MATHEMATICS OF COMPUTATION
VOLUME 60, NUMBER 202
APRIL 1993, PAGES 651-668

A PARALLEL ALGORITHM FOR COMPUTING THE EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX

PAUL N. SWARZTRAUBER

ABSTRACT. A parallel algorithm, called polysection, is presented for computing the eigenvalues of a symmetric tridiagonal matrix. 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 signs of the polynomials at the interval endpoints are determined a priori and used to guarantee that all zeros are found. The use of finite-precision arithmetic may result in multiple zeros; however, in this case, the intervals coalesce and their number determines exactly the multiplicity of the zero. For an $N \times N$ matrix the eigenvalues can be determined in $O(\log^2 N)$ time with N^2 processors and $O(N)$ time with N processors. The method is compared with a parallel variant of bisection that requires $O(N^2)$ time on a single processor, $O(N)$ time with N processors, and $O(\log N)$ time with N^2 processors.

1. INTRODUCTION

A parallel algorithm, called polysection, is presented for computing the eigenvalues of a symmetric tridiagonal matrix

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

If $c_i = 0$ for some i, the problem can be reduced to two independent eigenproblems and hence it is customary to assume that the off-diagonal elements are nonzero. In theory this also separates the eigenvalues but in practice, with finite-precision arithmetic, "multiple" eigenvalues are not uncommon. The method

Received by the editor September 1, 1989 and, in revised form, March 26, 1991 and March 26, 1992.

1991 *Mathematics Subject Classification.* Primary 65F15, 65H05; Secondary 15A18, 26C10, 68Q10.

Key words and phrases. Parallel computing, symmetric tridiagonal matrix, eigenvalues.

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 completed while this author was visiting the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035.

0025-5718/93 $1.00 + $.25 per page

presented here is particularly effective in locating multiple eigenvalues. We begin with a brief review of existing methods.

The QR algorithm has been the method of choice for computing the eigenvalues of a matrix on a single processor. However, the choice is less clear in the context of parallel computing. Nevertheless, there is a noteworthy paper by Sameh and Kuch [13] which describes a parallel implementation of the QR algorithm for symmetric tridiagonal matrices. This paper, as well as the paper by Stone [17], for the parallel solution of symmetric tridiagonal systems, contains parallel algorithms for solving two- and three-term recurrence relations that provide key computational tools for much of the subsequent work in this area. More recently, Cuppen [3] developed a method based on the splitting

$$\mathbf{A} = \begin{bmatrix} \mathbf{T}_1 & 0 \\ 0 & \mathbf{T}_2 \end{bmatrix} + \begin{bmatrix} 0 & \mathbf{C} \\ \mathbf{C}^T & 0 \end{bmatrix}, \tag{1.2}$$

where $\mathbf{T}_1$ and $\mathbf{T}_2$ are symmetric tridiagonal matrices and $\mathbf{C}$ has one nonzero element in its lower left-hand corner. The eigenvalues of $\mathbf{T}_1$ and $\mathbf{T}_2$ can be computed in parallel, followed by an update procedure in which the eigenvalues of $\mathbf{A}$ are computed from the eigenvalues of $\mathbf{T}_1$ and $\mathbf{T}_2$. This approach can be applied recursively by splitting $\mathbf{T}_1$ and $\mathbf{T}_2$ 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 Sorensen [6] as well as Sorensen and Tang [16], who solved many of the practical problems that arise in its implementation and demonstrated that zerofinding provides a viable technique for computing eigenvalues. In particular, they used deflation to improve reliability and performance [2].

Eigenvalues have also been computed as the zeros of the characteristic polynomial of $\mathbf{A}$. Sturm sequences are fundamental to the method of bisection, which provides a straightforward method for computing eigenvalues. Bisection is based on the theorem that states that the number of sign changes in the Sturm sequence is equal to the number of eigenvalues that are less than the current estimate. As early as 1962, Wilkinson [20] stated that "the theorem [bisection] can be used to locate an individual eigenvalue without locating any of the others." Although he was not concerned with parallel computing at that time, his observation forms the basis of a parallel algorithm. The pth processor uses interval bisection to find the largest value such that the Sturm sequence has p agreements in sign; i.e., the pth processor determines the pth eigenvalue independent of the other processors. With N processors all eigenvalues can be computed in $O(N)$ time. This result is valid even if one uses the highly sequential three-term recurrence relation for computing the Sturm sequence. The accuracy of bisection is known to be high [4, 9] and superior to the QR algorithm.

A variant of bisection, called multisectioning, was recently developed and analyzed in [12]. 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 [8] have made a detailed comparison of Cuppen's method, and multisectioning on the hypercube.

In this paper we introduce a method, called polysection, which is based on the parallel algorithm developed in [17, 19] for computing the characteristic polynomial. Krishnakumar and Morf [10] also use this parallel algorithm to compute the eigenvalues of a symmetric tridiagonal matrix in $O(N \log N)$ time; however, their method of separating the zeros is different from the one presented here. The characteristic polynomial is evaluated on a binary tree structure using a quadratic recurrence in which the degree of the polynomials doubles at each step. 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 recursion, beginning with the single zeros of linear polynomials and ending with the zeros of the characteristic polynomial. Each processor is given an interval that contains a unique zero, which ensures that different processors compute different zeros. In the presence of multiple or near multiple zeros, the intervals tend to coalesce and their number determines the multiplicity of the zero. This eliminates the traditional problems experienced by some methods for computing multiple zeros. Computational results are presented in §7 for a large symmetric matrix which, to machine precision, has many eigenvalues with a multiplicity of two.

