RIACS

Research Institute for Advanced Computer Science
NASA Ames Research Center

IN-61
43100

An O($\log^2 N$) Parallel Algorithm For Computing the Eigenvalues of a Symmetric Tridiagonal Matrix

p. 27

Paul N. Swarztrauber

December 1989

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.49

NASA Cooperative Agreement Number NCC 2-387

An $O(\log^2 N)$ Parallel Algorithm For Computing the Eigenvalues of a Symmetric Tridiagonal Matrix

Paul N. Swarztrauber

December 1989

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.49

NASA Cooperative Agreement Number NCC 2-387

An $O(\log^2 N)$ Parallel Algorithm For Computing the Eigenvalues of a Symmetric Tridiagonal Matrix

Paul N. Swarztrauber

Research Institute for Advanced Computer Science
NASA Ames Research Center - MS: 230-5
Moffett Field, CA 94035

RIACS Technical Report 89.49

December 1989

The Research Institute of Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 311, Columbia, MD 244, (301)730-2656

Work reported herein was supported in part by Cooperative Agreements NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

An $O(\log^2 N)$ parallel algorithm for computing the eigenvalues of a symmetric tridiagonal matrix

by

Paul N. Swarztrauber [1,2]

December 1989

ABSTRACT

An $O(\log^2 N)$ parallel algorithm is presented for computing the eigenvalues of a symmetric tridiagonal matrix using a parallel algorithm for computing the zeros of the characteristic polynomial. The method is based on a quadratic recurrence in which the characteristic polynomial is constructed on a binary tree from polynomials whose degree doubles at each level. Intervals that contain exactly one zero are determined by the zeros of polynomials at the previous level which ensures that different processors compute different zeros. The exact behavior of the polynomials at the interval endpoints is used to eliminate the usual problems induced by finite precision arithmetic.

1 National Center for Atmospheric Research, Boulder, Colorado 80307, which is sponsored by the National Science Foundation.

2 This work was supported by the NAS Systems Division via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA). This work was performed while this author was visiting the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035.

1. Introduction. A parallel algorithm is presented for computing the eigenvalues of a symmetric tridiagonal matrix.

$$A = \begin{bmatrix} b_1 & c_1 & & & & \\ c_1 & b_2 & c_2 & & & \\ & c_2 & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & c_{N-1} \\ & & & & c_{N-1} & b_N \end{bmatrix} \tag{1.1}$$

If $c_i = 0$ the problem can be reduced to two independent eigenproblems and hence it is customary to assume that $c_i \neq 0$. However, we do not make use of this fact and the algorithm is valid for near multiple or multiple zeros. We begin with a brief review of existing methods. In 1981, Cuppen developed a method based on the splitting

$$A = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \begin{bmatrix} 0 & C \\ C^T & 0 \end{bmatrix} \tag{1.2}$$

where T_1 and T_2 are symmetric tridiagonal matrices and C has one nonzero element $c_{N/2}$ in its lower left hand corner. The eigenvalues of T_1 and T_2 can be computed in parallel followed by an update procedure in which the eigenvalues of A are computed from the eigenvalues of T_1 and T_2. This approach can be applied recursively by splitting T_1 and T_2 and so forth until matrices of order one are obtained. The parallelism is now evident in the recursive process of computing the eigenvalues, of successively larger matrices from those of smaller matrices. This approach has been analyzed extensively by Dongarra and Sorenson, 1987, who solved many of the practical problems that arise in its implementation and demonstrated that zero finding provides a viable technique for computing eigenvalues. In particular they used deflation to improve reliability and performance (Bunch, Nielson, and Sorenson, 1978).

Eigenvalues have also been computed as the zeros of the characteristic polynomial of A. A variant of bisection, called multisectioning was recently developed and analyzed (Lo, Philippe, and Sameh, 1987). Sturm sequences are developed on multiple subintervals to isolate the eigenvalues. The eigenvalues are then determined by bisection or Newton's method or the zero-in method that combines bisection and the secant method. Multisectioning is particularly appropriate if only a few eigenvalues and/or eigenvectors are of interest. Ipsen and Jessup, 1989, have made a detailed comparison of Cuppens method, bisection and multisectioning on the hypercube.

In this paper, the eigenvalues are also computed as the zeros of the characteristic polynomial. The polynomial is evaluated on a binary tree structure using a quadratic recurrence in which the degree of the polynomials doubles at each step (Swarztrauber, 1979). Using N processors, the characteristic polynomial can be evaluated in $0(\log N)$ time. Krishnakumar and Morf, 1986, also use this quadratic recurrence to compute the eigenvalues of a symmetric tridiagonal matrix in $O(N\log N)$ time. Their method of separating the zeros is different from the one presented here.

