PARALLEL BISECTION ALGORITHMS FOR SOLVING THE SYMMETRIC TRIDIAGONAL EIGENPROBLEM *

J. M. BADíA† AND A. M. VIDAL‡

Abstract. In this paper we study the different approaches used so far to apply the bisection method for solving the symmetric tridiagonal eigenproblem. We review the sequential methods used to perform the final extraction of the eigenvalues and compare two new techniques that offer very good results. The sequential version of the bisection method we have implemented even surpasses the results of the QR iteration in some cases. We also perform an exhaustive survey of the approaches that can be used to parallelize the bisection method and we compare two parallel algorithms that apply different schemes for distributing the computation among the processors. The experimental analysis developed shows that the bisection method, besides its flexibility in the partial computation of the spectrum, is a method that offers very good results when it is adequately parallelized. We also show that the behaviour of the algorithms is clearly matrix-dependent.

Key words. symmetric tridiagonal eigenproblem, bisection method, Laguerre iteration, parallel algorithms

AMS subject classifications. 15A18, 68-04

1. Introduction. It is well known that one of the most important problems in numerical computing is the computation of eigenvalues and eigenvectors of matrices. There are many areas where this problem arises, such as structural dynamics, quantum chemistry, oceanography, economics theory or control process.

Often the matrices that appear when solving these problems have some special structure. Certainly, the symmetric tridiagonal matrices have received more attention, not only due to their intrinsic importance, but because they appear during the solution of the eigenproblem of general symmetric matrices.

There are many methods to compute eigenvalues and eigenvectors. Probably the most commonly used, due to its excellent performance in the sequential case, is the QR iteration. Recently, however, with the rise of parallel architectures some other methods have received attention. We can point out, for example, the Jacobi method, the divide and conquer approaches and the bisection and multisection methods. Each of them has its advantages and disadvantages regarding their execution time, the accuracy of the results they offer, the possibility of partially computing the spectrum of matrices, etc.

On the other hand, the increase in the use of parallel architectures is enormously affecting the solution of problems involving a large amount of computation, and is defining the kind of methods used to solve these problems. It is very important to make an effort to design and implement good algorithms that exploit the enormous potential of parallel architectures. Algorithms that solve the eigenproblem of different kinds of matrices should be included in the new mathematical libraries for parallel machines.

We have taken into account two important tendencies that are been followed to design and use parallel architectures. First, different machines like the Cray T3D are a good example of massively parallel computers, and can be included in the architectures with their

* This paper was partially supported by the projects: ESPRIT No. 9072-GEPPCOM: "Foundations of General Purpose Parallel Computing", and CICYT TIC96-1062-C03: "Parallel Algorithms for the computation of the eigenvalues of sparse and structured matrices".

† Computer Science Dpt. Univ Jaume I. Castellón. 12071. Spain. (badia@inf.uji.es).

‡ Computer Science Dpt. Univ. Politécnica de Valencia. Spain

memory physically distributed but logically shared. Another important tendency is the design of portable algorithms using message-passing libraries like the PVM.

In this paper we will focus on the eigenproblem of symmetric tridiagonal matrices. A lot of work has been done to solve this problem, and so this paper will first make an exhaustive review of the previous efforts developed by other authors, both in the parallel and sequential cases.

For various reasons, we will concentrate specifically on the bisection method. It provides excellent results in the sequential case, close to these of the best method (the QR iteration), it is flexible enough to compute any part of the spectrum, and it offers enormous potential for parallelism using different approaches.

During the experimental analysis we will pay special attention to the behaviour of the algorithms with different classes of tridiagonal matrices, because they are clearly problem-dependent.

The article is organized as follows. In section 2 we make a thorough review of the previous work to reflect the state of the art. In section 3 we briefly describe the sequential algorithm we have parallelized. Section 4 is devoted to the parallel algorithms we have implemented. In section 5 we describe the architecture of the Cray T3D, the test matrices used and the experimental results obtained. Finally, in section 6 we give some conclusions.

2. State of the art. In recent years different authors have developed many algorithms to compute the eigenvalues, and sometimes the eigenvectors, of symmetric tridiagonal matrices. However, all these algorithms can perfectly be classified into four basic methods: the QR iteration, the bisection and multisection method (BM), the divide and conquer method (DV) and the homotopy method. In this section we will review the origin and the main contributions for each one of these methods. We will specially focus on the BM method because the two parallel algorithms that we analyze in this paper use it.

Several papers that compare some of the previous methods have been presented. First, in the sequential case we should cite the paper [19], where the authors present some general ideas to design good symmetric eigenroutines to be included in LAPACK [1]. The practical comparative study developed in [37] is also very interesting. Some comparative analyses have also been performed in the parallel case and on different architectures. We can, for example, point out the classical papers [39] on shared memory machines, and [29] on distributed memory machines, and more recently the studies by [42] and [3].

QR iteration [26] has been considered the most efficient method to compute all eigenvalues in the sequential case. A parallel algorithm for multicomputers that uses this method can be found in [27]. In the symmetric tridiagonal case the cost of the QR factorization is of O(n). However, there are not efficient scalable parallel implementations of this method, so other approaches are being considered for parallel architectures. Moreover, other methods, such as DV and BM methods offer competitive results even in the sequential case.