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 a priori scaling of $\mathbf{A}$; however, dynamic parallel scaling [17, 19] provides a more reliable approach in which catastrophic over/underflow is eliminated, since intermediate computations are maintained at $O(1)$. This approach is reviewed in §6. Over/underflow can also be avoided by reformulating the problem in terms of self-scaling rational functions. This approach together with the polynomial implementation are presented in §5 and their accuracy is compared with the QR and bisection algorithms in §7.

In §7, the accuracy of polysection is compared with a variant of bisection in which the Sturm sequence is computed using the parallel algorithm developed in [17, 19]. With this approach all eigenvalues can be computed in $O(\log N)$ time with N^2 processors. The numerical results in §7 show that the parallel algorithm for the Sturm sequence provides a highly accurate algorithm for the eigenvalues. The accuracy of both bisection and polysection is shown to be superior to the QR algorithm, particularly for the case of near multiple zeros. Bisection with the parallel computation of the Sturm sequence is discussed in §6. This straightforward variant will be called parallel bisection, and although it has not appeared in the literature it is known to the computational community (see acknowledgments).

The asymptotic time for bisection is less than polysection, and the two methods have comparable accuracy. These results, combined with the simplicity of bisection, make it reasonable to question the relevance of polysection. However, polysection has three attributes that are not necessarily shared with bisection. First, the theory of polysection developed in §§3–5 guarantees its reliability in the sense that N zeros will always be found. Although this attribute is shared with the serial bisection algorithm [21, pp. 304–305], it remains to be demonstrated for parallel bisection. Second, "deflation" can be used with polysection, which can substantially reduce the amount of computation. An example is given in §7 of a matrix with order 4096 whose eigenvalues were determined

from polynomials with degree less than 15. Third, Newton's or other high-order methods can be used with polysection to speed zerofinding. The most appropriate method may likely depend on the particular application.

2. A PARALLEL ALGORITHM FOR EVALUATING THE CHARACTERISTIC POLYNOMIAL

In this section we will review the parallel algorithm given in [17, 19] for computing the characteristic polynomial of $\mathbf{A}$. It is computed in terms of characteristic polynomials $d_{i,j}$ of principal submatrices consisting of rows (and columns) i through j. Expanding about the jth row, we obtain the well-known three-term recurrence relation

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

If we define the sequence

$$e_{i,j} = c_j d_{i,j-1}, \tag{2.2}$$

we obtain the two-term matrix recurrence

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

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 $i \le k \le j$,

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

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

Step 1. For $i = 1, 2, 3$, and 4 compute

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

Step 2. Compute

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

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 3 additions per matrix multiplication for a total of $4N$ flops. The rth step requires $12N/2^r$ flops for $r = 2, \ldots, \log_2 N$, which totals about $6N$ flops. Therefore, on a single processor, the computation of the characteristic polynomial totals about $10N$ 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 elementwise 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 [19], although equations (2.11) were first developed in block form for solving separable elliptic partial differential equations [18]. Although equations (2.11a–d) are valid for any $i < k < j$, the parallel algorithm presented in §4 uses $k = (i + j - 1)/2$.

3. The separation theorem

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 and that all zeros will be found. To prove the theorem and develop the precise nature of the separation, we will need the following four lemmas.

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

Proof. The partial fraction expansion of $r(\lambda)$ is

$$r(\lambda) = \frac{p_k}{q_{k-1}}\lambda + \frac{p_{k-1}q_{k-1} - p_k q_{k-2}}{q_{k-1}^2} + \sum_{l=1}^{k-1} \frac{w_l}{(\lambda - \lambda_l)}, \tag{3.3}$$

where λ_l are the zeros of $q(\lambda)$ and $w_l = p(\lambda_l)/q'(\lambda_l)$. Therefore,

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

If p_k and q_{k-1} are both greater than zero, then $\operatorname{sign}[p(\lambda_l)] = \operatorname{sign}(-1)^{l-k}$ and $\operatorname{sign}[q'(\lambda_l)] = \operatorname{sign}(-1)^{l-k+1}$ and hence $w_l < 0$. This result together with (3.4) implies the desired result $r'(\lambda) > p_k/q_{k-1}$. A similar proof for negative p_k and q_{k-1} yields the same result, and for p_k and q_{k-1} with opposite sign we obtain $r'(\lambda) < p_k/q_{k-1}$. □

Lemma 2. *Let* $R(\lambda) = r_1(\lambda)r_2(\lambda) - c^2$, *where* $r_1(\lambda) = p^{(1)}(\lambda)/q^{(1)}(\lambda)$ *and* $r_2(\lambda) = p^{(2)}(\lambda)/q^{(2)}(\lambda)$ *are like* $r(\lambda)$ *in Lemma* 1 *and* $c \neq 0$ *is an arbitrary real constant. Let* λ_l *be the collated zeros of* $p^{(1)}(\lambda)$, $q^{(1)}(\lambda)$, $p^{(2)}(\lambda)$, *and* $q^{(2)}(\lambda)$. *From Lemma* 1, *the sign of* $r_1'(\lambda)$ *is the same on all intervals* $(\lambda_{l-1}, \lambda_l)$. *We will assume that* $r_2'(\lambda)$ *has the same sign as* $r_1'(\lambda)$. *Define*

$$d(\lambda) = q^{(1)}(\lambda)q^{(2)}(\lambda)R(\lambda) = p^{(1)}(\lambda)p^{(2)}(\lambda) - c^2 q^{(1)}(\lambda)q^{(2)}(\lambda). \tag{3.5}$$

Then $d(\lambda)$ *does not have more than one zero in any interval* $(\lambda_l, \lambda_{l+1})$.