A fundamental problem in parallel zero finding is to ensure that different processors compute different zeros. We show that the zeros of the polynomials at any step in the quadratic recurrence are separated by the zeros of the polynomials at the previous step. Hence the zeros can be determined by recurrsion beginning with the single zeros of linear polynomials and ending with the zeros of the characteristic polynomial. The precise nature of the separation is given in section 3.

There are two other problems associated with the implementation of zero finding methods for characteristic polynomials, both of which are associated with the use of finite precision arithmetic. First, even if $c_i \neq 0$, the zeros may not be separated because of rounding errors. A procedure is presented in section 5 that permits the automatic detection and handling of these cases without a machine dependent test. Second, for large N and/or $||\mathbf{A}||$, the possibility of over/underflow exists when computing the characteristic polynomial. Usually this can be handled by an apriori scaling of **A** but a more reliable approach is required if the algorithm is used in general purpose software.

An acceptable alternative is to use dynamic parallel scaling in which the intermediate computations are maintained at 0(1), (Swarztrauber, 1979). Another alternative is to reformulate the problem in terms of self-scaling rational functions. Both alternatives are presented in section 5 and their accuracy is compared with the QR algorithm in section 6.

2. A Parallel Algorithm For Evaluating the Characteristic Polynomial.

The characteristic polynomial will be computed in terms of characteristic polynomials $d_{i,j}$ of submatrices consisting of rows (and columns) i through j of A. Expanding about the jth row we obtain the well known three term recurrence relation

$$d_{i,j} = (b_j - \lambda)d_{i,j-1} - c_{j-1}^2 d_{i,j-2}. \qquad (2.1)$$

If we define the sequence

$$e_{i,j} = c_j d_{i,j-1} \qquad (2.2)$$

we obtain the two term matrix recurrence

$$\begin{bmatrix} d_{i,j} \\ e_{i,j} \end{bmatrix}^T = \begin{bmatrix} d_{i,j-1} \\ e_{i,j-1} \end{bmatrix}^T \begin{bmatrix} b_j-\lambda & c_j \\ -c_{j-1} & 0 \end{bmatrix}. \qquad (2.3)$$

To solve this recurrence relation we define

$$\mathbf{Q}_{i,j} = \prod_{k=i}^{j} \begin{bmatrix} b_k-\lambda & c_k \\ -c_{k-1} & 0 \end{bmatrix}. \tag{2.4}$$

Next we show that (2.4) has the closed form

$$\mathbf{Q}_{i,j} = \begin{bmatrix} d_{i,j} & c_j d_{i,j-1} \\ -c_{i-1}d_{i+1,j} & -c_{i-1}c_j d_{i+1,j-1} \end{bmatrix}. \tag{2.5}$$

with elements that can be determined from the characteristic polynomials of four submatrices. The desired characteristic polynomial $d_{1,N}$ is given as the upper left element of $\mathbf{Q}_{1,N}$. The proof of (2.5) is by induction on j. Equation (2.5) can be verified by direct computation for $j=i+1$. If we define $d_{i+1,i} = d_{i,i-1} = 1$ and $d_{i+1,i-1} = 0$ then equation (2.5) is also true for $j=i$. Now assume that it is true for $j-1$, then

$$\mathbf{Q}_{i,j} = \begin{bmatrix} d_{i,j-1} & c_{j-1}d_{i,j-2} \\ -c_{i-1}d_{i+1,j-1} & -c_{i-1}c_{j-1}d_{i+1,j-2} \end{bmatrix} \begin{bmatrix} b_j-\lambda & c_j \\ -c_{j-1} & 0 \end{bmatrix} \tag{2.6}$$

After matrix multiplication, (2.1) can be used with (2.6) to verify (2.5) which completes the proof.

The associative property of matrix multiplication provides a splitting formula that is fundamental to the parallel algorithm. For any k

$$\mathbf{Q}_{i,j} = \mathbf{Q}_{i,k}\mathbf{Q}_{k+1,j}. \tag{2.7}$$

Consider now the parallel algorithm for computing $\mathbf{Q}_{1,N}$ for the case $N = 8$.

Step 1 Compute

$$\mathbf{Q}_{2i-1,2i} = \begin{bmatrix} b_{2i-1}-\lambda & c_{2i-1} \\ & \\ -c_{2i-2} & 0 \end{bmatrix} \begin{bmatrix} b_{2i}-\lambda & c_{2i} \\ & \\ -c_{2i-1} & 0 \end{bmatrix}. \tag{2.8}$$

for $i = 1,2,3,$ and 4.

