $O(\theta')$ relative perturbations in its eigenvalues, where $\hat{\theta} \leq \theta' \ll 1$. Although theoretically we can only say that $\theta' = (\lambda_1/\lambda_N)\hat{\theta}$ (see Barlow and Demmel [10] and Demmel and Kahan [12] for some exceptions to this rule), we often do get $\theta' = O(\hat{\theta})$ in practical applications. However, to be on the safe side, we will assume in the following that the preprocessing stage is performed with $\theta = \overline{\theta}(\lambda_N/\lambda_1)$ precision, where $\overline{\theta} = \hat{\theta}(\lambda_N/\lambda_1)$. The actual precision should be determined by the application according to some previous knowledge or based on the a posteriori bounds. We note that increasing the precision is a scalable operation that does not require extra memory space.

THEOREM 8.1. *Let $i_p = i2^s$, where $i = 2^l j + 1$. Then, the pivots x_{i_p} and x'_{i_p} are computed from the same tridiagonal symmetric matrix, namely,*

$$\mathcal{T}_{i_p} = \begin{pmatrix} X_j^{(s+l)} & & & & & & \\ & T_{2i-1}^{(s-1)} & & & & & \\ & & T_{4i-1}^{(s-2)} & & & & \\ & & & \ddots & & & \\ & & & & T_{2^s i-1}^{(0)} & & \\ & & & & & T_{2^s i} & \end{pmatrix},$$

of order $n_{i_p} = n + 2s + 1$ when j is odd, and

$$\mathcal{T}_{i_p} = \begin{pmatrix} X_1^{(s-1)} & & & & & \\ & T_{4i-1}^{(s-2)} & & & & \\ & & \ddots & & & \\ & & & T_{2^s i-1}^{(0)} & & \\ & & & & T_{2^s i} & \end{pmatrix},$$

of order $n_{i_p} = n + 2s - 1$ when $i = 1$.

Proof. The computation of x_{i_p} in the case $i \geq 3$, proceeds by diagonalizing T_{i_p}, and then by merging,

$$T_i^{(s)} = (\cdots((T_{i2^s}^{(0)} \bigcup T_{i2^s-1}^{(0)}) \bigcup T_{i2^{s-1}-1}^{(1)}) \cdots) \bigcup T_{2i-1}^{(s-1)},$$

in the Bottom-Up sweep. Finally, we merge $X_j^{(s+l)}$ with $T_i^{(s)}$ in the Top-Down sweep. A similar procedure is performed for the case $i = 1$. The computation of x'_{i_p} proceeds by a sequence of Top-Down transformations, and a Cholesky factorization of the respective modified matrix T'_{i_p}. □

THEOREM 8.2. *The computed pivots $\hat{x}_{i_p}$ and $\hat{x}'_{i_p}$ correspond to different $O(\overline{\theta})$ relative perturbations of the computed matrix $\hat{\mathcal{T}}_{i_p}$, and $O(\overline{\theta}\lambda_1)$ absolute perturbations of its diagonal elements. Hence,*

$$|\hat{x}_{i_p}/\hat{x}'_{i_p}| = (1 + \eta) \qquad |\eta| = O((n + r)\hat{\theta}) = O(\hat{\theta}).$$

Most likely, $|\eta| = O(\overline{\theta})$ so that we can perform the computation using $O(\overline{\theta})$ precision.

For technical reasons, we now state and prove some lemmas and postpone the proof of Theorem 8.2 to the end of this section.

We denote the transformations in stage (i) by

$$\begin{pmatrix} v & z & \\ z & A & c \\ & c & w \end{pmatrix} \Rightarrow \begin{pmatrix} v - \Delta^{(0)} & & u \\ z & A & c \\ u & & w - \Sigma^{(0)} \end{pmatrix},$$

where

$$\begin{aligned} \Sigma^{(0)} &= \sigma_0, & \sigma_0 &= c(e_{n-2}^T A^{-1} e_{n-2}), \\ \Delta^{(0)} &= \delta_0, & \delta_0 &= z(e_1^T A^{-1} e_1), \\ u &= cz(e_{n-2}^T A^{-1} e_1), \end{aligned}$$

where we have replaced the off diagonal elements with their squares to simplify the representation. We denote the middle transformations in the Bottom-Up sweep by