Proof. By hypothesis, $r_1'(\lambda)$ and $r_2'(\lambda)$ have the same sign. If $R(\alpha)$ is zero, then $r_1(\alpha)$ and $r_2(\alpha)$ must have the same sign and hence they have the same sign on the entire interval $(\lambda_{l-1}, \lambda_l)$. Therefore, $R'(\lambda) = r_1'(\lambda)r_2(\lambda) + r_1(\lambda)r_2'(\lambda)$ does not change sign on $(\lambda_{l-1}, \lambda_l)$, which implies that $R(\lambda)$ is monotone and hence α is a unique zero. Since $q^{(1)}(\lambda)q^{(2)}(\lambda) \neq 0$, α must also be a unique zero of $d(\lambda)$. □

Lemma 3. *Let* λ_l *and* $d(\lambda)$ *be defined as in Lemma* 2. *If* $d(\alpha) = 0$, *where* α *is in the open interval* $(\lambda_{l-1}, \lambda_l)$, *then both* $d(\lambda_{l-1}) \neq 0$ *and* $d(\lambda_l) \neq 0$; *i.e., neither of the interval endpoints are zeros of* $d(\lambda)$.

Proof. The product $q^{(1)}(\lambda)q^{(2)}(\lambda) \neq 0$ on $(\lambda_{l-1}, \lambda_l)$; hence from (3.5), $R(\alpha) = 0$. But from the proof of Lemma 2, $R(\lambda)$ is monotone and therefore nonzero at the endpoints λ_{l-1} and λ_l. If $d(\lambda_l) = 0$, then from (3.5) either $q^{(1)}(\lambda_l) = 0$ or $q^{(2)}(\lambda_l) = 0$. But then (3.5) also implies either $p^{(1)}(\lambda_l) = 0$ or $p^{(2)}(\lambda_l) = 0$. However, since the zeros of the q-polynomials strictly interlace with those of the p-polynomials, the only possibilities are either $p^{(1)}(\lambda_l) = q^{(2)}(\lambda_l) = 0$ or $p^{(2)}(\lambda_l) = q^{(1)}(\lambda_l) = 0$. Without loss of generality we can assume the former, which implies that $\lim_{\lambda \to \lambda_l} r_1(\lambda) = 0$ and $\lim_{\lambda \to \lambda_l} |r_2(\lambda)| = \infty$. However, this contradicts the hypothesis that $r_1'(\lambda)$ and $r_2'(\lambda)$ have the same sign and the result that $r_1(\lambda)$ and $r_2(\lambda)$ also have the same sign as demonstrated in the proof of Lemma 2. Hence, we obtain the desired result; namely, that $d(\lambda_l) \neq 0$. A similar proof yields $d(\lambda_{l-1}) \neq 0$. □

Lemma 4. *Let* λ_l *and* $d(\lambda)$ *be defined as in Lemma* 2. *Then* $d(\lambda)$ *has at most one zero in any two adjacent intervals* $(\lambda_{l-1}, \lambda_l)$, $(\lambda_l, \lambda_{l+1})$.

Proof. If neither interval has zero length, then only one of the polynomials $p^{(1)}(\lambda)$, $p^{(2)}(\lambda)$, $q^{(1)}(\lambda)$, $q^{(2)}(\lambda)$ changes sign at λ_l. Therefore, $p^{(1)}(\lambda)p^{(2)}(\lambda)$

and $q^{(1)}(\lambda)q^{(2)}(\lambda)$ must have opposite signs on one of the intervals, which by (3.5) implies that $d(\lambda)$ does not have a zero in that interval. If $d(\lambda)$ has a zero on the other interval, then by Lemma 2 it must be unique.

At least one of the intervals must have nonzero length, for otherwise some λ_l would have multiplicity greater than two, which contradicts the strict interlacing of the zeros of $q^{(1)}(\lambda)$ with those of $p^{(1)}(\lambda)$ and the zeros of $q^{(2)}(\lambda)$ with those of $p^{(2)}(\lambda)$. If the interval with nonzero length contains a zero, then, by Lemma 3, the interval with zero length does not contain a zero. This argument also implies that if the zero length interval contains a zero of $d(\lambda)$ then the interval with nonzero length does not contain a zero of $d(\lambda)$. Note that an open interval with zero length is replaced by a closed interval consisting of a single point. □

The separation theorem. *Let* λ_l *for* $l = 1, \ldots, 2(j-i)$ *be the ordered zeros of the characteristic polynomials* $d_{i,k}$, $d_{k+1,j}$, $d_{i,k-1}$, *and* $d_{k+2,j}$ *augmented with* λ_0 *and* $\lambda_{2(j-i)+1}$ *as the lower and upper bounds, respectively, on the zeros of* $d_{i,j}$. *If the off-diagonal elements of* **A** *are nonzero, then the characteristic polynomial* $d_{i,j}$ *defined in* (2.11a) *has* $j-i+1$ *zeros located one per every other interval in* $(\lambda_{2l}, \lambda_{2l+1})$ *for* $l = 0, \ldots, j-i$.

Proof. Set $d(\lambda) = d_{i,j}$, $p^{(1)}(\lambda) = d_{i,k}$, $q^{(1)}(\lambda) = d_{i,k-1}$, $p^{(2)}(\lambda) = d_{k+1,j}$, $q^{(2)}(\lambda) = d_{k+2,j}$, and $c = c_k$; then (2.11a) takes the form of (3.5). Before Lemmas 1 through 4 can be applied to (3.5) we have to establish the validity of two assumptions. First, Lemma 1 requires strict interlacing and second, Lemma 2 requires $r_1'(\lambda)$ and $r_2'(\lambda)$ to have the same sign. But strict interlacing of zeros is a well-known property of any sequence of polynomials that satisfy the three-term recurrence (2.1) with nonzero off-diagonal elements c_i. Therefore, it remains only to show that $r_1'(\lambda)$ and $r_2'(\lambda)$ have the same sign.