A method that is presently being studied is the homotopy method, also called the continuation or embedding method. It is well known and has been used in the past for finding the zeros of nonlinear functions. Its theoretical foundation is the topology. The basic idea is to construct a homotopy from a trivial function to the one of interest. Under certain conditions, smooth curves starting from the trivial solutions will lead to the desired solutions.

The homotopy method has only recently begun to be used for the computation of matrix eigenvalues and eigenvectors. This method has received attention for this

application due to its natural parallelism. Therefore, algorithms based on this method are excellent candidates for multiprocessor computers.

The method was originally proposed in [15] for the computation of the eigenproblems of symmetric matrices, and it was practically applied in [36]. Afterwards this method was applied to the symmetric tridiagonal case in [35]. There are also parallel versions of the method in this case [38] and in the general symmetric case [42].

The application of the divide and conquer algorithms to the computation of the eigenvalues of symmetric tridiagonal matrices is based on the possibility of dividing the original matrix into two tridiagonal submatrices by using some kind of modification. Once we have computed the eigenvalues of the submatrices, it can be proved that the intervals they define allow us to delimit the position of the eigenvalues of the original matrix. The real power of the method is based on the possibility of applying the divide and conquer scheme at different levels.

The method originally developed by Cuppen [17] is based on the application of rank one modifications [14]. More recently some efficient algorithms that use rank two modifications have been implemented [37].

The development of the method is independent of the class of modifications applied. During the division stage, the original matrix is divided into many tridiagonal submatrices. If we continue this process until we have obtained matrices of size 1x1 or 2x2, their eigenvalues can be computed trivially. On the other hand, if we stop the division stage with bigger submatrices, their eigenvalues can be computed using, for example, the QR iteration. The development of the updating stage is based on the application of some algorithm to approximate the eigenvalues of the divided matrices using the eigenvalues of the submatrices as delimiters. In [17] the author uses a rational interpolation algorithm to approximate the zeros of the secular equation. In [13] and [44] the authors apply different combinations of the bisection and the Newton method to approximate the zeros of the characteristic polynomial. Finally, in [37] the authors use the Laguerre method to complete this task.

The divide and conquer method has been the object of some efficient parallelizations. Among them we can point out the first parallel version of the method developed in [20] or different parallel algorithms designed in [18] based on some variant of the same method. More recently, in [48] we can find an efficient parallelization of the method that uses rank two modifications in the updating stage.

On the other hand, the bisection and multisection method has produced the largest number of different approaches, both in the sequential and parallel case. The bisection and multisection method in the tridiagonal case has its foundation in the possibility of dividing the spectrum of the matrix. Specifically, it is possible to define a function called $neg_n(c)$ that, computed in any point of the real line, gives us the number of eigenvalues on each side of the point. This idea was originally exploited by Wallace Givens in [24] and [25]. That author used the Sturm sequence property [26] as a method to divide the spectrum.

If we define the symmetric tridiagonal matrix T as

$$T = \begin{bmatrix} a_1 & b_1 & & & 0 \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ 0 & & & b_{n-1} & a_n \end{bmatrix}, \tag{1}$$

and define the Sturm sequence of the matrix in the point c as

$$\{p_0(c), p_1(c), ..., p_n(c)\} \tag{2}$$

with

$$\begin{cases} p_0(c) = 1 \\ p_i(c) = \det(T_i - cI) \qquad i : 1, 2, ..., n \end{cases}$$

then the number of sign changes of this sequence equals the number of eigenvalues of T smaller than c.

The computation of this sequence can be done using the following widely known recurrence:

$$\begin{cases} p_0(c) = 1, \;\; p_1(c) = a_1 - c \\ p_i(c) = (a_i - c)p_{i-1}(c) - b_{i-1}^2 p_{i-2}(c) \qquad i : 2, 3, ..., n \end{cases} \tag{3}$$

The original sequential method by Givens was implemented and improved using several acceleration techniques in later papers [53], [30] and [31].

In [53] and [52] the author establishes that the bisection method is very good to isolate the eigenvalues, but that it can be notably accelerated if combined with some root finding method that converges faster than the plain bisection. The author cites as examples the Newton method (quadratic) and the Laguerre method (cubic).

Some improved versions of the bisection method were later presented. Specially important is [9], where the authors used for the first time a modified version of the Sturm sequence that avoids the overflow problems of the original sequence.

Specifically if we define

$$q_i(c) = \frac{p_i(c)}{p_{i-1}(c)} \qquad i : 1, 2, ..., n, \tag{4}$$

it can be shown very easily that the number of negative elements of this sequence equals the number of eigenvalues of T smaller than c. From this point on, we will call this value $neg_n(c)$.

Sequence (4) can be computed by using the following recurrence:

$$\begin{cases} q_0(c) = 1, \;\; q_1(c) = a_1 - c \\ q_i(c) = (a_i - c) - \dfrac{b_{i-1}^2}{q_{i-1}(c)} \qquad i : 2, 3, ..., n \end{cases} \tag{5}$$

The routine implemented in [9] has served as base and reference of almost all the later algorithms using the bisection method. Indeed, one version of that routine was included in the EISPACK library [2] with the name *bisect*.