Step 2 Compute

$$\mathbf{Q}_{1,4} = \mathbf{Q}_{1,2}\mathbf{Q}_{3,4} \text{ and } \mathbf{Q}_{5,8} = \mathbf{Q}_{4,5}\mathbf{Q}_{6,7}. \tag{2.9}$$

Step 3 Compute

$$\mathbf{Q}_{1,8} = \mathbf{Q}_{1,4}\mathbf{Q}_{5,8}. \tag{2.10}$$

The computations within each step can be performed simultaneously. For general N, the first step requires 5 multiplications and 2 additions per matrix multiplication for a total of $3.5N$ flops. The rth step requires $12N/2^r$ flops for $r = 2,\ldots,\log_2 N$ or about $6N$ flops. Therefore on a single processor, the computation of the characteristic polynomial requires $6.5N$ flops. With N processors the first step requires 4 flops (since 2 processors are available for each matrix multiply), 3 flops are required for the second step and 2 flops are required for each step thereafter. A minimum of two flops are required since the additions must follow the multiplications. Hence a total of $2\log_2 N+3$ flops are required to compute the characteristic polynomial using N processors. In the sections that follow we will use the element-wise form of (2.7), which is obtained by substituting (2.5) into (2.7).

$$d_{i,j} = d_{i,k}d_{k+1,j} - c_k^2 d_{i,k-1}d_{k+2,j}. \tag{2.11a}$$

$$d_{i,j-1} = d_{i,k}d_{k+1,j-1} - c_k^2 d_{i,k-1}d_{k+2,j-1}. \tag{2.11b}$$

$$d_{i+1,j} = d_{i+1,k}d_{k+1,j} - c_k^2 d_{i+1,k-1}d_{k+2,j}. \tag{2.11c}$$

$$d_{i+1,j-1} = d_{i+1,k}d_{k+1,j-1} - c_k^2 d_{i+1,k-1}d_{k+2,j-1}. \tag{2.11d}$$

Equations (2.11b) through (2.11d) are the same as (2.11a) but with a suitable shift in the subscripts i and j. Nevertheless all four equations are required to "close" or complete the recurrence relations.

The parallel algorithm developed in this section can be used to solve a general tridiagonal system of equations (Swarztrauber, 1979). It has also been used (Krishnakumar and Morf, 1986) as part of a parallel algorithm for computing the eigenvalues of a symmetric tridiagonal matrix. Their method of separating the zeros is different from the one given in this paper.

3. The Separation Theorem. The fundamental problem with zero finding on a multiprocessor is ensuring that different processors compute different zeros. In this section we will show that if λ_l are the collated zeros of the four characteristic polynomials on the right side of equation (2.11a) then the zeros of $d_{i,j}$ occur one per interval in every other interval $(\lambda_{2l}, \lambda_{2l+1})$. This result also holds for (2.11b) through (2.11d) but it will only be demonstrated for (2.11a) since the proofs are almost identical. The separation theorem is fundamental to the parallel algorithm and ensures that each processor will find a different zero. To prove the theorem and develop the precise nature of the separation we will need the following three lemmas.

Lemma 1

Let $p(x) = p_0 + \cdots + p_k x^k$ and $q(x) = q_0 + \cdots + q_{k-1} x^{k-1}$ be polynomials with real and strictly interlacing zeros. If p_k and q_{k-1} have the same sign then $r(x) = p(x)/q(x)$ is monotone increasing on any interval that does not contain a zero of $q(x)$. If they have the opposite sign then $r(x)$ is monotone decreasing. More specifically, if $p_k/q_{k-1} > 0$ then $r'(x) > p_k/q_{k-1}$. If $p_k/q_{k-1} < 0$ then $r'(x) < p_k/q_{k-1}$.

Proof: The partial fraction expansion of $r(x)$ is

$$r(x) = \frac{p_k}{q_{k-1}} x + \frac{p_{k-1} q_{k-1} - p_k q_{k-2}}{q_{k-1}^2} + \sum_{l=1}^{k-1} \frac{w_l}{(x - \alpha_l)}. \tag{3.3}$$

where α_l are the zeros of $q(x)$ and $w_l = p(\alpha_l)/q'(\alpha_l)$. Therefore

$$r'(x) = \frac{p_k}{q_{k-1}} - \sum_{l=0}^{k-1} \frac{w_l}{(x - \alpha_l)^2}. \tag{3.4}$$

If p_k and q_{k-1} are both greater than zero then $\text{sign}[p(\alpha_l)] = \text{sign}(-1)^{l-k+1}$ and $\text{sign}[q'(\alpha_l)] = \text{sign}(-1)^{l-k}$ and hence $w_l < 0$. This result can be combined with

(3.4)

to complete the proof of Lemma 1 for this case. The remaining three cases are handled in a similar fashion.