As in Lemma 2, define $r_1(\lambda) = p^{(1)}(\lambda)/q^{(1)}(\lambda)$ and $r_2(\lambda) = p^{(2)}(\lambda)/q^{(2)}(\lambda)$. From the three-term recurrence (2.1) it can be determined that the high-order term $p^{(1)}(\lambda) = d_{i,k}$ is $(-1)^{k-i+1}\lambda^{k-i+1}$, hence the highest-order coefficient is $p^{(1)}_{k-i+1} = (-1)^{k-i+1}$. Similarly, $q^{(1)}_{k-i} = (-1)^{k-i}$, $p^{(2)}_{j-k} = (-1)^{j-k}$, and $q^{(2)}_{j-k-1} = (-1)^{j-k-1}$. Therefore, $\operatorname{sign}[p^{(1)}_{j-i+1}/q^{(1)}_{k-i}] = \operatorname{sign}[p^{(2)}_{j-k}/q^{(2)}_{j-k-1}] = \operatorname{sign}(-1)$. By Lemma 1 this implies that $r_1'(\lambda)$ and $r_2'(\lambda)$ have the same sign on all intervals $(\lambda_{l-1}, \lambda_l)$. This completes the validation of the assumptions in the lemmas, which can now be applied to (3.5) and hence to (2.11a).

By Lemma 4, $d_{i,j} = d(\lambda)$ does not have more than a single zero in any two adjacent intervals, including the case in which the length of one interval is zero. But from the fundamental theorem of algebra, $d_{i,j}$ must have $j-i+1$ zeros on all $2(j-i)+1$ intervals. However, the only possible distribution of these zeros, subject to the restriction of Lemma 4, is one zero per every other interval, beginning with the first interval (λ_0, λ_1) and ending with the last interval $(\lambda_{2(j-i)}, \lambda_{2(j-i)+1})$. This completes the proof of the separation theorem. □

4. The polysection algorithm and its complexity

An overview of the computations are presented in this section together with operation counts and a brief discussion of the communication complexity.

Definition 1. Let $\lambda_l(i:j)$ for $l = 1, \ldots, j-i+1$ be the zeros of the characteristic polynomial $d_{i,j}$.

Definition 2. Let $\Lambda_l(i:j)$ for $l = 1, \ldots, 2(j-i)$ be the collated zeros of $d_{i,k}$, $d_{k+1,j}$, $d_{i,k-1}$, and $d_{k+2,j}$, where $k = (i+j-1)/2$. Also let $\Lambda_0(i:j) = \lambda_0$ and $\Lambda_{2(j-i)+1}(i:j) = \lambda_{2(j-i)+1}$ be the lower and upper bounds, respectively, on the zeros of $d_{i,j}$. The bounds for $d_{i,j}$ can be determined from Gershgorin's theorem. The bounds for $d_{i+1,j}$ and $d_{i,j-1}$ can be determined from the zeros of $d_{i,j}$, and the bounds for $d_{i+1,j-1}$ can be determined from the zeros of either $d_{i+1,j}$ or $d_{i,j-1}$.

For exposition it will be assumed that $N = 2^s$ for some integer s. This requirement simplifies the presentation but is not imposed by the algorithm, since k does not have to be exactly midway between i and j in equations (2.11). The parallel algorithm for computing the eigenvalues of a symmetric tridiagonal matrix is then given by:
For $r = 1, \ldots, s$ and

for $m = 0, \ldots, 2^{s-r} - 1$
set $i = m2^r + 1$; $j = (m+1)2^r$ and compute
$\lambda_l(i:j) \in [\Lambda_{2l-2}(i:j), \Lambda_{2l-1}(i:j)]$ for $l = 1, \ldots, j-i+1$,
$\lambda_l(i:j-1) \in [\Lambda_{2l-2}(i:j-1), \Lambda_{2l-1}(i:j-1)]$ for $l = 1, \ldots, j-i$,
$\lambda_l(i+1:j) \in [\Lambda_{2l-2}(i+1:j), \Lambda_{2l-1}(i+1:j)]$ for $l = 1, \ldots, j-i$,
$\lambda_l(i+1:j-1) \in [\Lambda_{2l-2}(i+1:j-1), \Lambda_{2l-1}(i+1:j-1)]$
for $l = 1, \ldots, j-i-1$.

The four sets of eigenvalues belong to the four characteristic polynomials listed on the left side of equations (2.11a–d). The eigenvalues of $\mathbf{A}$ 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 $\mathbf{A}$ is about four times the amount that would be required to compute the eigenvalues of $\mathbf{A}$ if the intervals were known.

First we will determine the operation count for polysection on a single processor. For $r = 1, \ldots, \log_2 N - 1$, a total of $4N$ zeros of polynomials with degree 2^r must be computed. For $r = \log_2 N$ only N zeros must be computed. From §2, fewer than $10 \cdot 2^r$ operations are required to evaluate a polynomial with degree 2^r. Therefore, the number of operations TF_1 required to compute the eigenvalues of a symmetric tridiagonal matrix on a single processor, using the parallel algorithm for evaluating the polynomials, is bounded by

$$TF_1 < 10n_eN^2 + 40n_eN \sum_{r=1}^{\log_2 N - 1} 2^r < 50n_eN^2, \tag{4.1}$$

where n_e is the maximum number of polynomial evaluations that are required to determine a zero. With N processors each zero can be determined on a separate processor for a bound of

$$TF_N < 50n_eN. \tag{4.2}$$

With N^2 processors a characteristic polynomial with degree 2^r can be computed with $2r + 3$ flops, using the algorithm given in §2. In addition, each

zero can be determined on a separate processor, and hence the total number of operations TF_{N^2} is bounded by

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

Note that the four polynomials in (2.11) could be evaluated independently with additional processors for a minimum of $n_e \log^2 N$ flops.

In the development of these bounds 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 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 $\lambda_i^{(2)}$ can be merged to form the desired collated sequence.