The use of the modified Sturm sequence (4) is not problem-free. Supposse λ is an eigenvalue of T and is very close to an eigenvalues of its leading submatrix of size n-1, then a good number of root-finders can have a lot of problems to find a zero of $q_n(c)$ in λ. In this case we say that λ is a hidden eigenvalue, [43]. Another widely studied problem with (5) is that the recurrence can fail if any of its elements is zero.

Several authors [10] or [18] have solved this problem using different approaches. However, the basic idea is to replace the zero element with a very reduced value *eps*, that we can choose as the machine precision. Due to the special properties of the symmetric eigenproblem, this modification does not substantially affect the accuracy of the results obtained.

In [11] the BM method is divided into two phases. During the isolation phase we delimit the position of the eigenvalues into intervals that only contain one of them by applying a plain bisection algorithm. Later, during the extraction phase we can use faster root-finding algorithms to compute the eigenvalues. From these ideas, several authors have used different techniques during the extraction phase. For example, in [6] the author uses cubic polynomial interpolation as well as first and second order Newton iteration, and in [10] the authors use the pegasus method [21].

Some authors divide the extraction phase in two sub-phases. For example, in [10] the authors start the extraction process by applying several bisection steps until they obtain an interval where the characteristic polynomial is monotone. Then they can assure that the Pegasus method converges very fast. In [50] the author also uses the Pegasus method, but to compute the eigenvalues of hermitian Toeplitz matrices. In that paper the author uses the Pegasus method to find the zeros of the function $q(x)$ defined as the last element of the recurrence (5). As it is defined in (4), this function is discontinuous in the zeros of $p_{n-1}(x)$. Once isolated each eigenvalue, the author applies several bisection steps until he obtains an interval where $q(x)$ is continuous, then he applies the Pegasus method to extract the eigenvalue.

In this paper we implement two techniques of extraction and we compare them with others widely used so far. First we implement the Laguerre iteration to compute the zeros of the characteristic polynomial in each isolated interval, second we implement a method that starts the extraction phase by applying several steps of the Laguerre iteration in order to obtain an interval where the function $q(x)$ is continuous, then we apply the Pegasus method to approximate the eigenvalues as the zeros of this function. As it can be seen in [3], this last technique clearly gives better results than the one used in [50] and so we will use it in the experimental analysis shown in section 5.

Finally, we must point out that the application of the acceleration methods described above does not always produce faster convergence to the eigenvalues. Often the eigenvalues of some tridiagonal matrices tend to group in numerically indistinguishable clusters. In these cases the application of acceleration techniques can be less efficient than the use of the plain bisection approach.

Due to its special characteristics, the bisection method has been the object of multiple and very different parallelizations. In a later section of this paper we perform a survey of the fundamental approaches and analyze two parallel algorithms that we have implemented.

Moreover, we must point out the fact that the bisection method can be applied to the general symmetric case, though with very high costs. However, there are some kinds of structured matrices, called efficiently structured in [49], in which the cost of the method can be notably reduced. Very recently some sequential implementations of the method have been carried out in the case of Toeplitz matrices [50]. Besides, there are several parallel implementations of the bisection method in the case of banded Toeplitz matrices [4] and also in the Toeplitz-plus-Hankel case [5].

3. Sequential bisection algorithm. One of the fundamental features of the bisection method is its flexibility. While other methods like the QR iteration or the Cuppen's method

force us to compute all the eigenvalues, the BM methods allow us to compute any single eigenvalue, or any group of consecutive eigenvalues.

The foundation of the sequential algorithm is the adequate use of the function $neg_n(c)$ and the modified Sturm sequence (5). To implement this algorithm we begin with an initial interval (a,b) containing all the eigenvalues we want to compute. This interval can be obtained, for example, by means of the Gershgorin circles [26].

When we bisect the initial interval we define two subintervals (a,c) and (c,b), and from the computation of $n_c = neg_n(c)$ we can know the eigenvalues contained in each one. If we repeat this process we can finally obtain an interval (a,b) that isolates one eigenvalue, or that contains a cluster of eigenvalues with a separation less than a given value.

Once we have isolated the eigenvalue, we can use the function $q_n(c)$ defined by the last element of the modified Sturm sequence. We can then apply a root finding method faster than the plain bisection to extract the isolated eigenvalue as the only root of that function in the isolated interval.

```
program compute-group (p,q,tol,λ(p:q))
/* Given two integers p and q such that 1≤p<q≤n, this program
computes all the eigenvalues of T between λp and λq*/
   Compute the initial interval (ar,as) using Gershgorin circles
   Store the initial interval in the queue
   while queue non-empty
      if (interval (ar,as) is included in (ap-1, aq)
         and (s=r+1)) then /* contains only one eigenvalue of T */
            extract(ar,as,tol,λs)
            if queue non-empty then
                  get a new interval (ar,as) from the queue
            endif
      else
         if |ar - as| < tol then /* cluster of eigenvalues*/
            λi=(ar+as)/2, i=r+1, ..., s
         else
            c=(ar+as)/2; k=negn(c)
            if (k=r) or (k ≤ p-1) then
               r=k; ar=c
            else if (k=s) or (k ≥ q) then
               s=k; as=c
            else
               store the interval (c, as) in the queue
               s=k; as=c
            endif
         endif
      endif
   endwhile
```