$$\begin{pmatrix} v_1 - \Delta^{(s-1)} & u_1 & & \\ u_1 & w_1 - \Sigma_1^{(s-1)} & c & \\ & c & v_2 - \Delta_2^{(s-1)} & u_2 \\ & & u_2 & w_2 - \Sigma^{(s-1)} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} v_1 - \Delta^{(s)} & & & u \\ u_1 & w_1 - \Sigma_1^{(s-1)} & c & \\ & c & v_2 - \Delta_2^{(s-1)} & u_2 \\ u & & & w_2 - \Sigma^{(s)} \end{pmatrix}$$

for $s = 1, \ldots, r-1$, where

$$\Delta^{(s)} = \Delta^{(s-1)} + \delta_s, \qquad \Sigma^{(s)} = \Sigma^{(s-1)} + \sigma_s,$$

and

$$\delta_s = \tfrac{u_1}{d_1 \gamma}, \qquad \sigma_s = \tfrac{u_2}{d_2 \gamma}, \qquad u = \beta \delta_s \sigma_s, \tag{10}$$

with

$$\gamma = 1 - \beta, \quad \beta = \tfrac{c}{d_1 d_2}, \qquad d_1 = w_1 - \Sigma_1^{(s-1)}, \quad d_2 = v_2 - \Delta_2^{(s-1)}. \tag{11}$$

Here we may compute $d_1(d_2)$ in two different ways. a) Accumulate the corresponding partial shifts and then deduce their sum from the corresponding element $w_1(v_2)$. b) Update these elements with each newly created partial shift. We note that method a) will probably compute $d_1(d_2)$ more accurately than method b), especially when these partial shifts tend to increase monotonically. We therefore choose it in the following analysis, although the same bounds could be achieved using method b) as well.

Finally, we denote the transformations in the Top-Down sweep (as well as the first transformations in the Bottom-Up sweep) by

$$\begin{pmatrix} x_0 & c & \\ c & v - \Delta^{(s)} & u \\ & u & w - \Sigma^{(s)} \end{pmatrix} \Rightarrow \begin{pmatrix} x_0 & c & \\ & d & u \\ & & x_1 \end{pmatrix},$$

for $s = r-1, \ldots, 0$, where

$$x_1 = w - \Sigma^{(s+1)}, \qquad \Sigma^{(s+1)} = \Sigma^{(s)} + \sigma_{s+1}, \qquad \sigma_{s+1} = \tfrac{u}{d},$$

with

$$d = v - \Delta^{(s+1)}, \qquad \Delta^{(s+1)} = \Delta^{(s)} + \delta_{s+1}, \qquad \delta_{s+1} = \tfrac{c}{x_0}. \tag{12}$$

LEMMA 8.3. *The computed value of d_1, i.e.,*

$$\hat{d}_1 = fl(w_1 - \hat{\Sigma}^{(s)}) = fl(w_1 - fl(\textstyle\sum_{i=0}^{s} \hat{\sigma}_i)), \qquad \hat{\sigma}_i = fl(\sigma_i),$$

in step $s+1$ of the Bottom-Up sweep is such that

$$\hat{d}_1 = \hat{w}_1 - \overline{\Sigma}^{(s)}, \qquad \overline{\Sigma}^{(s)} = \textstyle\sum_{i=0}^{s} \overline{\sigma}_i,$$

where

$$\begin{array}{rcll} \hat{w}_1 & = & w_1(1+\epsilon_{\hat{w}_1}), & |\epsilon_{\hat{w}_1}| \le \theta, \\ \overline{\sigma}_i & = & \hat{\sigma}_i(1+\epsilon_{\sigma_i}), \quad i = 0, \ldots, s, & |\epsilon_{\sigma_i}| = O(\theta). \end{array}$$

Proof. We have that

$$\hat{d}_1 = (w_1 - \hat{\Sigma}^{(s)})(1+\epsilon_{\hat{d}_1}) = \hat{w}_1 - \overline{\Sigma}^{(s)}, \quad \hat{w}_1 = w_1(1+\epsilon_{\hat{w}_1}), \quad |\epsilon_{\hat{w}_1}| \le |\epsilon_{\hat{d}_1}| \le \theta,$$

and $\overline{\Sigma}^{(s)} = \hat{\Sigma}^{(s)}$. Then we obtain

$$\overline{\Sigma}^{(s)} = \sum_{i=0}^{s} \hat{\sigma}_i \prod_{j=i-1}^{s-1} (1+\epsilon_j) = \sum_{i=0}^{s} \overline{\sigma}_i,$$

where $\epsilon_{-1} = 0$ and $|\epsilon_j| \le \theta$ for $j = 0, \ldots, (s-1)$. Hence,

$$\overline{\sigma}_i = \hat{\sigma}_i \prod_{j=i-1}^{s-1} (1+\epsilon_j) = \hat{\sigma}_i(1+\epsilon_{\sigma_i}),$$

with $|\epsilon_{\sigma_i}| = O(\theta)$. □

COROLLARY 8.4. *Under the hypotheses of Lemma 8.3 we obtain similar results for d_2, that is,*

$$\hat{d}_2 = \hat{v}_2 - \overline{\Delta}^{(s)}, \quad \hat{v}_2 = v_2(1+\epsilon_{\hat{v}_2}), \quad |\epsilon_{\hat{v}_2}| \le \theta,$$

where $\overline{\Delta}^{(s)} = \hat{\Delta}^{(s)}$, and

$$\overline{\Delta}^{(s)} = \textstyle\sum_{i=0}^{s} \overline{\delta}_i, \qquad \overline{\delta}_i = \hat{\delta}_i(1+\epsilon_{\delta_i}), \qquad |\epsilon_{\delta_i}| = O(\theta).$$

Similarly, in step $s-1$ of the Top-Down sweep (or in the first transformations of the Bottom-Up sweep), we obtain

$$\begin{array}{rcll} \hat{d} = \hat{v} - \overline{\Delta}^{(s)}, & \hat{v} = v(1+\epsilon_{\hat{v}}), & |\epsilon_{\hat{v}}| \le \theta, \\ \hat{x}_1 = \hat{w} - \overline{\Sigma}^{(s)}, & \hat{w} = w(1+\epsilon_{\hat{w}}), & |\epsilon_{\hat{w}}| \le \theta, \end{array}$$

where $\overline{\Delta}^{(s)} = \hat{\Delta}^{(s)}$, *and* $\overline{\Sigma}^{(s)} = \hat{\Sigma}^{(s)}$, *as above.*

We now consider stage (i).

LEMMA 8.5. *The computed transformation in stage (i) is affected by perturbations of the order of* $\overline{\theta}$, *with the following structure:*

$$\begin{pmatrix} v & \overline{z} & \\ \overline{z} & \overline{A} & \overline{c} \\ & \overline{c} & w \end{pmatrix} \Rightarrow \begin{pmatrix} v - \overline{\Delta}^{(0)} & & \hat{u} \\ \overline{z} & \overline{A} & \overline{c} \\ \hat{u} & & w - \overline{\Sigma}^{(0)} \end{pmatrix}.$$

Proof. The forward elimination procedure can be viewed as a Cholesky factorization of A (see Section 4), and therefore produces a matrix $\overline{A}$ which differs from A for an $O(\theta)$ perturbation. Hence,

$$\hat{\sigma}_0 = c(e_{n-2}^T \overline{A}^{-1} e_{n-2})(1+\epsilon_1), \qquad |\epsilon_1| \leq \theta.$$

Then,

$$\overline{\sigma}_0 = \overline{c}(e_{n-2}^T \overline{A}^{-1} e_{n-2}), \qquad \overline{c} = (1+\epsilon_{\sigma_0})(1+\epsilon_1)c = (1+\epsilon_c)c,$$

and $|\epsilon_c| = O(\theta)$. Similarly, the backward elimination produces another θ-perturbation of A, which we denote by $\hat{A}$. Hence, we have

$$\hat{\delta}_0 = z(e_1^T \hat{A}^{-1} e_1)(1+\epsilon_2) = z(e_1^T \overline{A}^{-1} e_1)(1+\epsilon_z'),$$

where

$$1 + \epsilon_z' = \frac{(e_1^T \hat{A}^{-1} e_1)}{(e_1^T \overline{A}^{-1} e_1)}(1+\epsilon_2), \qquad |\epsilon_2| \leq \theta. \tag{13}$$

We now analyze expression (13).