Two sequences can be shuffled in a time proportional to the logarithm of their length. Also, if two ordered sequences are juxtaposed, with the first increasing and the second decreasing, then the combined sequences form a single bitonic sequence that can also be ordered in a time proportional to the logarithm of its length. Hence, for a polynomial of degree 2^r, each of the steps above can be performed with $O(r)$ parallel transmissions, and therefore 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. Implementations of polysection

In this section we will develop both a polynomial and rational function implementation of polysection. The polynomial implementation may require scaling to avoid over/underflow, but it is more accurate than the rational function implementation. The topic of over/underflow is important in the context of computing characteristic polynomials and is discussed further in §6. Over/underflow can also be eliminated by reformulating the method in terms of rational functions, which is the reason for the second implementation given in this section.

The sign of each polynomial at the interval endpoints can be determined a priori and used to eliminate the usual problems associated with finite-precision

arithmetic. This also provides both implementations with guaranteed 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 describe both implementations and certain details that are required to ensure that all eigenvalues are found.

A. **Polynomial implementation.** In this implementation the polynomials are computed using the parallel algorithm that was given in §2. We do not discuss the zerofinding method itself other than to note that the method of interval bisection was used for the results presented in §7. 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 $\mathbf{A}$, the high-order term in $d_{i,j}$ is $(-1)^{j-i+1}\lambda^{j-i+1}$. 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 $\Lambda_l(i:j)$ for $l = 0, \ldots, 2^{r+1}-1$ are $+--++\cdots++--+$. Similarly, the signs of $d_{i,j-1}$ on $\Lambda_l(i:j-1)$ for $l = 0, \ldots, 2^{r+1}-3$ are $+--++\cdots--++-$. The signs of $d_{i+1,j}$ on $\Lambda_l(i+1:j)$ for $l = 0, \ldots, 2^{r+1}-3$ are $+--++\cdots--++-$ and the signs of $d_{i+1,j-1}$ on $\Lambda_l(i+1:j-1)$ for $l = 0, \ldots, 2^{r+1}-5$ are $+--++\cdots++--+$.

In practice there are two reasons why these sign patterns may 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 imply that 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 endpoint value whose sign differs from the correct sign by the machine epsilon or any value 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. Sign tests are attractive since they are machine-independent.

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" was also reported by Dongarra and Sorensen [6] who observed that it could make Cuppen's method competitive with the QR algorithm on a single processor. Deflation could be initiated with a machine-dependent test that detects very small intervals. However, a more satisfactory approach has been to detect interlace faults and set the zeros that have crossed to their average value. This produces zero-length intervals that are detectable without the use of a machine-dependent test. A common matrix, for which deflation does not occur, has elements $b_i = \alpha$ and $c_i = 1$, where α is arbitrary.

B. **Rational function implementation.** The eigenvalues of **A** can also be computed as the zeros of the rational function $R(\lambda)$ that was introduced in Lemma 2, §3, namely

$$R(\lambda) = r_1(\lambda)r_2(\lambda) - c_k^2, \tag{5.1}$$

where $r_1(\lambda) = d_{i,k}/d_{i,k-1}$ and $r_2(\lambda) = d_{k+1,j}/d_{k+2,j}$. All polynomials are evaluated and stored in factored form in this 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 small 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. The accuracy of the two implementations are compared experimentally in §7.

To provide the rational implementation with the same degree of reliability as the polynomial implementation, it is necessary to determine the behavior of $R(\lambda)$ at the interval endpoints α_1, α_2. If the interval contains a zero of (5.1), then $r_1(\lambda)$ and $r_2(\lambda)$ must have the same sign. They also satisfy the conditions of Lemma 1, which implies that they are both monotone decreasing functions, since the ratio of the high-order coefficients is -1. 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 guarantee that all of the zeros will be found.

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 $r_1(\lambda)r_2(\lambda)$ 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 §7 for which the maximum order of any computed polynomial is 15!
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 through 5 apply. Once $R(\alpha_1)$ and $R(\alpha_2)$ are determined from step 4 or step 5, the zeros can be determined using bisection or any variant thereof. Only the signs of $R(\alpha_1)$ and $R(\alpha_2)$ are relevant if bisection is used. Other zerofinding methods can be used but they should probably be combined with bisection if reliability is to be maintained.

Although scaling is not required for this implementation, the rational functions $r_1(\lambda)$ and $r_2(\lambda)$ should be computed as a product of quotients

$(\lambda - \lambda_i)/(\lambda - \beta_i)$, where λ_i and β_i are chosen as close as possible. This prohibits the growth of intermediate computations that might result in numerical difficulties.

6. Parallel bisection

As early as 1962, Wilkinson [20] presented bisection as a method for computing the eigenvalues of a symmetric tridiagonal matrix. In 1967, Barth, Martin, and Wilkinson [1] published an improved algorithm that was resistant to overflow and made more efficient use of the Sturm sequences. Information obtained from the Sturm sequences for one zero was used to speed finding the other zeros. Evans et al. [7] also make efficient use of the Sturm sequences to speed the bisection process. Schreiber [14] uses bisection in defining systolic arrays for computing the eigenvalues. The accuracy of serial bisection is quite good, as determined by Kahan [9] in 1966. Recently, Demmel and Kahan [4] and Demmel and Gragg [5] provided additional results concerning the accuracy of singular values and eigenvalues.

Both Kahan and Wilkinson recommended a variant of the three-term recurrence (2.1), namely

$$q_j = b_j - \lambda - c_{j-1}^2/q_{j-1}, \tag{6.1}$$

where $q_j = d_{i,j}/d_{i,j-1}$. The purpose of this variant is to avoid over/underflow. The method of bisection is based on the following theorem:

The number of agreements in the sign of consecutive characteristic polynomials $d_{1,j}(\lambda)$ of leading principal submatrices is equal to the number of eigenvalues strictly greater than λ.

The Sturm sequence includes $d_{1,0} = 1$. In [20], Wilkinson noted "The theorem may be used to locate an individual eigenvalue without locating any of the others" which forms the basis of an elegant parallel algorithm. The pth processor uses bisection to find the largest value of λ such that $d_{1,j}(\lambda)$ has p agreements in sign; i.e., the pth processor determines the pth eigenvalue independent of any other processor.

With the use of the sequential recurrence relations (2.1) or (6.1) together with N processors, this approach requires $O(N)$ time. On a single processor $O(N^2)$ time is required. In this paper we will use a parallel algorithm for computing the Sturm sequence $d_{1,j}(\lambda)$. The algorithm will be reviewed only briefly here since the details can be found in [19]. It is a variant of a similar algorithm given in [13]. Both Sturm sequence calculations require the computation of the partial products of a sequence of two-by-two matrices. The parallel algorithm used for these partial products has been generalized to any associative operator in [11], where it is called the parallel prefix computation.

In §2 we reviewed the parallel algorithm for computing the characteristic polynomial of $\mathbf{A}$, namely $d_{1,N}(\lambda)$. As a by-product of this computation certain other elements of the Sturm sequence were computed, namely $d_{1,2^l}(\lambda)$. The remaining elements in the Sturm sequence can be "filled-in", using a parallel algorithm with the same complexity $O(\log N)$ as the parallel algorithm in §2. For the case $N = 8$ the parallel algorithm in §2 computed the elements $d_{1,1}(\lambda)$, $d_{1,2}(\lambda)$, $d_{1,4}(\lambda)$, and $d_{1,8}(\lambda)$ in the Sturm sequence. The remaining $d_{1,j}$ can be determined as follows:

Step 1.

$$\mathbf{Q}_{1,6} = \mathbf{Q}_{1,4}\mathbf{Q}_{5,6}.$$

Step 2.

$$\mathbf{Q}_{1,3} = \mathbf{Q}_{1,2}\mathbf{Q}_{2,3},$$
$$\mathbf{Q}_{1,5} = \mathbf{Q}_{1,4}\mathbf{Q}_{5,5},$$
$$\mathbf{Q}_{1,7} = \mathbf{Q}_{1,6}\mathbf{Q}_{7,7}.$$

From §2, $10N$ flops are required to evaluate the characteristic polynomial of $\mathbf{A}$. Asymptotically, an equal number of flops are required to "fill-in" the Sturm sequence. Therefore, if n_e is the maximum number of Sturm sequences required to compute a zero, then the total number of operations required to compute N zeros using parallel bisection on a single processor is less than $20n_eN^2$ flops. This compares with a bound of $50n_eN^2$ flops for computing the eigenvalues using polysection. The operation counts are compared in Table 0.

TABLE 0. Asymptotic operation counts for bisection and polysection

processors	bisection	polysection
1	$20n_eN^2$	$50n_eN^2$
N	$20n_eN$	$50n_eN$
N^2	$4n_e \log_2 N$	$4n_e \log_2^2 N$

If the two Sturm sequence "fill-in" steps given above are combined with the three steps in §2, then the total number of parallel steps is five. At the expense of additional sequential computations, the number of parallel computations are reduced to three in the following parallel algorithm that computes all elements in the Sturm sequence in three parallel steps. In an early paper, Stone [17] observed this tradeoff between sequential and parallel computations.

Step 1. Compute

$$\mathbf{Q}_{1,2} = \mathbf{Q}_{1,1}\mathbf{Q}_{2,2}, \quad \mathbf{Q}_{2,3} = \mathbf{Q}_{2,2}\mathbf{Q}_{3,3},$$
$$\mathbf{Q}_{3,4} = \mathbf{Q}_{3,3}\mathbf{Q}_{4,4}, \quad \mathbf{Q}_{4,5} = \mathbf{Q}_{4,4}\mathbf{Q}_{5,5},$$
$$\mathbf{Q}_{5,6} = \mathbf{Q}_{5,5}\mathbf{Q}_{6,6}, \quad \mathbf{Q}_{6,7} = \mathbf{Q}_{6,6}\mathbf{Q}_{7,7},$$
$$\mathbf{Q}_{7,8} = \mathbf{Q}_{7,7}\mathbf{Q}_{8,8}.$$

Step 2. Compute

$$\mathbf{Q}_{1,3} = \mathbf{Q}_{1,1}\mathbf{Q}_{2,3}, \quad \mathbf{Q}_{1,4} = \mathbf{Q}_{1,2}\mathbf{Q}_{3,4},$$
$$\mathbf{Q}_{2,5} = \mathbf{Q}_{2,3}\mathbf{Q}_{4,5}, \quad \mathbf{Q}_{3,6} = \mathbf{Q}_{3,4}\mathbf{Q}_{5,6},$$
$$\mathbf{Q}_{4,7} = \mathbf{Q}_{4,5}\mathbf{Q}_{6,7}, \quad \mathbf{Q}_{5,8} = \mathbf{Q}_{5,6}\mathbf{Q}_{7,8}.$$

Step 3. Compute

$$\mathbf{Q}_{1,5} = \mathbf{Q}_{1,1}\mathbf{Q}_{2,5}, \quad \mathbf{Q}_{1,6} = \mathbf{Q}_{1,2}\mathbf{Q}_{3,6},$$
$$\mathbf{Q}_{1,7} = \mathbf{Q}_{1,3}\mathbf{Q}_{4,7}, \quad \mathbf{Q}_{1,8} = \mathbf{Q}_{1,4}\mathbf{Q}_{5,8}.$$

This algorithm requires about half the time on a parallel computer but twice the number of processors required by the previous algorithm. In general, $O(N \log N)$ operations are required on a single processor compared with $O(N)$ for the previous algorithm.