After defining the way to approximate each individual eigenvalue, we can describe how to exploit these ideas to extract any group of consecutive eigenvalues. To face this task we will use a queue of intervals containing eigenvalues. During the process of isolating of each eigenvalue, the non-empty intervals produced that contain any of the eigenvalues to be computed are stored in a queue. Specifically the queue contains the limits of the intervals (a,b) and the value of the function $neg_n(c)$ in these points (n_a, n_b). Each time we extract

one eigenvalue or a cluster, we start the process again with a new interval taken from the queue in order to approximate the next one. The algorithm `compute-group` summarizes the process of computing any group of eigenvalues. In this algorithm an interval of the form (a_r, a_s) contains all the eigenvalues between λ_{r+1} and λ_s, and *tol* is a value that can be defined like in the stopping criterion (9).

The algorithm used to implement the function `extract(`a_r`,`a_s`,`*tol*`,`λ_s`)` defines the speed of the method, except when we have many clusters of eigenvalues. In [3] we prove that, of all the algorithms used so far, the Laguerre iteration is the one that offers the best results (see Figure 1). In the following section we describe how to use this iteration to approximate the eigenvalues.

3.1. The Laguerre iteration. The application of the Laguerre iteration as a method for accelerating the extraction of the eigenvalues was already suggested in [52], and that technique has been used in [37] as a basic step in a divide-and-conquer method. The Laguerre method has been proved to have cubic convergence in the neighbourhood of a single eigenvalue.

Each iteration of the Laguerre method is based on the following expression:

$$L_{\pm}(x) = x + \frac{n}{\left(-\frac{p'(x)}{p(x)}\right) \pm \sqrt{(n-1)\left[(n-1)\left(-\frac{p'(x)}{p(x)}\right)^2 - n\left(\frac{p''(x)}{p(x)}\right)\right]}} \tag{6}$$

where the characteristic polynomial $p(x) = \det(T - xI)$ can be evaluated as the n^{th} term of the recurrence (3). As we can see in (6), to evaluate one iteration of Laguerre we need not only $p(x)$, but its first and second derivatives $p'(x)$ and $p''(x)$. For the same reasons that we computed the modified Sturm sequence instead of the original one, we use the following scaled recurrences related with each derivative:

$$r_i = \frac{p_i'}{p_i} \text{ and } s_i = \frac{p_i''}{p_i} \qquad i = 0, 1, \ldots, n$$

These recurrences can be obtained by differentiating (3), scaling each term and taking (4) into account

$$\begin{cases} r_0 = 0, \;\; r_1 = -\dfrac{1}{q_1} \\ r_i = \dfrac{1}{q_i}\left[(a_i - x) r_{i-1} - 1 - \left(\dfrac{b_{i-1}^2}{q_{i-1}}\right) r_{i-2}\right] \end{cases} \qquad i = 2, 3, \ldots, n \tag{7}$$

$$\begin{cases} s_0 = 0, \;\; s_1 = 0 \\ s_i = \dfrac{1}{q_i}\left[(a_i - x) s_{i-1} - 2r_{i-1} - \left(\dfrac{b_{i-1}^2}{q_{i-1}}\right) s_{i-2}\right] \end{cases} \qquad i = 2, 3, \ldots, n \tag{8}$$

and, in (6) we denote

$$\frac{p'(x)}{p(x)} = r_n \text{ and } \frac{p''(x)}{p(x)} = s_n$$

An algorithm to evaluate the previous recurrences and to apply the Laguerre iteration can be found in [37] and in [3].

Another important aspect of the iterative method is the criterion to stop it. From an analysis of the accuracy of the Laguerre iteration that can be found in [37] we have chosen the following condition:

$$\left|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}\right| \leq \max\left\{\delta,\ \left|\tilde{\lambda}_i^{(k)}\right|\varepsilon\right\}, \tag{9}$$

where $\tilde{\lambda}_i^{(k)}$ is the k$^{\text{th}}$ approximate to the exact eigenvalue λ_i, ε is the machine precision and δ is an absolute error bound given by:

$$\delta = 2.5\varepsilon \max_{1\leq j\leq n-1}\left\{\left|b_j\right| + \left|b_{j+1}\right|\right\}.$$

4. Parallel algorithms

4.1. Antecedents of the parallel bisection method. The parallelization of the BM method admits different approaches and can be accomplished efficiently on different kinds of architectures. The idea of performing a parallel implementation of the bisection method to compute the eigenvalues appears very early in [33], or in [7], where the authors propose a technique of parallelization of the method on a MIMD architecture based on the use of a queue of intervals.

The idea of replacing the plain bisection by a multisection technique was also applied very early in order to obtain a large number of intervals to work in parallel with. Regarding this, we can point out the approaches by [28] on a SIMD architecture, or by [51], where the author performs an initial multisection to define groups of intervals containing the same number of eigenvalues and then distributes these groups to the processors. After this initial phase, each processor uses a bisection technique to compute the eigenvalues of its intervals. The application of a parallel multisection technique can also be found in [8].

In [12] a parallel algorithm that uses a "dynamic spawning" technique as an alternative to the use of a queue is studied. The algorithm is implemented on a MIMD simulator that includes a distributed and a shared memory. In that paper the three classes of parallelism that we can exploit in the bisection method appear for the first time.