Let

$$\gamma_1 > \gamma_2 > \cdots \gamma_{n-3} > \gamma_{n-2} > 0,$$

be the eigenvalues of A, and $\overline{\gamma}_i, \hat{\gamma}_i$, $i = 1, \ldots, n-2$, those of $\overline{A}$ and $\hat{A}$, respectively. Let A' denote the leading submatrix of A of order $n-3$, and let its eigenvalues be

$$\gamma_1' > \gamma_2' > \cdots \gamma_{n-4}' > \gamma_{n-3}' > 0.$$

Let $\overline{A}'$ and $\hat{A}$ denote the corresponding submatrices, whose eigenvalues are denoted by $\overline{\gamma}_i'$ and $\hat{\gamma}_i'$, $i = 1, \ldots, n-3$, respectively. Then

$$\frac{(e_1^T \hat{A}^{-1} e_1)}{(e_1^T \overline{A}^{-1} e_1)} = \prod_{i=1}^{n-3} \frac{\hat{\gamma}_i'}{\overline{\gamma}_i'} \prod_{i=1}^{n-2} \frac{\overline{\gamma}_i}{\hat{\gamma}_i} = \prod_{i=1}^{n-3} \frac{(1+\hat{\epsilon}_i')}{(1+\overline{\epsilon}_i')} \prod_{i=1}^{n-2} \frac{(1+\overline{\epsilon}_i)}{(1+\hat{\epsilon}_i)} = (1+\eta),$$

with $|\eta| = O(\overline{\theta})$. Though very general, this bound captures the possible behavior of the error since we can actually exhibit a matrix for which this bound is achieved. More special upper bounds can also be derived, see for example Sun [25]. However, since A is a relatively small submatrix of the original matrix, the estimate $\gamma_1 = O(\gamma_{n-2})$ is likely to be more accurate, and in this case $|\eta| = O(\theta)$ so that $\theta = \overline{\theta}$ will suffice.

We then obtain

$$\overline{\delta}_0 = \overline{z}(e_1^T \overline{A}^{-1} e_1), \qquad \overline{z} = (1+\eta)(1+\epsilon_2)(1+\epsilon_{\delta_0})z = (1+\epsilon_z)z,$$

with $|\epsilon_z| = O(\overline{\theta})$. Finally, we have that

$$\hat{u} = zc(e_{n-2}^T \overline{A}^{-1} e_1) \prod_{i=3}^{n} (1+\epsilon_{2i-1})(1+\epsilon_{2i}), \qquad |\epsilon_j| \leq \theta, \quad j = 5, \ldots, 2n,$$

and from $\overline{u} = \overline{zc}(e_{n-2}^T \overline{A}^{-1} e_1)$, we get

$$\hat{u} = (1+\epsilon_u)\overline{u} = \frac{\prod_{i=3}^{n}(1+\epsilon_{2i-1})(1+\epsilon_{2i})}{(1+\epsilon_c)(1+\epsilon_z)}\overline{u}, \qquad |\epsilon_u| = O(\overline{\theta}),$$

which completes the proof. We further note that for $i_p = 1$, the corresponding analysis is much simpler, and the computed pivot satisfies:

$$\hat{x} = \overline{w} - \overline{\Sigma}^{(0)}, \qquad \overline{w} = w(1+\epsilon_w), \quad |\epsilon_w| \leq \theta.$$

□

We next analyze the Bottom-Up stage.

LEMMA 8.6. *The computed transformations in each of the Bottom-Up steps are affected by perturbations of the order of θ, except for the first step where the perturbations are of the order of $\overline{\theta}$. The structure of the perturbations is*

$$\begin{pmatrix} v_1 - \overline{\Delta}^{(s-1)} & \overline{u}_1 & & & \\ \overline{u}_1 & \overline{w}_1 - \overline{\Sigma}_1^{(s-1)} & \overline{c} & & \\ & \overline{c} & \overline{v}_2 - \overline{\Delta}_2^{(s-1)} & \overline{u}_2 & \\ & & \overline{u}_2 & w_2 - \overline{\Sigma}^{(s-1)} & \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} v_1 - \overline{\Delta}^{(s)} & & & & \hat{u} \\ \overline{u}_1 & \overline{w}_1 - \overline{\Sigma}_1^{(s-1)} & \overline{c} & & \\ & \overline{c} & \overline{v}_2 - \overline{\Delta}_2^{(s-1)} & \overline{u}_2 & \\ \hat{u} & & & w_2 - \overline{\Sigma}^{(s)} & \end{pmatrix}$$

for $s = 1, \ldots, r-1$.