Lemma 2

Let $R(x) = r_1(x)r_2(x) - c^2$ where $r_1(x) = p^{(1)}(x)/q^{(1)}(x)$ and $r_2(x) = p^{(2)}(x)/q^{(2)}(x)$ are like $r(x)$ in Lemma 1. Also let $c \neq 0$ be real and λ_l be the zeros of $p^{(1)}(x)$, $q^{(1)}(x)$, $p^{(2)}(x)$ and $q^{(2)}(x)$ which are assumed to be real, distinct, and ordered. Then $R(x)$ can not have a zero in both adjacent intervals $(\lambda_{l-1},\lambda_l)$ and $(\lambda_l,\lambda_{l+1})$.

Proof: Only one of the polynomials $p_1(x)$, $q_1(x)$, $p_2(x)$ or $q_2(x)$ changes sign at λ_l and hence $r_1(x)r_2(x)$ must be negative on one of the intervals which therefore does not contain a zero of $R(x)$.

Lemma 3

Let $R(x)$ be defined as in Lemma 2 where the degrees of $r_1(x)$ and $r_2(x)$ are l and m respectively. Also let the high order coefficients in the polynomials satisfy $\text{sign}[p_l^{(1)}/q_{l-1}^{(1)}] = \text{sign}[p_m^{(2)}/q_{m-1}^{(2)}]$. Then $R(x)$ does not have more than one zero in any interval $(\lambda_l,\lambda_{l+1})$.

Proof: From Lemma 1, $r'_1(x)$ and $r'_2(x)$ have the same sign. If $R(\alpha)$ is zero then $r_1(x)$ and $r_2(x)$ also have the same sign. From the chain rule for differentiation $r_1(x)r_2(x)$ must be monotone which implies that α is a unique zero.

The four sets of eigenvalues belong to the four characteristic polynomials listed on the left side of equations (2.11a-d). The eigenvalues of A are computed in the first of the four steps listed above when $r = s$ and hence the remaining three sets do not have to be computed. The amount of computation doubles when r increases by one until $r = s$ when it is halved. Therefore the amount of computation for the intervals that contain the eigenvalues of A is about four times the amount that would be required to compute the eigenvalues of A if the intervals were known.

Let n_e be the maximum number of polynomial evaluations that are necessary to determine a zero. For each $r = 1,\ldots,log_2 N$ a total of about $4N$ zeros of characteristic polynomials with degree 2^r must be computed. From section 2, $6.5 \cdot 2^r$ flops are required to evaluate a polynomial on a single processor. Therefore a total of

$$TF_1 = 26 n_e N \sum_{r=1}^{\log_2 N} 2^r < 52 n_e N^2 \tag{4.1}$$

flops are required to compute the eigenvalues of a symmetric tridiagonal matrix on a single processor (using the parallel algorithm for evaluating the polynomials. With N processors each zero can be determined on a separate processor for a total flop count of

$$TF_N < 52 n_e N. \tag{4.2}$$

With N^2 processors a characteristic polynomial with degree 2^r can be computed with $2r+3$ flops using the algorithm given in section 2. In addition each zero can be determined on a separate processor so the total number of flops is

$$TF_{N^2} \leq 4n_e \sum_{r=1}^{\log_2 N} (2r+3) < 4n_e \log_2^2 N. \tag{4.3}$$

This is the count for computing the eigenvalues of a symmetric tridiagonal matrix with N^2 processors. Note that the four polynomials in (2.11a-d) could be evaluated independently with additional processors for a minimum of $n_e \log^2 N$ flops.

In the development of the operation counts we have, in the traditional way, overlooked what could be a significant contribution to the total computing time;

namely, the time required for communication. The collation of the zeros of the four characteristic polynomials in the separation theorem must also be performed in $O(\log^2 N)$ time or the overall algorithm cannot be considered to be $O(\log^2 N)$. The collation of $d_{i,k}$, $d_{k+1,j}$, $d_{i,k-1}$, and $d_{k+2,j}$ can be done in the following steps.

1. The zeros of $d_{i,k}$ and $d_{i,k-1}$ interlace and hence a shuffle can be used to combine them in an ordered sequence, say $\lambda_i^{(1)}$.
2. Similarly the The zeros of $d_{k+1,j}$ and $d_{k+2,j}$ can be shuffled to produce an ordered sequence, say $\lambda_i^{(2)}$.
3. Finally the ordered sequences $\lambda_i^{(1)}$ and and $\lambda_i^{(2)}$ can be merged to form the desired collated sequence.

For a polynomial of degree 2^r each of these steps can be performed with $O(r)$ parallel transmissions and hence the overall time for communication with N processors is proportional to

$$\sum_{r=0}^{\log_2 N} r \approx \log_2^2 N \tag{4.4}$$

Therefore the overall algorithm including communication is $O(\log^2 N)$ if the architecture of the multiprocessor supports the algorithmic requirements of the shuffle and merge. The hypercube and related interconnection topologies support both the parallel computation and communication that are implicit in the algorithms presented here.

5. Implementation. In this section we will develop both a polynomial and rational function implementation of the algorithm. The polynomial implementation may require scaling to avoid over/underflow but it is more accurate than the rational function implementation. We determine the behavior of the functions at the endpoints of the intervals to eliminate the usual problems associated with finite precision arithmetic. This also provides both implementations with a high level of reliability. The computation of near multiple zeros (or multiple zeros if $c_i = 0$) is facilitated by multiple intervals with zero (or near zero length) that provide the correct multiplicity. The purpose of this section is to provide certain details of the implementation that are necessary before the algorithm can be considered of practical use.

A. Polynomial Implementation.

In this implementation the polynomials are computed using the parallel algorithm that was given in section 2. We do not discuss the zero finding method itself other than to note that the method of bisection was used for the results presented in section 6. A considerable amount of literature is available on this topic and many options exist.

The reliability of the algorithm is greatly enhanced by knowing the signs of the characteristic polynomials at the endpoints of the intervals. Because the eigenvalue enters the polynomial with a negative sign, i.e. as $b_i-\lambda$ on the diagonal of A, the high order term in $d_{i,j}$ is $(-1)^{j-i+1}x^{j-i+1}$. But $j-i+1=2^r$ is an even integer which combined with the separation theorem, implies that the signs of $d_{i,j}$ on the intervals $\lambda_l(i:k:j)$ for $l=0,...,2^{r+1}-1$ are $+ - - + + \cdots + + - - +$. Similarly the signs of $d_{i,j-1}$ on $\lambda_l(i:k:j-1)$ for $l=0,...,2^{r+1}-3$ are $+ - - + + \cdots - - + + -$. The signs of $d_{i+1,j}$ on $\lambda_l(i+1:k:j)$ for $l=0,...,2^{r+1}-3$ are $+ - - + + \cdots - - + + -$ and the signs of $d_{i+1,j-1}$ on $\lambda_l(i+1:k:j-1)$ for $l=0,...,2^{r+1}-5$ are $+ - - + + \cdots + + - - +$.

In practice there are two reasons why these sign patterns can be interrupted. First, as previously noted, $d_{i,k}$ and $d_{k+2,j}$ are characteristic polynomials of different submatrices and may therefore share a common zero that would also be a zero of $d_{i,j}$. Then, with finite precision, a small value could be obtained for $d_{i,j}$ with the wrong sign. Second, although in theory the zeros of $d_{i,k}$ and $d_{i,k-1}$ (or $d_{k+1,j}$ and $d_{k+2,j}$) interlace, with finite precision they may coalesce or cross and again produce a small number with the opposite sign. Both cases occur when a zero of the characteristic polynomial has already been found to machine precision and it would therefore seem reasonable to select one or the other as a zero. However, endpoint zeros might belong to a different interval in the presence of near multiple or multiple zeros. The most satisfactory approach is to replace any small end point value, whose sign differs from the correct sign, by the machine epsilon divided by, say 10, with the correct sign. The bisection method or variants thereof can then be used to select the zero. This procedure is fundamental to the reliability of the algorithm and guarantees that N zeros will be found. Note that these tests do not require a machine dependent test, i.e. only sign tests are required.

In practice, some of the intervals usually shrink to zero as the computation proceeds. This is beneficial because it reduces the amount of computation that is required to compute the zeros. This "deflation" is similar to that reported by Dongarra and Sorenson, 1987, who observed that it could make their method competitive with the QR algorithm on a single processor. However there are important cases where deflation does not occur, e.g., the tridiagonal matrix with $b_i = -2$ and $c_i = 1$. For cases where deflation occurs it is likely that the length of the intervals is about machine precision and not identically zero. This could be detected using a machine dependent test; however, a more satisfactory approach has been to detect interlace faults and set the zeros that have crossed to their average value. This creates an environment where the length of many intervals is identically zero and detectable without the use of a machine dependent test.

For large N or $||\mathbf{A}||$ the characteristic polynomial may over/underflow the arithmetic unit. This can be avoided by scaling the intermediate 2×2 matrices that occur during the parallel algorithm, (Swarztrauber, 1979). Scaling is not required at each level r so the time required for scaling can be kept to a minimum. For example, if scaling is performed for $r=5,10,15,\ldots$ about $N/32$ matrices are scaled. If "power of two" scaling is used then accuracy is unchanged and the time is negligible, particularly if it is done in machine language. Scaling can also be eliminated by a reformulation in terms of rational functions.

B. Rational Function Implementation.

There are three reasons to consider a second implementation of the algorithm. First, the rational function is self-scaling; second, one can take further advantage of deflation to reduce the order of the rational functions; and third, the operation count for the rational function implementation is somewhat less than the polynomial implementation. This must be weighed against a loss of accuracy compared with the polynomial implementation. The loss is not substantial since only two binary bits are lost when compared with the QR algorithm for a matrix with order 1024. Accuracy tables are provided in section 6.

The eigenvalues of $\mathbf{A}$ are computed as the zeros of the rational function $R(x)$ that was introduced in Lemma 2, section 3, namely

$$R(x) = r_1(x)r_2(x) - c_k^2 \qquad (5.1)$$

where $r_1(x) = d_{i,k}/d_{i,k-1}$ and $r_2(x) = d_{k+1,j}/d_{k+2,j}$. All polynomials are evaluated and stored in factored form. Like the polynomial implementation, it is necessary, from a practical point of view, to know the behavior of $R(x)$ at the end points of the interval $[\alpha_1,\alpha_2]$ under consideration. If a zero of (5.1) exists then $r_1(x)$ and $r_2(x)$ must have the same sign. They also satisfy the conditions of Lemma 1 which ensure that they are both monotone decreasing functions. Since the ratio of the high order coefficients is -1 for both $r_1(x)$ and $r_2(x)$. These conditions are satisfied only if $d_{i,k}$ or $d_{k+1,j}$ are zero at one end of the interval and $d_{i,k-1}$ or $d_{k+2,j}$ are zero at the other which implies that only two cases are possible, namely the ones listed in steps 4. and 5. below. Consider now the steps that must be taken to ensure a reliable implementation.

1. Like the polynomial implementation, any interlace faults are corrected by replacing the zeros that have crossed by their average value.
2. Any zeros that are identically common to both the numerator and denominator of $r_1(x)r_2(x)$ are removed and selected as zeros of $d_{i,j}$. This deflation can substantially reduce the amount of computation. A matrix with order $N=4096$ is presented in section 6 for which the maximum order of any computed polynomial is 24!
3. If $\alpha_1 = \alpha_2$ then α_1 is selected as a zero of $d_{i,j}$. This deflation step can be used with both implementations.
4. If $\alpha_1 \neq \alpha_2$ and either $d_{i,k}$ or $d_{k+1,j}$ are zero at α_1 then $R(\alpha_1)$ is set to $-c_k^2$ and $R(\alpha_2)$ is set to a large positive number.
5. If $\alpha_1 \neq \alpha_2$ and either $d_{i,k-1}$ or $d_{k+2,j}$ are zero at α_1 then $R(\alpha_1)$ is set to a large positive number and $R(\alpha_2)$ is set to $-c_k^2$.

Case 4. applies to the first interval and case 5. applies to the last interval but with minor modifications. Machine independent tests can be used to determine which of the cases 1.-5. apply. Once the values of $R(x)$ are established using step 4. or 5. the zeros can be determined using bisection or any variant thereof. Note that this does not preclude the use of another method such as the one used by Dongarra and Sorenson, 1987, if it is combined with bisection.

Although scaling is not required for this implementation, the rational functions $r_1(x)$ and $r_2(x)$ should be computed as a product of quotients $(x-\lambda_i)/(x-\beta_i)$ where λ_i and β_i are chosen as close as possible. This prohibits the growth of intermediate computations that might result in over/underflow.

6. Computational results. The accuracy of implementations A and B are compared with the QR algorithm as implemented in subroutine TQL1 in EISPACK (Smith, et.al.,1976) for symmetric tridiagonal matrices. The entries in the tables below are computed from

$$e = \frac{\max_i |\lambda_i - \lambda_i^*|}{\max_i |\lambda_i^*|} \tag{6.1}$$

where λ_i is computed in single precision and λ_i^* is computed using a double precision version of subroutine TQL1. All the computations were done on the CRAY-2 computer located at NASA Ames Research Center.

Three tables are presented that correspond to three different matrices. Table 1 contains a comparison of accuracy for a matrix with random coefficients. The matrix corresponding to Table 2 is W_{2k}^+ which is a slight variant of W_{2k-1}^+ (Wilkinson, 1965, p.308). The variant is used because the parallel algorithms were implemented for matrices with order equal to a power of 2. Although this restriction can be removed it nevertheless simplifies implementation. W_{2k}^+ is of interest because it tests the ability of a method to handle near multiple eigenvalues. Table three corresponds to the matrix with zeros on the diagonal and ones adjacent to the diagonal. The following observations can be made from the tables.

1. The accuracy of the polynomial implementation is always superior to the accuracy of the QR algorithm. This result can be established by comparing the second and third columns in Tables 1,2 and 3.

2. The accuracy of rational function implementation is also superior to the QR algorithm for matrices with random elements or the matrices W_{2k}^+. These results can be established by comparing columns two and four in Tables 1 and 2.

3. The accuracy of the QR algorithm is superior to the accuracy of the rational function implementation for the matrix with zeros on the diagonal and 1s in the sub and super diagonals. This result can be established by comparing columns two and four in Table 3. The difference is not substantial, for

$N = 1024$ they differ in only the last two bits

Table 1

Accuracy of the QR method and two implementations of the parallel algorithm for a symmetric tridiagonal matrix with random coefficients

N	QR	PI	RI
128	5.67×10^{-13}	2.06×10^{-14}	3.95×10^{-13}
256	1.25×10^{-12}	3.95×10^{-14}	2.57×10^{-13}
512	1.82×10^{-12}	3.94×10^{-14}	7.79×10^{-13}
1024	3.42×10^{-12}	4.60×10^{-14}	7.79×10^{-13}
2048	7.33×10^{-12}	6.94×10^{-13}	1.61×10^{-12}
4096	1.32×10^{-11}	7.00×10^{-13}	2.14×10^{-12}

QR as implemented in subroutine TQL1 from EISPACK
PI is the polynomial implementation
RI is the rational function implementation

Table 2

Accuracy of the QR method and two implementations of the parallel algorithm for a symmetric tridiagonal matrix W_{2k}^{+} with near multiple eigenvalues

N	QR	PI	RI
128	2.63×10^{-13}	2.81×10^{-14}	1.05×10^{-14}
256	6.36×10^{-13}	3.53×10^{-14}	1.06×10^{-14}
512	1.28×10^{-12}	4.25×10^{-14}	1.42×10^{-14}
1024	2.55×10^{-12}	4.97×10^{-14}	1.06×10^{-14}
2048	5.31×10^{-12}	5.68×10^{-14}	1.07×10^{-14}
4096	9.88×10^{-12}	6.39×10^{-14}	1.42×10^{-14}

QR as implemented in subroutine TQL1 from EISPACK
PI is the polynomial implementation
RI is the rational function implementation

Table 3

Accuracy of the QR method and two implementations of the parallel algorithm for a symmetric tridiagonal matrix with zero diagonal and 1's off diagonal

N	QR	PI	RI
128	4.37×10^{-13}	5.33×10^{-14}	1.53×10^{-13}
256	7.35×10^{-13}	6.75×10^{-14}	6.64×10^{-13}
512	1.44×10^{-12}	1.46×10^{-13}	2.38×10^{-12}
1024	2.73×10^{-12}	2.06×10^{-13}	1.04×10^{-11}
2048	5.42×10^{-12}	3.45×10^{-13}	2.78×10^{-11}
4096	1.01×10^{-11}	6.25×10^{-13}	2.24×10^{-10}

QR as implemented in subroutine TQL1 from EISPACK
PI is the polynomial implementation
RI is the rational function implementation

The accuracy of the rational function implementation in Table 3 is somewhat less than in Tables 1 and 2, probably because deflation does not occur in Table 3. Deflation occurred for all the parallel computations in Tables 1 and 2 and appears to be the rule rather than the exception. For $N = 4096$ the eigenvalues of the matrix W_{2k}^{+} are given as the zeros of a polynomial of degree 4096; however because of deflation, the maximum degree of any computed rational polynomial is 24! Similarly, the maximum degree of any computed rational polynomial for Table 1 is 75. The eigenvalues of the matrix in Table 3 are $\lambda_i = 2\cos i\pi/(N+1)$ which are separated to the extent that deflation is minimal. Since the submatrices are identical some deflation does occur but the maximum degree is 4096 compared to 24 for the matrix W_{2k}^{+} presented in Table 2. This difference in computation may explain the difference between the fourth column of Table 3 and the fourth column in Tables 1 and 2.

The results in Table 2 show that near multiple zeros do not present any difficulties for the parallel method. Indeed the amount of computation may be reduced. Recall that the method produces exactly N intervals that contain one and only one zero and therefore, in the presence of near multiple zeros, the interval length can be near or identically zero. Therefore a zero can be obtained directly and with the proper multiplicity without using any zero finding methods. For intervals with near zero length the usual concern about slow convergence of Newton like methods for multiple zeros is not warranted because: first, the zeros are distinct to machine precision and second, any point on the interval provides a good initial approximation to the zero.

Since the interval end points are themselves zeros of characteristic polynomials it is reasonable to ask if they are subject to the usual concerns about near multiple zeros, i.e. has the problem simply been deferred to the interval endpoints? The answer is that they too may be found on adjacent short intervals or on a juxtaposition of intervals with near zero length. This line of reasoning continues recursively to the first level where linear polynomials provide the interval endpoints for quadratic polynomials. Since near multiple zeros are clearly not a problem at this level, by induction they are not a problem at any level.

The performance of the algorithm on a parallel computer is a complex issue that depends on many factors including:

a) Many methods could be used to determine the zeros within each interval including Newtons method, bisection, and the secant method together with variants and combinations. Zero finding is itself a significant area of computational mathematics. Dongarra and Sorenson, 1987, use a variant of Newtons method in which a local quadratic rational approximation to the function is computed. The zero-in method (Lo, Philippe, and Sameh, 1987) combines bisection and the secant method.

b) Deflation plays a significant role in reducing computing time on either a single or multiprocessor. The effects are two-fold: first, the number of computed zeros is reduced and second, the order of the rational functions is decreased. With deflation the computing time on a single processor for the rational function implementation was proportional to N rather than N^2. Dongarra and Sorenson, 1987, report performance superior to QR algorithm in the presence of deflation even on a single processor. However, as evidenced by Table 3, deflation does not always occur which verifies that a matrix with random coefficients does not provide a stringent test because it probably does not correspond to either the worst or the best case.

c) The algorithm requires global communication and will therefore perform better on parallel computers that provide efficient global parallel pathways such as those provided by the hypercube and related interconnection topologies.

d) The algorithm requires the implementation of the shuffle and merge communication tasks. To maintain an overall complexity of $O(\log^2 N)$ these tasks must be implemented with parallel communication algorithms on suitable multiprocessor architectures. Although communication does not change the overall complexity it is likely to make a significant contribution to the overall computing time.

7. Summary and conclusions. A parallel algorithm has been presented for computing the eigenvalues of a symmetric tridiagonal matrix in $O(\log^2 N)$ time using N^2 processors or $O(N)$ time using N processors. Attributes of the method that contribute to reliability and performance are: i) a separation theorem that ensures that different processors find different eigenvalues, ii) implementations that eliminate the usual problems associated with finite precision arithmetic, iii) reliable treatment of multiple and near multiple zeros, and iv) high accuracy. Two implementations of the algorithm were presented. The polynomial implementation is more accurate than the QR and the rational function implementation has comparable accuracy with deflation but is less accurate in the absence of deflation. Nevertheless it remains of interest because it is self scaling and takes full advantage of deflation both from the standpoint of having to compute fewer zeros of rational functions with lower degree.

ACKNOWLEDGEMENT

The author wishes to acknowledge a helpful conversation with William Gragg who pointed out the partial fraction expansion for computing the derivative of a rational function.

REFERENCES

J.R. Bunch, C.P. Nielsen, and D.C. Sorensen, 1978, Rank-one modification of the symmetric eigenproblem, Numer. Math., vol. 31, pp. 31-48.

J.J.M. Cuppen, 1981, A divide and conquer method for the symmetric tridiagonal eigenproblem, Numer. Math., vol. 36, pp. 177-195.

J.J. Dongarra and D.C. Sorenson, 1987, A fully parallel algorithm for the symmetric eigenvalue problem, SIAM J. Sci. Stat. Comput., vol. 8, pp. s139-s154.

I.C.F. Ipsen and E.R. Jessup, Solving the symmetric tridiagonal eigenvalue problem on the hypercube, SIAM J. Sci. Stat. Comput., to appear.

A.S. Krishnakumar and M. Morf, 1986, Eigenvalues of a symmetric tridiagonal matrix: a divide and conquer approach, Numer. Math., vol. 48, pp. 349-368.

S.-S. Lo, B. Philippe and A. Samed, 1987, A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem, SIAM J. Sci. Stat. Comput., vol. 8, pp. s155-s165.

B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler, 1976, Matrix Eigensystem Routines - EISPACK Guide, 2nd edition, Lecture Notes in Computer Science, no. 6, Springer-Verlag, New York.

P.N. Swarztrauber, 1974, A direct method for the discrete solution of separable elliptic equations, SIAM J. Numer. Anal., vol. 11, pp. 1136-1150.

P.N. Swarztrauber, 1979 A parallel algorithm for solving general tridiagonal equations, Math. Comp., vol. 33, pp. 185-199.

J.H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.