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 [19]. Scaling is not required at each level r, which reduces the time for scaling. For example, if scaling is performed for $r = 5, 10, 15, \ldots$ about $N/32$ matrices are scaled. Scaling time would be negligible if it were performed in machine code. If "power-of-two" scaling is used, then accuracy is unchanged. More specifically, over/underflow can be avoided by an occasional scaling of the 2×2 matrices in the following manner: First compute

$$\text{qmax} = \|\mathbf{Q}_{i,j}\|_\infty ; \tag{6.2}$$

next compute

$$l = [\log_2 \text{qmax}]; \tag{6.3}$$

then $\mathbf{Q}_{i,j}$ is replaced by

$$\hat{\mathbf{Q}}_{i,j} = 2^{-l}\mathbf{Q}_{i,j}. \tag{6.4}$$

A rounded value of l was used for the computational results in §7.

7. Computational results

In this section we compare the accuracy of: (a) the QR algorithm as implemented in subroutine TQL1 in EISPACK [15]; (b) parallel bisection; (c) polynomial polysection; and (d) rational function polysection. As mentioned in §6, the accuracy of serial bisection has been demonstrated to be high. Although the accuracy of both parallel bisection and polysection is yet to be determined, the computational results in this section imply that they are both quite accurate. The entries in the tables below are computed from

$$e = \frac{\max_i |\lambda_i - \lambda_i^*|}{\max_i |\lambda_i^*|}, \tag{7.1}$$

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 Sun SPARCstation 2.

Three tables are presented that correspond to three different matrices. Table 1 contains a comparison of accuracy for a matrix with random coefficients. Table 2 compares accuracies for the matrix W_{2k}^+ which is a slight variant of W_{2k-1}^+ in [21, 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 3 (see p. 666) 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. In general, the accuracy of the QR algorithm is inferior to the accuracy of all the other methods with the exception of the rational function implementation for $N \leq 1024$ in Table 3.
2. The accuracy of QR is noticeably inferior in the presence of multiple zeros, as demonstrated in Table 2.

TABLE 1. Accuracy of the QR method and parallel algorithms for a symmetric tridiagonal matrix with random coefficients

N	QR	PB	PI	RI
128	4.72×10^{-6}	6.63×10^{-8}	5.30×10^{-8}	1.79×10^{-6}
256	5.09×10^{-6}	7.43×10^{-7}	5.84×10^{-7}	5.25×10^{-6}
512	9.96×10^{-6}	2.93×10^{-7}	1.17×10^{-7}	2.99×10^{-6}
1024	1.64×10^{-5}	3.88×10^{-7}	6.65×10^{-7}	6.04×10^{-6}
2048	1.81×10^{-5}	2.67×10^{-6}	3.02×10^{-6}	7.82×10^{-6}
4096	2.12×10^{-5}	1.15×10^{-5}	3.60×10^{-5}	2.55×10^{-5}

QR as implemented in subroutine TQL1 from EISPACK
PB parallel bisection
PI polynomial implementation of polysection
RI rational function implementation of polysection

TABLE 2. Accuracy of the QR method and parallel algorithms for a symmetric tridiagonal matrix W_{2k}^{+} with near multiple eigenvalues

N	QR	PB	PI	RI
128	3.77×10^{-6}	1.18×10^{-7}	1.18×10^{-7}	1.18×10^{-7}
256	8.06×10^{-6}	1.19×10^{-7}	1.19×10^{-7}	1.19×10^{-7}
512	2.30×10^{-5}	1.19×10^{-7}	1.19×10^{-7}	1.19×10^{-7}
1024	4.46×10^{-5}	5.95×10^{-8}	5.95×10^{-8}	2.38×10^{-7}
2048	8.14×10^{-5}	1.19×10^{-7}	1.19×10^{-7}	1.19×10^{-7}
4096	1.30×10^{-4}	5.95×10^{-8}	5.95×10^{-8}	1.19×10^{-7}

QR as implemented in subroutine TQL1 from EISPACK
PB parallel bisection
PI polynomial implementation of polysection
RI rational function implementation of polysection

3. In general, the accuracy of the polynomial implementation of polysection is superior to the rational function implementation. Nevertheless, the difference is relatively small and the errors are comparable in the presence of extensive deflation, as shown in Table 2.
4. The accuracy of polynomial polysection and bisection are comparable and in general superior to the other methods.

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 15! Similarly, the maximum degree of any computed rational polynomial for Table 1 is 42. The eigenvalues of the matrix in Table 3 are

TABLE 3. Accuracy of the QR method and parallel algorithms for a symmetric tridiagonal matrix with zero diagonal and 1's off diagonal

N	QR	PB	PI	RI
128	1.49×10^{-6}	1.79×10^{-7}	1.79×10^{-7}	8.65×10^{-6}
256	1.91×10^{-6}	3.58×10^{-7}	1.79×10^{-7}	2.95×10^{-5}
512	4.41×10^{-6}	4.17×10^{-7}	7.75×10^{-7}	2.93×10^{-5}
1024	9.06×10^{-6}	1.98×10^{-6}	6.56×10^{-7}	4.23×10^{-5}
2048	8.26×10^{-5}	2.92×10^{-6}	1.97×10^{-6}	7.26×10^{-5}
4096	2.59×10^{-4}	2.98×10^{-6}	3.21×10^{-6}	8.93×10^{-5}

QR as implemented in subroutine TQL1 from EISPACK
PB parallel bisection
PI polynomial implementation of polysection
RI rational function implementation of polysection

$\lambda_i = 2\cos i\pi/(N+1)$, 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 15 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 Table 2.

The extent of the deflation is quite significant, even for random matrices, and leads one to question its origin. We conjecture that, for most matrices, the eigenvalues may be somewhat loosely coupled to most coefficients; i.e., they may be determined to considerable accuracy as the eigenvalues of some submatrix and not influenced to any significant extent by coefficients outside the submatrix. Since the eigenvalues in polysection are determined from submatrices whose order increases, it is possible for the eigenvalues to converge rapidly and essentially remain unchanged thereafter, which results in deflation. The limiting case is the diagonal matrix in which the eigenvalues are determined by a 1-by-1 matrix and remain independent of *any* other coefficients. This may be the main reason for deflation, although multiple eigenvalues also seem to contribute to deflation. These observations remain to be confirmed with theory.

The results in Table 2 show that near multiple zeros do not present any difficulties for the parallel methods. Indeed, the amount of computation may be reduced. Recall that polysection 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 zerofinding 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, since the intervals coalesce, any point on the interval is close to the multiple zero and therefore a good initial approximation to the zero.

8. SUMMARY AND CONCLUSIONS

The polysection algorithm was 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 QR, and the rational function implementation has comparable accuracy with deflation. The rational function implementation 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.

The performance of polysection 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 Newton's method, bisection, and the secant method together with variants and combinations. Zerofinding is itself a significant area of computational mathematics. Dongarra and Sorensen [6] use a variant of Newton's method in which a local quadratic rational approximation to the function is computed. The zero-in method [12] combines bisection and the secant method. Whatever method is chosen, it should probably be combined with bisection to ensure that all zeros are found.

(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 results in Table 2, using the rational function implementation, was proportional to N rather than N^2. However, deflation does not occur for the matrix that corresponds to Table 3, which supports the observation 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.

Polysection was compared to parallel bisection, which was found to have comparable accuracy and lower asymptotic operation count. However, as mentioned in the introduction, the reliability and the ability to speed zerofinding through the use of high-order methods and deflation may make polysection the algorithm of choice for certain application.

Acknowledgments

The author wishes to acknowledge a helpful conversation with Bill Gragg who suggested the use of the partial fraction expansion in §3, Lemma 1. The author also wishes to thank Rob Schreiber for directing attention to parallel bisection,

following a lecture on polysection at RIACS in August 1989. Thanks are also due the reviewers who provided many excellent contributions and particularly to Jim Demmel whose extensive comments on parallel bisection resulted in §6.

BIBLIOGRAPHY

1. W. Barth, R. S. Martin, and J. H. Wilkinson, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection*, Numer. Math. **9** (1967), 386–393.
2. J. R. Bunch, C. P. Nielsen, and D. C. Sorensen, *Rank-one modification of the symmetric eigenproblem*, Numer. Math. **31** (1978), 31–48.
3. J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math. **36** (1981), 177–195.
4. J. Demmel and W. Kahan, *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Statist. Comput. **11** (1990), 873–912.
5. J. Demmel and W. Gragg, *On computing accurate singular values and eigenvalues of acyclic matrices*, IMA Preprint Series, no. 962, Institute for Math. and its Appl., Univ. of Minnesota, 1992.
6. J. J. Dongarra and D. C. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput. **8** (1987), s139–s154.
7. D. J. Evans, J. Shanehchi, and C. C. Rick, *A modified bisection algorithm for the determination of the eigenvalues of a symmetric tridiagonal matrix*, Numer. Math. **38** (1982), 417–419.
8. I. C. F. Ipsen and E. R. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, SIAM J. Sci. Statist. Comput. **11** (1990), 203–229.
9. W. Kahan, *Accurate eigenvalues of a symmetric tri-diagonal matrix*, Tech. Report CS41, Comput. Sci. Dept., Stanford University, July 22, 1966, with revisions to June 1968, available from the author at the University of California at Berkeley.
10. A. S. Krishnakumar and M. Morf, *Eigenvalues of a symmetric tridiagonal matrix: a divide and conquer approach*, Numer. Math. **48** (1986), 349–368.
11. R. E. Ladner and M. J. Fisher, *Parallel prefix computation*, J. Assoc. Comput. Mach. **27** (1980), pp. 831–838.
12. S.-S. Lo, B. Philippe, and A. H. Sameh, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Statist. Comput. **8** (1987), s155–s165.
13. A. H. Sameh and D. J. Kuch, *A parallel QR algorithm for symmetric tridiagonal matrices*, IEEE Trans. Comput. **26** (1977), 147–153.
14. R. Schreiber, *Computing generalized inverses and eigenvalues of symmetric matrices using systolic arrays*, Computing Methods in Applied Sciences and Engineering VI (R. Glowinski and J.-L. Lions, eds.), North-Holland, Amsterdam, 1984, pp. 285–295.
15. B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix eigensystem routines—EISPACK Guide*, 2nd ed., Lecture Notes in Comput. Sci., vol. 6, Springer-Verlag, New York, 1976.
16. D. C. Sorensen and P. T. P. Tang, *On the orthogonality of eigenvectors computed by divide-and-conquer techniques*, SIAM J. Numer. Anal. **28** (1991), 1172–1775.
17. H. S. Stone, *Parallel tridiagonal solvers*, ACM Trans. Math. Software **1** (1975), 289–307.
18. P. N. Swarztrauber, *A direct method for the discrete solution of separable elliptic equations*, SIAM J. Numer. Anal. **11** (1974), 1136–1150.
19. ——, *A parallel algorithm for solving general tridiagonal equations*, Math. Comp. **33** (1979), 185–199.
20. J. H. Wilkinson, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection*, Numer. Math. **4** (1962), 362–367.
21. ——, *The algebraic eigenvalue problem*, Clarendon Press, Oxford, 1965.

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH, P. O. BOX 3000, BOULDER, COLORADO 80303
E-mail address: pauls@ncar.ucar.edu