1. We can parallelize the computation of the modified Sturm sequence that allows us to obtain the functions $neg_n(c)$ and $q_n(c)$.
2. We can parallelize the computation of each eigenvalue, so that all processors cooperate in its calculation.
3. We can parallelize the computation of groups of eigenvalues, so that different processors, in parallel, compute different groups of eigenvalues.

One of the best known parallel versions of the bisection method is the one proposed in [39] on a shared memory machine. The algorithm, applying the ideas of [28] and [33], uses a multisection technique during the isolation phase, while during the extraction phase a plain bisection method or the *Zeroin* method (based on the combination of the secant method and the bisection) is applied. Besides, in [39] the authors prove that bisection is

better than multisection in order to compute a single eigenvalue. However, during the isolation phase they prefer to apply the multisection technique.

Another classical reference about the parallel bisection method is the paper by [29], where the authors compare three different approaches to compute the eigenpairs of symmetric tridiagonal matrices: the Cuppen's method, the bisection method and a method that combines multisection during the isolation phase and bisection during the extraction phase. The algorithms are implemented and studied on a distributed memory machine with hypercube topology.

Another version of the parallel bisection method is presented in [32]. In that paper the algorithm is implemented on a network of transputers using a bidirectional ring topology. The author describes one algorithm for the computation of a single eigenvalue and another to compute the whole spectrum. In the first case he applies a parallel multisection method, while in the second case the author uses a method that separates the isolation and extraction phases.

The load unbalance problem that can arise in the previous method can be solved with the approach presented in [46], where the author also uses an array of transputers. In this paper three different algorithms that apply pipeline and farming technique in order to balance the load are described.

In [10] the authors apply a bisection method during the isolation phase and then use the Pegasus method to extract the isolated eigenvalues once they have obtained good starting points.

Regarding the first class of parallelism cited above, the author of [22] uses a “recursive doubling” strategy to compute the Sturm sequences in parallel. In [40] the authors apply a parallel technique that combines the advantages of segmentation and the cyclic reduction method for solving general linear first order recurrence problems.

Other algorithms that use vector and SIMD architectures have been proposed. We can mention, for example, the method proposed in [47] where the author performs a comparative study of the bisection and multisection methods on different vector computers. Another algorithm that use this class of architecture can be found in [16].

4.2. Parallel algorithms implemented. Continuing with the evolution of the parallel bisection method, in [3] we performed an experimental comparison of the basic approaches used to parallelize the method, and we compared the results with parallel implementations of the divide and conquer method. The complete analysis is developed on different classes of parallel architectures and using different programming environments. In this paper we extract and analyze two of the parallel algorithms that obtain the best experimental results by applying the bisection method.

Both algorithms exploit the third kind of parallelism described in the previous section. The basic difference between the algorithms is the technique used to distribute the intervals to the processors.

The first of the parallel algorithms implemented, that we will call *pstsep*, uses a static distribution scheme. The initial interval (a,b) containing all the eigenvalues to be computed is distributed to all the processors. Each processor is in charge of approximating a group with the same number of consecutive eigenvalues. To accomplish this task each processor uses the algorithm `compute-group` described in section 3. During the isolation phase, each processor only stores the intervals containing the eigenvalues that it is in charge of computing.

The algorithm presents the advantage of using an *SPMD* (single program multiple data) scheme. Besides, every processor works fully independent of the others without having to

perform any communication or synchronisation. As far as every processor computes the same number of eigenvalues, the load could be perfectly balanced. However it is well known that the eigenvalues of different classes of tridiagonal matrices can have very different distributions along the spectrum. Even in a limit case like the one that occurs with the Wilkinson matrices [52], where the majority of the eigenvalues can form clusters in pairs. In these circumstances, the load balance in the *pstsep* algorithm can be quite poor, producing bad parallel results. This justifies the implementation of an algorithm that applies a dynamic load distribution technique.

The second parallel algorithm implemented and analyzed, that we will call *pstfar*, uses a dynamic technique for distributing the intervals to the processors. The idea we have applied is a generalization of the one used in [46] on an array of transputers. Instead of using a farming method, we apply a more general master/slave strategy where all communications among the master and the slaves can be performed directly.

The master processor is in charge of managing a queue of intervals, and sending them each time one of the slaves is ready to compute a new eigenvalue. The master starts by sending the initial interval (a,b) to one of the slaves. This processor computes the first eigenvalue contained in the interval. The non-empty intervals produced during the isolation phase are sent to the master as soon as they are produced. The master sends the intervals back to the slaves or stores them in the queue.

```
Process master
   received=0  -- number of eigenvalues received
   ready=p-1   -- number of slaves ready to receive an interval
   Send the initial interval to one slave
   while received < number of eigenvalues to compute
      Receive data from the slaves
      case
         • an interval is received
           Store it in the queue
           if ready >0 then
                Send the interval to a ready slave
                ready = ready - 1
           endif
         • an eigenvalue or cluster is received
           received = received + eigenvalues received
           ready = ready + 1
           if queue non-empty then
                Send an interval to a ready slave
                ready = ready - 1
           endif
      endcase
   endwhile
   Broadcast the signal to finish.
```

On the other hand, the slave processors wait to receive an interval and are in charge of approximating the first eigenvalue contained in it by means of the algorithm described in section 3. Each time the slave extracts an eigenvalue or a cluster of eigenvalues it sends the result to the master. The master collects the eigenvalues and is able to know that the slave is ready to receive a new interval.

```
Process slave
   while the signal to finish is not received
      Receive an interval
      if interval contains more than one eigenvalue then
            Bisect until the first eigenvalue or cluster is isolated
            Send the non-empty intervals to the master
      endif
      Extract and send the eigenvalue or cluster to the master
   endif
```

5. Experimental analysis.

5.1. Computational Environment. All the practice experiments have been performed on a CRAY T3D. This machine is a highly scalable multiprocessor that consists of from 32 up to 2048 very powerful super scalar processors. Thus, it can be included in the class of massively parallel computers. Our target machine is located in the Ecole Politechnique Fédérale of Lausanne (Switzerland) and consists of 256 processors.

The CRAY T3D is a shared distributed memory multiprocessor. That is, each processor has its own local memory but the memory of every node can be globally addressed. In this way, each processor can access the contents of the memory of any other processor. This operation is performed by hardware, by using special circuits incorporated to the different process elements. As a consequence, the access and management of data located in every processor is transparent for the user, who may not worry about the location of the data to be accessed in the memory by its programs.

The basic node of this parallel computer consists of two process elements (PEs) and a routing switch. Each PE includes an Alpha microprocessor by DEC, a local memory and an adequate support hardware, which in combination with the routing switch, allows the processor to globally address all the local memories.

The different nodes of the CRAY T3D are interconnected through a 3D bi-directional torus network. This network permits a bandwidth up to 300 Mbytes per second over each one of the three possible dimensions.

To implement the parallel algorithms on this machine we have used the standard PVM message-passing environment [23]. This allows us to obtain a highly portable code and to make comparisons of the experimental results on different types of parallel machines [3].

5.2. Experimental results. As we said before, the performance of the sequential and parallel algorithms strongly depends on the problem, i.e., on the kind of the tridiagonal matrix on which they are applied. Both the BM and the DV methods depend on the distribution of the eigenvalues along the spectrum. Moreover, the results of the BM method are greatly influenced by the number of clusters and hidden eigenvalues. On the other hand, the performance of the DV algorithms depends on the number of deflations produced during its execution.

Several authors have started from the previous arguments to analyse the computing algorithms for several types of matrices, giving rise to a wide set of test matrices which represents the most diverse conditions. In the analysis of our algorithms we are going to utilise a group of matrices similar to that used in [37] and [42]. This group consists of 12 types of matrices whose behaviour can differ from the different algorithms studied. For the specific case of the *pstsep* and *pstfar* algorithms we can group the above matrices into four basic classes, which we shall use as target matrices to analyze the cited algorithms. A thorough study of the characteristics of the twelve types of matrices and the corresponding

analysis of the behaviour of bisection/multisection and divide-and-conquer algorithms on them can be found in [3].

The first class of matrices is represented by tridiagonal Toeplitz matrices, in which the eigenvalues are symmetrically distributed with regard to the centre of the spectrum, and tend to group near the extremes but without forming a cluster. Class 2 is given by tridiagonal matrices whose eigenvalues are uniformly distributed along the spectrum. Class 3 is constituted by the Wilkinson matrices [52]. Most of their eigenvalues are grouped in clusters of two eigenvalues. The matrices of the last class have their eigenvalues grouped in a single cluster, with the exception of one, which is far from the rest and whose value is one.

Let us now summarise some of the experimental results obtained by comparing different sequential and parallel algorithms for computing the eigenvalues of symmetric tridiagonal matrices. A wider study can be found in [3].

The results for the sequential case have been obtained by using Fortran language on an Alpha processor of the CRAY T3D.

First of all, it is worth noting that the bisection method, applied in two phases, has been the basis for all the algorithms presented in this work. The efficiency of the sequential algorithm is based mainly on the speed of the approximation method utilised during the extraction phase. In Figure 1, the execution time of four very representative approximation algorithms is compared. These algorithms are based on the pure bisection (bis), the Newton method (nwt), the Laguerre method (lag) and the Laguerre-plus-Pegasus method (lpe).

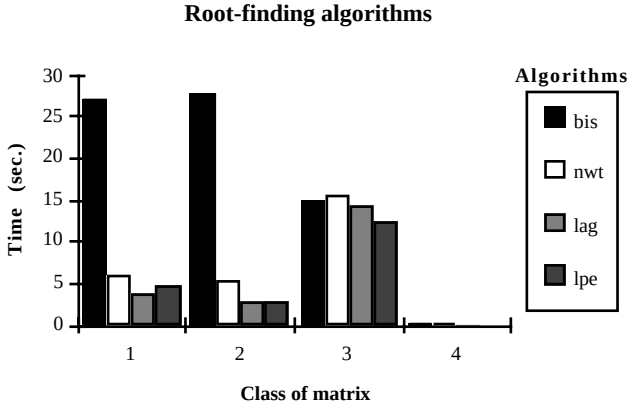

FIG. 1. *Comparison of extraction methods. n=1024.*

Figure 1 helps us to clearly verify that the experimental results depend on the type of matrix whose eigenvalues must be computed. As can be observed, the computing time for the matrices of type 4 is much smaller than the rest and cannot be represented in the same scale. It can also be noticed that for type 3 matrices (Wilkinson matrices) the final extraction method has little influence, as practically all the eigenvalues are located in clusters by pairs. Thus, the algorithm has difficulties to isolate most of the eigenvalues and the majority of the execution time is devoted to this phase, where just pure bisection is utilized.

In the other two types of matrices, it can be seen how pure bisection method (bis) is clearly slower than the others, and the Laguerre method (lag) is the most efficient one, even surpassing the Newton method (nwt) which has been considered by some authors, [6], [46], [45], as the most efficient one for the extraction phase of the method.

As the Laguerre method has proved to be the most efficient one, we have utilised it to implement the extraction phase of the two-phases bisection algorithm (*dstrid*). In Figure 2 we compare it with other methods for computing eigenvalues of symmetric tridiagonal matrices.

The *dspmg* algorithm is the implementation performed in [37], based on a method of DV type which combines rank-two modifications during the division phase with the use of the Laguerre method during the updating phase.

The *dsterf* algorithm is the routine of LAPACK which applies the QR iteration to compute all the eigenvalues of symmetric tridiagonal matrices.

Finally, the *dstebz* algorithm is the routine of LAPACK which applies pure bisection to solve the same problem.

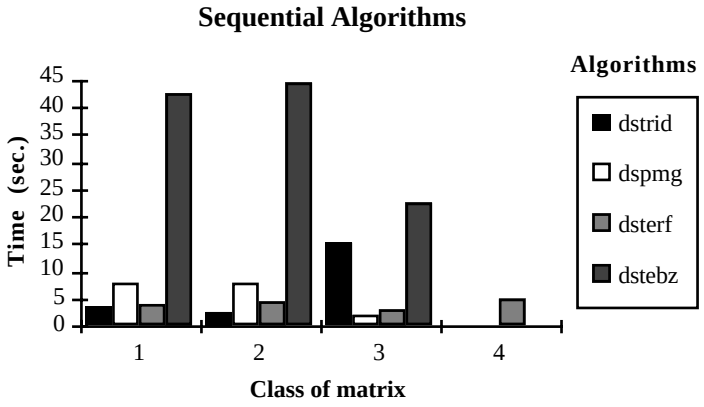

FIG. 2. *Comparison of sequential algorithms. n=1024.*

In Figure 2 we can check that, except for the *dsterf* algorithm, the remaining ones clearly depend on the problem. The *dstrid* algorithm surpasses the routines of LAPACK and the DV algorithm, except for the type 3 matrices (Wilkinson matrices). We can also see that the pure bisection method (*dstebz*) achieves worse performance than the obtained by combining it with the Laguerre method during the extraction phase (*dstrid*).

Tables 1 and 2 enable us to compare the different speed-ups of the two parallel algorithms described in section 4, with regard to the best sequential algorithm (*dstrid*). The results in this case depend on the type of parallelization and on the influence of the features of different types of matrices. The tables allow us to study the influence of modifying either the number of processors utilised or the size of the matrices whose eigenvalues are computed. The results in the case of type 4 matrices with sizes larger than 1024 are not shown because the memory needed to generate them surpasses that available.

It is worth pointing out the excellent performance of the *pstsep* algorithm. It achieves almost optimum speedup with practically any matrix size and any number of processors. Moreover, it scales very well in the sense that it presents no degradation of the performance when the number of processors increases, even if the size of the matrix is kept constant.

Although the use of a dynamic distribution strategy in the *pstfar* algorithm is intended to solve the problem of a possible load imbalance in the *pstsep* algorithm, it introduces a new factor which can severely penalize the performance in a message-passing environment: the communications overhead. While the *pstsep* algorithm can evolve with no communications, the *pstfar* algorithm requires the sending of a large number of short messages. This fact strongly determines the relative performance of both algorithms. Thus,

pstfar can only reach the performance of the *pstsep* when a large enough arithmetic cost compensates the cost of the communications introduced. This can happen in the case of the Wilkinson matrices, where the cost of extracting each eigenvalue is high, or for very large scale matrices.

On the other hand, the application of both parallel schemes to the matrices of class 4 is completely useless. The fact that all the eigenvalues are located in a single cluster and the short execution time of the algorithm in this case make it impossible to obtain the advantages of a parallel scheme based on the third class of parallelism cited in 4.1.

TABLE 1
Speedup modifying the number of processors. n=1024.

		Number of processors				
Class	Algorithm	2	4	8	16	32
1	*pstsep*	2,00	3,97	7,70	14,51	27,47
	pstfar	1,50	2,98	5,73	7,81	7,85
2	*pstsep*	2,00	3,91	7,67	15,09	29,23
	pstfar	1,44	2,82	5,39	6,04	5,80
3	*pstsep*	1,98	3,95	7,69	15,01	29,27
	pstfar	1,69	3,36	6,65	12,97	23,92
4	*pstsep*	1,00	1,04	1,03	1,04	1,03
	pstfar	0,96	0,97	0,96	0,95	0,94

TABLE 2
Speedup modifying the size of the matrices. p=32.

		Matrix size					
Class	Algorithm	128	256	512	1024	2048	4096
1	*pstsep*	20,03	23,11	26,63	27,47	30,01	30,07
	pstfar	1,39	2,38	4,26	7,85	19,78	23,87
2	*pstsep*	22,07	25,69	28,20	29,23	28,87	26,59
	pstfar	1,04	1,76	3,19	5,80	14,24	22,33
3	*pstsep*	25,49	27,58	29,15	29,27	29,57	29,76
	pstfar	5,38	9,58	16,35	23,92	23,82	23,92
4	*pstsep*	1,06	1,06	1,21	1,03		
	pstfar	0,66	0,82	0,95	0,94		

Figure 3 allows us to analyze the scalability of the two parallel algorithms [34]. Specifically we use a isospeed scalability metric as it is defined in [41]. In this sense, we consider that a parallel algorithm is scalable if we could maintain the execution time by simultaneously increasing the number of processors and the size of the problem. In the case of the two parallel algorithms that we are analyzing the problem size is of $O(n^2)$ to compute all the eigenvalues of the matrix. In order to maintain the size of the sub-problem solved in each processor, each time we double the size of the matrix we have to multiply by four the number of processors. Figure 3 shows the almost perfect scalability of the *pstsep* algorithm, even when we use 256 processors. Therefore, we can consider the *pstsep* algorithm very adequate for solving this problem on massively parallel architectures. On the other side the overhead cost introduced by the communications in the case of the *pstfar* algorithm produces a clear increment of the execution time, even if scaling the problem size.

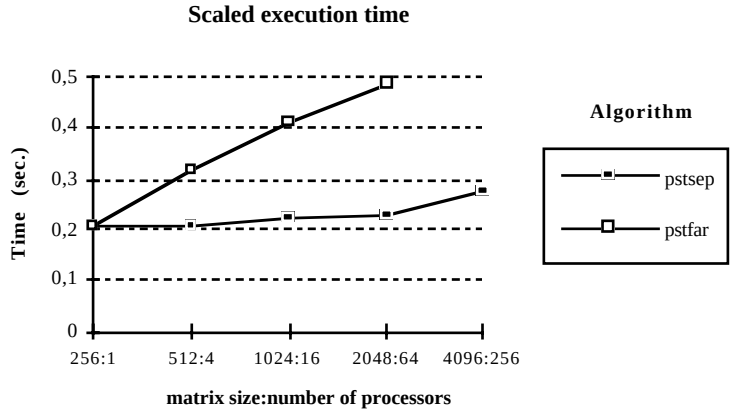

FIG. 3. Scaled execution time of the parallel algorithms (matrices of type 1).

6. Conclusions. From the work presented in [3], of which one part has been compiled in this paper, we can conclude that the best method to solve the symmetric tridiagonal eigenvalue problem is the bisection method. This method is specially efficient when it is implemented in two phases and an adequate choice of the method used in the isolation phase and in the extraction phase is made. The experimental results obtained confirm that the parallel bisection method, adequately implemented, surpasses the best divide-and-conquer algorithms developed up till now.

To get a good implementation of the bisection method, we have compared different root-finding algorithms. For the first time, we have combined the bisection method in the isolation phase with the Laguerre iteration in the extraction phase. The two-phases bisection algorithm thus implemented surpasses the performance of what has generally been considered the best sequential algorithm, the QR iteration.

The parallelization of the bisection method can be carried out in several ways. As far as we know, the best performance with this algorithm is achieved by equally dividing the computation of the different eigenvalues among the processors. This scheme can be implemented by means of two approaches for distributing the load among the nodes of the multiprocessor: a static approach and a dynamic approach. Although the dynamic distribution allows a better load balance, the introduction of a large number of communications has turned out to be decisive in decreasing the performance of the algorithm, specially with small and medium size matrices. Moreover, the static version shows a very good scalability and so it is very adequate for massively parallel architectures.

It is also worth noticing that both the bisection and the divide and conquer methods are dependent on the problem. This has been clearly stated by observing the behaviour of the different algorithms when the type of the matrix varies, and more specifically, when the distribution of the eigenvalues in the spectrum, the number of clusters, the deflations or the number of hidden eigenvalues varies.

Acknowledgements. We want to express our gratitude to Cray Research Inc. for providing the supercomputing facilities, specifically the Cray T3D. We also want to express our gratitude to professor Zhonggang Zeng for all the information that he sent us about his algorithm *dspmg*.

REFERENCES

[1] E. ANDERSON, Z. BAY, et al., *LAPACK User's Guide*, SIAM, Eds. 1992.

[2] L. ARGONNE NAT, *EISPACK. Eigensystem package*, Illinois, (1972).

[3] J. M. BADÍA, *Algoritmos Paralelos para el Cálculo de los Valores Propios de Matrices Estructuradas*, Ph. D. Thesis, Univ. Politécnica de Valencia, 1996.

[4] J. M. BADÍA AND A. M. VIDAL, *Efficient solution of the eigenproblem of banded symmetric Toeplitz matrices on multicomputers*, in 3rd. Euromicro Workshop on Parallel and Distributed Processing. (IEEE Computer Society Press, San Remo, Italy, 1995), pp. 416-423.

[5] J. M. BADