SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 2, March 1987

A MULTIPROCESSOR ALGORITHM FOR THE SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEM*

SY-SHIN LO†, BERNARD PHILIPPE† AND AHMED SAMEH†

Abstract. A multiprocessor algorithm for finding few or all eigenvalues and the corresponding eigenvectors of a symmetric tridiagonal matrix is presented. It is a pipelined variation of EISPACK routines—BISECT and TINVIT which consists of the three steps: isolation, extraction-inverse iteration, and partial orthogonalization. Multisections are performed for isolating the eigenvalues in a given interval, while bisection or the Zeroin method is used to extract these isolated eigenvalues. After the corresponding eigenvectors have been computed by inverse iteration, the modified Gram–Schmidt method is used to orthogonalize certain groups of these vectors. Experiments on the Alliant FX/8 and CRAY X-MP/48 multiprocessors show that this algorithm achieves high speed-up over BISECT and TINVIT; in fact it is much faster than TQL2 when all the eigenvalues and eigenvectors are required.

Key words. eigenvalues, multiprocessors, tridiagonal matrices

AMS(MOS) subject classification. 65F15

1. Introduction.

This paper deals with solving the real symmetric tridiagonal eigenvalue problem on a multiprocessor. The main purpose of this study is solving for a few of the eigenvalues and the corresponding eigenvectors of large tridiagonal symmetric matrices such as those resulting from the Lanczos tridiagonalization of a sparse symmetric matrix. In fact, we show that our scheme, the origins of which date back to the Illiac IV [8] and [7], is equally or more effective than other multiprocessor schemes for obtaining either all the eigenvalues, or all the eigenvalues and eigenvectors of a symmetric tridiagonal matrix.

Two kinds of methods are usually used for solving this problem on a uniprocessor. When only a part of the spectrum is desired, the combination of bisection and inverse iteration is the method of choice. For the whole eigenvalue problem, the QR (or QL) method is more effective [2]. These methods are implemented in EISPACK: BISECT and TINVIT for the partial eigenvalue problem, and TQL1 (eigenvalues only) and TQL2 (eigenvalues and eigenvectors) for the whole problem. A multiprocessor version of TQL2 has already been designed [4] using a divide and conquer technique.

In § 2 we give a brief description of the method, in § 3 we analyze the various steps of the computation with respect to efficient use of the parallelism offered by the multiprocessor. In § 4, we compare our scheme with all of the above schemes on the Alliant FX/8 and the CRAY X-MP/48 multiprocessors.

2. Description of the method.

Let $\mathbf{T}$ be a symmetric tridiagonal matrix of order n with d_i and e_i as the diagonal and subdiagonal elements, respectively,

$$\mathbf{T} = [e_i, d_i, e_{i+1}].$$

Let $p_n(\lambda)$ be its characteristic polynomial:

$$p_n(\lambda) = \det(\mathbf{T} - \lambda \mathbf{I}).$$

* Received by the editors December 23, 1985; accepted for publication (in revised form) May 12, 1986. This work was supported in part by the National Science Foundation under Grants US NSF DCR84-10110 and US NSF DCR85-09970, the U.S. Department of Energy under Grant US DOE DE-FG02-85ER25001, the IBM Donation, and the French Ministry of Defense under Grant DRET No. 84-823. This paper was submitted to the Second SIAM Conference on Parallel Processing for Scientific Computing that was held in Norfolk, Virginia during November 18-21, 1985.

† Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801.

The sequence of the principal minors of the matrix can be built using the following recursion:

$$
\begin{aligned}
&p_0(\lambda)=1,\\
(2.1)\qquad &p_1(\lambda)=d_1-\lambda,\\
&p_i(\lambda)=(d_i-\lambda)p_{i-1}(\lambda)-e_i^2p_{i-2}(\lambda),\qquad i=2,\cdots,n.
\end{aligned}
$$

We assume that no subdiagonal element is zero, since if some e_i is equal to 0, the problem can be partitioned into two smaller problems. The sequence $\{p_i(\lambda)\}$ is called the Sturm sequence of $\mathbf{T}$ in λ. It is well known [11] that the number of eigenvalues smaller than a given λ is equal to the number of sign variations in the Sturm sequence (2.1). Hence one can find the number of eigenvalues lying in a given interval [a, b] by computing the Sturm sequences at a and b. The linear recurrence (2.1), however, suffers from the possibility of over- or underflow. This is remedied by replacing the Sturm sequence $p_i(\lambda)$ by the sequence

$$q_i(\lambda)=\frac{p_i(\lambda)}{p_{i-1}(\lambda)},\qquad i=1,n.$$

The second order linear recurrence (2.1) is then replaced by the nonlinear recurrence

$$(2.2)\qquad q_1(\lambda)=d_1-\lambda,\quad q_i(\lambda)=d_i-\lambda-\frac{e_i^2}{q_{i-1}(\lambda)},\quad i=2,\cdots,n.$$

Here, the number of eigenvalues that are smaller than λ is equal to the number of negative terms in the sequence $\{q_i(\lambda)\}$.

Therefore, given an initial interval, we can find the eigenvalues lying in it by repeated bisection or multisection of the interval. This partitioning process can be performed until we obtain each eigenvalue to a given accuracy. On the other hand, we can stop the process once we have isolated each eigenvalue. In the latter case the eigenvalues may be extracted using a faster method. Several methods are available for extracting an isolated eigenvalue:

- Bisection (linear convergence);
- Newton's method (quadratic convergence);
- The Zeroin scheme, which is based on the secant and bisection methods (convergence of order $(\sqrt{5}+1)/2$).

Ostrowski [10] defines an efficiency index which links the amount of computation to be done at each step and the order of the convergence. The respective indices of the three methods are 1, 1.414 and 1.618. This index, however, is not the only aspect to be considered here. Both Zeroin and Newton methods require the use of the linear recurrence (2.1) in order to obtain the value of $\det(\mathbf{T}-\lambda\mathbf{I})$, or its derivative as well, for a given λ. Hence, if the possibility of over- or underflow is small we select the Zeroin method; otherwise we select the bisection method. After the computation of an eigenvalue, the corresponding eigenvector can be found by inverse iteration [6]. This is a very fast process where one iteration is often sufficient to achieve convergence.

It is possible that some eigenvalues are computationally coincident; hence, the isolation process actually performs "isolation of clusters," where a cluster is defined as a single eigenvalue or a number of computationally coincident eigenvalues. If such a cluster of coincident eigenvalues is isolated the extraction step is skipped, since convergence has been reached. Observing that, there can be loss of orthogonality for those eigenvectors corresponding to close eigenvalues; orthonormalization of such eigenvectors via the Modified Gram-Schmidt method is necessary.

In summary, the whole computation consists of the following five steps:

1) Isolation by partitioning;
2) Extraction of a cluster by bisection or by the Zeroin method;
3) Computation of the eigenvectors of the cluster by inverse iteration;
4) Grouping of close eigenvalues;
5) Orthogonalization of the corresponding groups of vectors by the Modified Gram-Schmidt process.

3. The parallel algorithm.

3.1. The partitioning process. Obviously, the parallelism in this process is achieved by performing simultaneously the computation of several Sturm sequences. However, there are several ways for achieving this; two options are

-Performing bisection on several intervals;

-Performing a partition of one interval into several subintervals.

A multisection of order k splits the interval $I=[a, b]$ into $k+1$ subintervals $I_i=[x_i, x_{i+1}]$, where $x_i=a+i((b-a)/(k+1))$ for $i=0, \cdots, k+1$. If there exists only one eigenvalue in the interval I and if we wish to compute it with an absolute error ε, then it is necessary to perform

$$n_k=\log_2\left(\frac{(b-a)}{(2\varepsilon)}\right)\Big/\log_2(k+1)$$

multisections of order k. Thus, the efficiency of the multisection of order k compared to bisection (multisection of order 1) is

$$E_f=n_1/(k\,n_k)=(\log_2(k+1))/k.$$

Hence, for extraction of eigenvalues, we prefer to perform parallel bisections rather than one multisection of high order. On the other hand, during the isolation step, the efficiency of multisectioning is higher because: (i) a multisection creates more tasks than bisection, and (ii) there are several eigenvalues in one interval. The way we propose to use bisections or multisections is almost the same as that stated in [1].

3.2. The computation of the Sturm sequence. The recurrence (2.2) is intrinsically serial, so parallelism is not possible in this computation. The algorithm in [3] may be used, however, to vectorize the linear recurrence (2.1). Here, the computation of the regular Sturm sequence (2.1) is equivalent to solving a lower-triangular system of order $n+1$ consisting of three diagonals,

$$\begin{bmatrix} 1 & & & & & \\ -a_1 & 1 & & & 0 & \\ -b_2 & -a_2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & 0 & & -b_n & -a_n & 1 \end{bmatrix},$$

where $a_i=d_i-\lambda$, and $b_i=-e_i^2$. For a vector length $k<n/2$ the total number of arithmetic operations in the parallel algorithm is roughly $10n+11k$ while it is only $4n$ for the serial algorithm (we do not consider the operation that computes e_i^2, since this quantity can be provided by the user); resulting into an arithmetic redundancy which varies between 2.5 and 4.

This algorithm, therefore, is efficient only when vector operations are at least 4 times faster than sequential operations, which excludes the FX/8. The results on one processor of a CRAY X-MP are displayed in Fig. 1. In this figure, three methods are compared:

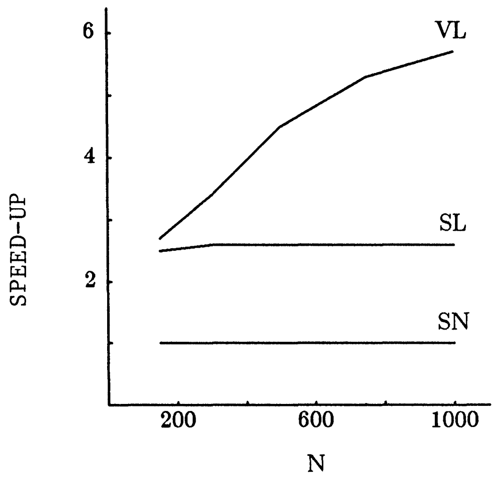

FIG. 1. *Computation of Sturm sequence on* CRAY X-MP, *speed-up over* SN. *N is the order of test matrices of* $[-1, 2, -1]$.

- Sequential computation of the linear recurrence (2.1) (SL);
- Vector computation of the linear recurrence (2.1) (VL);
- Sequential computation of the nonlinear recurrence (2.2) (SN).

Using 64 bit arithmetic on the test matrix $[-1, 2, -1]$, we have evaluated elements of the Sturm sequence (2.1) with no over- or underflow. The (SL) method is always faster than the (SN) method. The method (VL) (coded in FORTRAN) can reach a speed-up of 2.1 over the (SL) method and of 5.7 over the (SN) method.

3.3. Computation of the eigenvectors and orthonormalization. The computation of an eigenvector can be started as soon as the corresponding eigenvalue is computed. So, we consider the extraction of an eigenvalue and the computation of its eigenvector as two parts of the same task, with the order of potential parallelism being dependent on the number of desired eigenvalues.

Orthonormalization is only performed on eigenvectors whose corresponding eigenvalues meet a predefined grouping criterion. The modified Gram–Schmidt method is used to orthonormalize each group. The algorithm is as follows:

```
do $k = 1, p$
    normalize $(z_k)$
    do $j = k + 1, p$
        $z_j = z_j - (z_j \cdot z_k) z_k$
    od
od
```

where z_j and z_k are vectors, and $(z_j \cdot z_k)$ denotes their inner product. The statement in the inner loop is a vector instruction which updates the vector z_j with respect to the base vector z_k. Considering the dependency of the variables, we see that the inner loop is a **doall** loop, since all the iterations j are independent of one another for a particular k. On the other hand, the outer loop is a **doacross** loop, since any iteration k cannot start before vector z_k has been updated with respect to vectors $z_1, \cdots, z_{k-1}$ and as soon as it has been updated it is ready to be used as base vector for $z_{k+1}, \cdots, z_p$. These dependency relations are shown in Fig. 2.

Two schemes have been designed to perform the above process. One is a doall-inner-sequential-outer algorithm (called PS); the second, with some synchronization

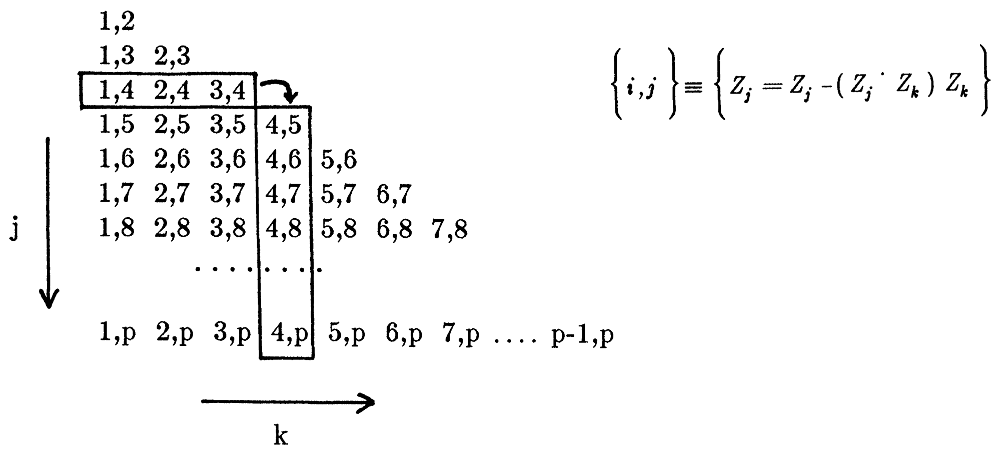

FIG. 2. *Diagram of dependencies.*

mechanisms, is a doall-inner-pipelined-outer algorithm (called PP). Because of the synchronization overhead, the second method is to be preferred only when the order of the matrix and the size of the group are large. Table 1 shows a comparison of the two algorithms on the CRAY X-MP/48. The efficiency of the pipelined method in this case is the consequence of the low overhead of the synchronization mechanisms on the CRAY multiprocessor.

TABLE 1
Two implementations for the modified Gram-Schmidt Method.

	1 processor	2 processors	4 processors
Pipelined outer loop (PP)	26.13	13.61	8.20
Sequential outer loop (PS)	23.18	15.60	11.97
Speed-Up (PS/PP)	.89	1.15	1.46

Test performed on the CRAY X-MP for a matrix of order 1000.

If we consider processing several groups of vectors in parallel, then we have three levels of parallelism:

1) Concurrent processing of the groups;
2) Concurrent updating of vectors;
3) Vectorizing the updating step of each vector.

It is necessary to select the algorithm depending on the levels of parallelism in the architecture. For example, to orthonormalize large groups of eigenvectors, Level 2 and Level 3 can be adopted on multiprocessors like the CRAY X-MP/48 and the Alliant FX/8; whereas all three levels of parallelism can be adopted on a hierarchical structured system such as the Cedar machine [9], which contains several "clusters" of multiprocessors, each processor having vector capability.

4. Implementation and experiments.

4.1. Implementation. In this section, we describe the specific implementations of our algorithm TREPS on the multiprocessors Alliant FX/8 (consisting of 8 processors) and the CRAY X-MP/48 (consisting of 4 processors). The implementation on the Alliant is slightly different from the implementation on the CRAY.

During the orthogonalization process on the CRAY, the groups of eigenvectors to be orthonormalized are split into two classes depending on their sizes: the small groups are processed, one per processor, in parallel; and the large groups are processed one group at a time, using all the processors with the parallel-inner pipelined-outer loop algorithm. With 4 processors, a group of more than 10 eigenvalues is regarded as large.

For both implementations, the repartitioning of the intervals containing an isolated cluster is dynamic. These intervals are stored in a stack. As soon as one processor is ready, it fetches a new task (i.e., a new subinterval) from the stack. The updating of the pointer to the top of the stack is protected as a critical section by locks. It has to be pointed out that multitasking on the Alliant FX/8 is easier and less costly than that on the CRAY X-MP/48, since on the former system the creation of tasks is automatic. Moreover, on the CRAY we have used the usual trick, which consists of declaring as many tasks as there are processors, and making them active or idle depending on the number of processes involved so as to reduce the cost of task creation. This manipulation adds some complexity to the code, but it is the price to be paid if one is to obtain good performance on the CRAY X-MP/48.

4.2. Performance. To analyze the speed of this program, we start by comparing TREPS1, a version of the program which performs the extraction stage via bisection, with the EISPACK subroutines BISECT and TINVIT on the sequential machine VAX 785. For several runs with different tridiagonal matrices, we found that, even on a VAX 785, TREPS1 is no more than 20% slower than BISECT+TINVIT. In Table 2 we show such an experiment for the matrix $[-1, 2, -1]$ (unless otherwise stated, all test matrices are of this form). On the FX/8, with some synchronization directives, TREPS ran in vector-concurrent mode, with high speed-ups when the number of desired eigenvalues exceeds the number of processors. All experiments on the Alliant FX/8 are performed in double precision. In Table 3, we show the performance of TREPS1 when *all* the eigenvalues and vectors are obtained on the FX/8. In Table 4,

TABLE 2
Timings for TREPS *and* BISECT+TINVIT *Subroutines on* VAX 785.

N		TREPS1	BISC+TINV
100	Time (seconds)	4.03	3.6
	Ratio	1.11	1
300	Time (seconds)	34.13	31.58
	Ratio	1.08	1

Computations are in single precision. N is the order of the test matrix.

TABLE 3
Time and speed-up for TREPS1 *on* Alliant FX/8.

CE's	Time (second)	Speed-Up
1	115.59	1.0
2	57.12	2.0
4	28.89	4.0
8	14.78	7.8

Test matrices are of order 500.

TABLE 4
Speed-up for TREPS1 *as function of number of eigenvalues and vectors.*

p	Time in seconds		Speed-Up
	1 CE	8 CEs	
1	.15	.15	1
2	.25	.12	2.1
4	.51	.14	3.6
8	1.03	.14	7.4
10	1.29	.26	5.0
20	2.59	.42	6.2
50	6.67	.97	6.9
100	13.33	1.86	7.2
200	27.09	3.74	7.2
300	40.85	5.46	7.5

Test matrix $[-1, 2, -1]$ is of order 300. p is the number of desired eigenvalues and eigenvectors.

TABLE 5
Time for steps in TREPS1 *and* TREPS2.

N		Total	Isolation	Extr+Inv.	Orthonorm.	
500	TREPS1	14.78	.62	13.78	.38	(sec)
		100	4.2	93.2	2.6	(%)
	TREPS2	3.71	.62	2.71	.38	(sec)
		100	4.9	79.8	15.3	(%)
1000	TREPS1	67.68	3.35	54.0	10.33	(sec)
		100	16.7	73.1	10.2	(%)
	TREPS2	24.47	3.39	11.02	10.06	(sec)
		100	13.9	45.0	41.1	(%)

Computing all the eigenvalues and eigenvectors. Computations are done on Alliant FX/8 in double precision.

we measure the speed-up of TREPS1 on the FX/8 as a function of the desired p smallest eigenvalues for a matrix of order 300. On one processor, the time to compute p eigenvalues is proportional to p (in this situation the multisections become bisections, and our program is essentially equivalent to BISECT+TINVIT). On 8 processors, the speed-up over one processor increases from 1 to 5.0 when the number of desired eigenvalues and vectors varies from 1 to 10. It increases from 6.9 (for 30 eigenvalues) to 7.5 (for all of the 300 eigenvalues). Similar results are obtained with the version TREPS2, in which we use the Zeroin method (see [5]) for extraction of the eigenvalues.

Let us now look at the percentage of the computational time involved in each stage. We consider the time for grouping close eigenvalues as negligible. The step for extracting the eigenvalues and computing the eigenvectors is more time-consuming than the isolation process for TREPS1. The orthonormalization time depends heavily on the problem; in the extreme case when orthogonalization of all n vectors is necessary, the elapsed time can reach up to 50% of the whole time needed to solve the problem. Some examples are given in Table 5.

4.3. Comparison with other subroutines. In this section we compare the performance of our algorithm with BISECT+TINVIT, TQL2, and that in [4] when *all* the eigenvalues and eigenvectors are required. We also compare our algorithm with BISECT and TQL1 when only the eigenvalues are needed. To evaluate the numerical performance, we compare the norm of the residuals, $\max_i \|Tz_i - \lambda_i z_i\|_2$, for the computed eigenvalues and eigenvectors for TREPS, BISECT+TINVIT, TQL2, and SESUPD [4]. Orthogonality of the eigenvectors is also checked by computing the $\max_{i,j} |Z^T Z - I|_{i,j}$, where $Z = [z_1, \cdots, z_n]$. Table 6 shows the results on the FX/8, machine precision of 1.11×10^{-16}, for the test matrix $[-1, 2, -1]$ of order 500. We see that for the above test matrix both the residuals and the quality of the eigenvectors and orthogonality of

TABLE 6
Residual and orthonormality of computed eigenvalues and eigenvectors.

	$\max_i \|Tz_i - \lambda_i z_i\|_2$	$\max_{i,j} \|Z^T Z - I\|_{i,j}$
TREPS1	1.00×10^{-12}	1.69×10^{-12}
TREPS2	4.12×10^{-13}	1.31×10^{-12}
TQL2	3.88×10^{-14}	2.66×10^{-14}
BISECT+TINVIT	9.75×10^{-13}	2.77×10^{-12}
SESUPD	5.87×10^{-15}	6.61×10^{-15}

Test matrices are of order 500.

TABLE 7a
Time and speed-up for computing all the eigenvalues and eigenvectors.

		Alliant		CRAY X-MP	
		1 CE	8 CE	1 CPU	4 CPU
TREPS1	time (sec)	115.6	14.8	11.06	3.04
	speed-up	1	7.8	1	3.6
TREPS2	time (sec)	25.7	3.7	1.72	.64
	speed-up	1	6.9	1	2.7
TQL2	time (sec)	486.4	103.1	6.68	—
	speed-up	1	4.7	1	—
BISECT+TINVIT	time (sec)	140.6	136.2	12.98**	—
	speed-up	1	1.0	1	—
SESUPD	time (sec)	—	17.91	—	—
	speed-up	—	—	—	—

Test matrix is of order 500.
** Probably has full orthonormalization due to the different grouping criterion used in the version of TINVIT on the Cray X-MP.

TABLE 7b
Speed-up over TQL2 *on the Alliant for computing all the eigenvalues and eigenvectors.*

i	Algorithm	$\frac{\text{Time (TQL2 on 1 CE)}}{\text{Time (algorithm } i \text{ on 8 CEs)}}$	$\frac{\text{Time (TQL2 on 8 CEs)}}{\text{Time (algorithm } i \text{ on 8 CEs)}}$
1	TREPS1	32.9	7
2	TREPS2	131.5	28
3	TQL2	4.7	1
4	BISECT+TINVIT	3.6	.8
5	SESUPD	27.1	5.8

Test matrix is of order 500.

TREPS1 and TREPS2 are close to that of BISECT+TINVIT but not as good as those of TQL2 and SESUPD [4].

In Tables 7a and 7b we give timing results comparing the performance of the above algorithms on both the FX/8 and the CRAY X-MP. Note that the time for TQL2 on one CE is 131 times slower than the time required by TREPS2 on 8 CE's, and that the time for TQL2 on 8 CE's is 28 times slower than that of TREPS2. Furthermore, TREPS2 is 4.8 times faster than SESUPD. In Tables 8a and 8b we compare both versions of our algorithm with TQL1 and BISECT for obtaining all the eigenvalues only. In Table 9 we show timing results for obtaining all the eigenvalues and eigenvectors of a random tridiagonal matrix on the Alliant FX/8.

TABLE 8a
Time and speed-up for computing all the eigenvalues.

		Alliant		CRAY X-MP	
		1 CE	8 CE	1 CPU	4 CPU
TREPS1	time (sec)	105.3	13.2	10.45	2.67
	speed-up	1	8.0	1	3.9
TREPS2	time (sec)	16.6	2.1	1.09	.4
	speed-up	1	7.9	1	2.7
TQL1	time (sec)	10.5	8.8	.87	—
	speed-up	1	1.2	1	—
BISECT	time (sec)	128.1	126.1	9.88	—
	speed-up	1	1.0	1	—

Test matrix is of order 500.

TABLE 8b
Speed-up over TQL1 *on the Alliant for computing all the eigenvalues.*

i	Algorithm	$\dfrac{\text{Time (TQL1 on 1 CE)}}{\text{Time (algorithm } i \text{ on 8 CEs)}}$	$\dfrac{\text{Time (TQL1 on 8 CEs)}}{\text{Time (algorithm } i \text{ on 8 CEs)}}$
1	TREPS1	.8	.7
2	TREPS2	5	4.2
3	TQL1	1.2	1
4	BISECT	.08	.07

Test matrix is of order 500.

TABLE 9
Computing all the eigenvalues and vectors of a random tridiagonal matrix on the Alliant FX/8.

	Time (second)	$\max_i \|Tz_i - \lambda_i z_i\|_2$	$\max_{i,j} \|Z^TZ - I\|_{i,j}$	Speed-up over TQL2
TREPS1	15.04	5.6×10^{-13}	5.1×10^{-12}	8
TREPS2	4.14	4.6×10^{-13}	1.3×10^{-11}	29
TQL2	120.54	2.8×10^{-14}	1.6×10^{-14}	1
BISECT+TINVIT	134.89	5.2×10^{-13}	6.4×10^{-12}	.9
SESUPD	6.65	2.5×10^{-14}	3.0×10^{-14}	18

Test matrices are of order 500.

TABLE 10
Results for full symmetric matrices on the Alliant FX/8.

Test matrix		Time (second)	$\max_{i,j} \|Z^TZ-I\|_{i,j}$
R	TREPS1	24.59	3.1×10^{-13}
	TREPS2	38.64	3.1×10^{-13}
A	TREPS1	23.96	1.1×10^{-12}
	TREPS2	12.64	5.7×10^{-13}

Test matrices are of order 500. **R** is full random symmetric matrix. $\mathbf{A}=(\mathbf{I}-2uu^T)\mathbf{T}(\mathbf{I}-2uu^T)$, where $\mathbf{T}=[-1,2,-1]$.

Table 10 compares the times consumed by both versions of our algorithm on full symmetric matrices. The times indicated do not take into account the reduction to the tridiagonal form. For the matrix $\mathbf{A}=(\mathbf{I}\text{-}2uu^T)\mathbf{T}(\mathbf{I}\text{-}2uu^T)$, where $\mathbf{T}=[-1,2,-1]$ and $u^Tu=1$, TREPS2 is very fast; but for a full symmetric random matrix **R**, TREPS2 is slower than TREPS1. This leads us to recall the implementation of the extraction algorithm. As mentioned earlier, although the Zeroin method is much faster than bisection, it may suffer from under- or overflow. Such problems are heralded by lack of convergence in the computation of some of the eigenvectors. To remedy this problem in TREPS2, we switch to bisection whenever the inverse iteration does not converge. The matrix **R** used in this experiment is an extreme case, where every single eigenvalue has first been computed by the Zeroin method, then recomputed by bisection; this explains the poor performance of TREPS2.

5. Conclusion. The algorithms TREPS1 and TREPS2 which have been presented in this paper are well suited for multiprocessors. A speed-up which is almost equal to the number of processors can be obtained as long as the number of desired eigenvalues is several times larger than the number of processors. They also achieve high efficiency because the amount of arithmetic operations for each element of data (a total of $O(n)$ elements) is high, thus avoiding the traditional bottleneck of memory bandwidth.

TREPS2, which extracts the eigenvalues using the Zeroin method, is much faster than the combination of EISPACK's BISECT and TINVIT, as well as TQL2. It can also be faster than SESUPD [4], on a multiprocessor such as the FX/8, especially when the linear recurrence (2.1) is well behaved. In fact we consider TREPS2 to be the algorithm of choice for obtaining either all the eigenvalues (it is faster than TQL1), or few of the eigenvalues and the corresponding eigenvectors of a tridiagonal matrix. Furthermore, comparisons with [4] indicate that TREPS2 is equally competitive with other multiprocessor algorithms that seek all the eigenvalues and vectors of full symmetric matrices. This is especially true since the back-transformation involved, in the full case, can be performed by matrix multiplication routines which are very efficient on both the FX/8 and the CRAY X-MP/48.

Acknowledgment. The authors would like to thank John Larson at CRAY Research for his valuable help in performing the experiments on the CRAY X-MP.

REFERENCES

[1] H. J. BERNSTEIN AND M. GOLDSTEIN, *Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices*, Courant Institute of Mathematical Sciences Report, New York University, New York, 1985.

[2] H.Bowdler, R. Martin, C. Reinsch and J. H. Wilkinson, *The* QR *and* QL *algorithms for symmetric matrices, Contribution* II/3, in Handbook for automatic computation, Vol. II, Linear Algebra, Springer-Verlag, Berlin, New York, 1971, pp. 227-240.
[3] S. C. Chen, D. J. Kuck and A. H. Sameh, *Practical parallel band triangular system solvers*, ACM Trans. Math. Software, 4 (1978), pp. 270-277.
[4] J. J. Dongarra and D. C. Sorensen, *A fast algorithm for the symmetric eigenvalue problem*, IEEE Proc. 7th Symposium on Computer Arithmetic, Urbana, IL, 1985, pp. 338-342, this Journal, 8 (1987), to appear.
[5] G. E. Forsythe, M. A. Malcom and C. B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.
[7] H.-M. Huang, *A Parallel Algorithm for Symmetric Tridiagonal Eigenvalue Problems*, CAC Document No. 109, Center for Advanced Computation, Univ. Illinois at Urbana-Champaign, February 1974.
[8] D. Kuck and A. Sameh, *Parallel computation of eigenvalues of real matrices*, IFIP Congress 1971, 2 (1972), pp. 1266-1272.
[9] D. Kuck, D. Lawrie, A. Sameh and E. Davidson, *Construction of a large-scale multiprocessor*, Cedar Document No. 45, Univ. Illinois at Urbana-Champaign, 1984.
[10] A. Ostrowski, *Solution of Equations and Systems of Equations*, Academic Press, New York, 1966.
[11] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, Oxford, 1965.