Proof. From (10) we have that

$$\hat{\sigma}_s = \frac{\hat{u}_2(1+\epsilon_5)}{\hat{d}_2\hat{\gamma}(1+\epsilon_4)}, \qquad \hat{\gamma} = (1-\hat{\beta})(1+\epsilon_3), \qquad \hat{\beta} = \frac{c(1+\epsilon_2)}{\hat{d}_1\hat{d}_2(1+\epsilon_1)},$$

with $|\epsilon_i| \leq \theta,\ i = 1, \ldots, 5$. Hence,

$$\overline{\sigma}_s = (1+\epsilon_{\sigma_s})\hat{\sigma}_s = \frac{(1+\epsilon_{\sigma_s})(1+\epsilon_{u_2})(1+\epsilon_5)}{(1+\epsilon_4)(1+\epsilon_3)}\frac{\overline{u}_2}{\hat{d}_2\overline{\gamma}} \equiv \frac{\overline{u}_2}{\overline{d}_2\overline{\gamma}},$$

where, $\overline{\beta} \equiv \hat{\beta}$ and $\overline{\gamma} \equiv 1 - \overline{\beta}$. Using Lemma 8.3 and Corollary 8.4, we get

$$\overline{d}_2 = (1+\epsilon_{d_2})\hat{d}_2 = (\hat{v}_2 - \overline{\Delta}_2^{(s-1)})(1+\epsilon_{d_2}) = \overline{v}_2 - \overline{\Delta}_2^{(s-1)}.$$

We observe that the perturbation in the original element v_2 of T is now determined by

$$\overline{v}_2 = v_2(1+\epsilon_{\hat{v}_2})(1+\epsilon_{\overline{v}_2}) = v_2(1+\epsilon_{v_2}), \qquad |\epsilon_{v_2}| = O(|\epsilon_{u_2}|),$$

since $|\epsilon_{\hat{v}_2}| \leq \theta$. Similarly,

$$\overline{\delta}_s = (1+\epsilon_{\delta_s})\hat{\delta}_s = \frac{(1+\epsilon_{\delta_s})(1+\epsilon_{u_1})(1+\epsilon_6)}{(1+\epsilon_7)(1+\epsilon_3)}\frac{\overline{u}_1}{\hat{d}_1\overline{\gamma}} \equiv \frac{\overline{u}_1}{\overline{d}_1\overline{\gamma}},$$

where $|\epsilon_6|, |\epsilon_7| \leq \theta$, and

$$\overline{d}_1 = (1+\epsilon_{d_1})\hat{d}_1 = (\hat{w}_1 - \overline{\Sigma}_1^{(s-1)})(1+\epsilon_{d_1}) = \overline{w}_1 - \overline{\Sigma}_1^{(s-1)}.$$

Here, the perturbation in w_1 is determined by

$$\overline{w}_1 = w_1(1+\epsilon_{\hat{w}_1})(1+\epsilon_{\overline{w}_1}) = w_1(1+\epsilon_{w_1}), \qquad |\epsilon_{w_1}| = O(|\epsilon_{u_1}|).$$

Finally, from (11), we get

$$\overline{\beta} = \frac{c(1+\epsilon_2)(1+\epsilon_{d_1})(1+\epsilon_{d_2})}{\overline{d}_1\overline{d}_2(1+\epsilon_1)} \equiv \frac{\overline{c}}{\overline{d}_1\overline{d}_2},$$

so that

$$\overline{c} = (1+\epsilon_c)c, \qquad |\epsilon_c| = O(|\epsilon_{u_1}| + |\epsilon_{u_2}|),$$

and from

$$\hat{u} = \hat{\beta}\hat{\delta}_s(1+\epsilon_8)\hat{\sigma}_s(1+\epsilon_9) = \frac{(1+\epsilon_8)(1+\epsilon_9)}{(1+\epsilon_{\sigma_s})(1+\epsilon_{\delta_s})}\overline{\beta}\overline{\delta}_s\overline{\sigma}_s \equiv (1+\epsilon_u)\overline{u},$$

with $|\epsilon_8|, |\epsilon_9| \leq \theta$, we get $|\epsilon_u| = O(\theta)$. Therefore,

$$|\epsilon_{v_2}|, |\epsilon_{w_1}| = O(\theta), \qquad s = 2, \ldots, (r-1),$$

except, possibly, for the first step, where $|\epsilon_{v_2}|, |\epsilon_{w_1}| = O(\overline{\theta})$. □

We conclude with the analysis of the Top Down stage.

LEMMA 8.7. *The computed transformations in each Top-Down step are affected by perturbations of the order of θ, except for the last step where they are affected by perturbations of the order of $\overline{\theta}$. The structure of the perturbation is*

$$\begin{pmatrix} \overline{x}_0 & \overline{c} & \\ \overline{c} & \overline{v} - \overline{\Delta}^{(s-1)} & \overline{u} \\ & \overline{u} & \overline{w} - \overline{\Sigma}^{(s-1)} \end{pmatrix} \Rightarrow \begin{pmatrix} \overline{x}_0 & \overline{c} & \\ & \overline{d} & \overline{u} \\ & & \overline{x}_1 \end{pmatrix}$$

for $s = (r-1), \ldots, 1$. A similar statement holds for the first transformation in the Bottom-Up sweep.

Proof. We have from (12),

$$\hat{\delta}^{(s)} = (c/\hat{x}_0)(1+\epsilon_1), \qquad \overline{\delta}_s = \hat{\delta}_s(1+\epsilon_{\delta_s}) = \overline{c}/\overline{x}_0,$$

where

$$\overline{x}_0 = \hat{x}_0, \qquad \overline{c} = c(1+\epsilon_c), \quad |\epsilon_c| = O(\theta).$$

Hence, using Corollary 8.4,

$$\hat{d} = \hat{v} - \overline{\Delta}^{(s)}, \qquad \overline{\Delta}^{(s)} = \overline{\Delta}^{(s-1)} + \overline{\delta}_s,$$

and from,

$$\overline{\sigma}_s = (1+\epsilon_{\sigma_s})\hat{\sigma}_s = (1+\epsilon_{\sigma_s})(1+\epsilon_u)(1+\epsilon_2)\frac{\overline{u}}{\hat{d}} \equiv \frac{\overline{u}}{\overline{d}},$$

with $|\epsilon_2| \leq \theta$, we get

$$\overline{d} = (\hat{v} - \overline{\Delta}^{(s)})(1+\epsilon_d) = \overline{v} - \overline{\Delta}^{(s)}.$$

Here, the perturbation in v is given by

$$\overline{v} = v(1+\epsilon_{\hat{v}})(1+\epsilon_{\overline{v}}) = v(1+\epsilon_v), \qquad |\epsilon_v| = O(|\epsilon_u|),$$

and,

$$|\epsilon_v| = O(\theta), \qquad s = r-1, \ldots, 1,$$

except, possibly, for step $s = 1$, where $|\epsilon_v| = O(\overline{\theta})$. Finally, the perturbation in w is given by

$$\overline{x}_1 \equiv \hat{x}_1 = \hat{w} - \overline{\Sigma}^{(s)} \equiv \overline{w} - \overline{\Sigma}^{(s)}, \qquad \overline{w} = w(1+\epsilon_w),$$

with $|\epsilon_w| \leq \theta$. □

COROLLARY 8.8. *The computed pivots $\hat{x}_{i_p}$ and $\hat{x}'_{i_p}$ are affected by perturbations of the order of $\overline{\theta}$, so that*

$$|\hat{x}_{i_p}/\hat{x}'_{i_p}| = (1+\eta), \qquad |\eta| = O(N\hat{\theta}).$$

Proof. We note that $\mathcal{T}_{i_p}$ is derived from the original matrix by a sequence of parallel transformations of the type described in Lemmas 8.5, 8.6, and 8.7. Then, both x_{i_p} and x'_{i_p} are derived from this matrix by further, though different, transformations of this same type. The upper bound is determined as in the proof of Lemma 8.5. □

Note that the bound of Corollary 8.8 is pessimistic. Most likely, the computed pivots are affected by perturbations of the order of θ only, so that $|\eta| = O(N\overline{\theta})$.

We now proceed to refine this statement. Let $\hat{\mathcal{T}}_{i_p}$ denote the computed matrix $\mathcal{T}_{i_p}$. We take, as the diagonal elements that have not been computed as yet, the difference between the original element of T and the corresponding computed shifted sum.

LEMMA 8.9. *The matrix $\hat{\mathcal{T}}_{i_p}$ is obtained from order of $\overline{\theta}$ perturbations in $\overline{\mathcal{T}}_{i_p}$, a matrix affected by $O(\overline{\theta})$ perturbations in T. Hence, if we denote by*

$$\lambda_1^{(i_p)} > \lambda_2^{(i_p)} > \cdots \lambda_{n_p-1}^{(i_p)} > \lambda_{n_p}^{(i_p)} > 0$$

the eigenvalues of $\mathcal{T}_{i_p}$, and by $\hat{\lambda}_i^{(i_p)}$ the corresponding eigenvalues of $\hat{\mathcal{T}}_{i_p}$, we get

$$\hat{\lambda}_{n_p}^{(i_p)} \quad \geq \quad \lambda_N(1 + O(\hat{\theta})) > 0,$$

i.e., the matrix $\hat{\mathcal{T}}_{i_p}$ is positive definite.

Proof. We note that the computed matrix $\hat{\mathcal{T}}_{i_p}$ is derived from perturbations of the order of $\overline{\theta}$ of the off diagonal elements of some matrix $\overline{\mathcal{T}}_{i_p}$ affected by $O(\overline{\theta})$ perturbations in T. Let us denote by $\overline{\lambda}_i^{(i_p)}$, $i = 1, \ldots, np$, the corresponding eigenvalues of $\overline{\mathcal{T}}_{i_p}$, then,

$$(1 + O(\hat{\theta}))\lambda_N \leq \overline{\lambda}_{n_p}^{(i_p)}, \qquad \overline{\lambda}_1^{(i_p)} \leq \lambda_1(1 + O(\hat{\theta})),$$

and therefore

$$\hat{\lambda}_{n_p}^{(i_p)} = \overline{\lambda}_{n_p}^{(i_p)} + O(\overline{\theta})\overline{\lambda}_1^{(i_p)} = \overline{\lambda}_{n_p}^{(i_p)}(1 + O(\overline{\theta})(\lambda_1/\lambda_N)) \geq \lambda_N(1 + O(\hat{\theta})) > 0,$$

as required. □

Proof of Theorem 8.2. The two pivots are computed from $\hat{\mathcal{T}}_{i_p}$ by a sequence of transformations almost as in Lemmas 8.5, 8.6 and 8.7. The only difference is the following. The diagonal elements of $\hat{\mathcal{T}}_{i_p}$ are given by pair of numbers, the first being the original element of the tridiagonal matrix, and the other the accumulated partial sum. Hence, beside the standard relative perturbations of the order of $\overline{\theta}$, we may get additional absolute perturbations of the order of $\theta\lambda_1$, which are negligible. Hence, the proof is the same as in Lemma 8.5. Moreover, since perturbations are probably of the order of θ only, we should expect the relative errors to be bounded by a quantity of the order of $\overline{\theta}$. □

9. Conclusions and Open Problems. The results of this paper can be easily generalized in two directions, i.e. to block-tridiagonal and to band matrices. In both cases the algorithmic framework remains essentially the same. In addition, it is possible to apply similar ideas to the development of a parallel version of the QR algorithm.

Future work to be done include a detailed analysis of the latter, together with a comparison with previous work by two of the authors [9].

REFERENCES

[1] E. Anderson et al., *LAPACK Users' Guide*, SIAM, 1992.

[2] I. Bar-On, *A practical parallel algorithm for solving band symmetric positive definite systems of linear equations*, ACM Trans. Math. Softw., 13 (1987), pp. 323–332.

[3] ———, *Efficient logarithmic time parallel algorithms for the Cholesky decomposition of band matrices*, Tech. Report 661, Technion, Computer Science Department, 1990.

[4] ———, *A new divide and conquer parallel algorithm for the Cholesky decomposition of band matrices*, Tech. Report 667, Technion, Computer Science Department, 1991. Submitted to SIMAX.

[5] ———, *Fast parallel LR and QR algorithms for symmetric band matrices*, Tech. Report 749, Technion, Computer Science Department, 1992. Submitted to TOMS.

[6] ———, *Interlacing properties for tridiagonal symmetric matrices with applications to parallel computing*, Tech. Report 9317, Technion, Computer Science Department, 1993.

[7] ———, *Parallel computation of the spectrum of a symmetric tridiagonal matrix*, Tech. Report 9318, Technion, Computer Science Department, 1993.

[8] ———, *A new divide and conquer parallel algorithm for computing the eigenvalues of a symmetric tridiagonal matrix*, Tech. Report 832, Technion, Computer Science Department, 1994. Presented in ICCAM 94, Submitted to the Journal of Computational and Applied Mathematics.

[9] I. Bar-On and B. Codenotti, *A fast and stable parallel QR algorithm for symmetric tridiagonal matrices*, Linear algebra and its applications, (1994). To Appear.

[10] J. Barlow and J. Demmel, *Computing accurate eigensystems of scaled diagonally dominant matrices*, Siam J. on Numerical Analysis, 27 (1990), pp. 762–791.

[11] B. Codenotti and M. Leoncini, *Introduction to Parallel Processing*, Addison-Wesley Publishing Company, 1993.

[12] J. Demmel and W. Kahan, *Accurate singular values of bidiagonal matrices*, Siam J. Sci. Stat. Comput., 11 (1990), pp. 873–912.

[13] J. J. Dongarra and A. H. Sameh, *On some parallel banded system solvers*, Parallel Computing, 1 (1984), pp. 223–235.

[14] Z. Drmač, M. Omladič, and K. Veselić, *On the perturbation of the Cholesky factorization*, Siam J. on Matrix Analysis and Applications, 15 (1994), pp. 1319–1332.

[15] K. V. Fernando and B. Parlett, *Accurate singular values and differential qd algorithms*, Numeriche Mathematics, 67 (1994), pp. 191–229.

[16] J. Francis, *The QR transformation, part I*, Computer J., 4 (1961), pp. 265–271.

[17] ———, *The QR transformation, part II*, Computer J., 4 (1962), pp. 332–345.

[18] G. H. Golub and C. F. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, 1989.

[19] J. Grad and E. Zakrajšek, *LR algorithm with Laguerre shift for symmetric tridiagonal matrices*, The Computer Journal, 15 (1972), pp. 268–270.

[20] B. N. Parlett, *Laguerre's method applied to the matrix eigenvalue problem*, Mathematics of Computation, 18 (1964), pp. 464–485.

[21] H. Rutishauser, *Solution of eigenvalue problems with the LR transformation*, Nat Bur. Standards, AMS, 49 (1958), pp. 47–81.

[22] H. Rutishauser and H. Schwarz, *The LR transformation method for symmetric matrices*, Numeriche Mathematics, 5 (1963), pp. 273–289.

[23] A. Sameh and D. Kuck, *On stable parallel linear system solver*, J. Assoc. Comput. Mach., 25 (1978), pp. 81–91.

[24] H. S. Stone, *Parallel tridiagonal equation solver*, ACM Trans. Math. Softw., 1 (1975), pp. 289–307.

[25] J. Sun, *Componentwise perturbation bounds for some matrix decompositions*, BIT, 32 (1992), pp. 702–714.

[26] P. Swarztrauber, *A parallel algorithm for solving general tridiagonal equations*, Mathematics of Computation, 33 (1979), pp. 185–199.

[27] Thinking Machine Coroporation, *The Connection Machine CM-5 Technical summary*, Oct. 1991.

[28] J. H. Wilkinson, *A priori error analysis of algebraic processes*, in Proc. International Congress Math., 1968, pp. 629–639. Moscow: Izdat.

[29] ———, *The Algebraic Eigenvalue Problem*, Oxford Science Publications, 1988.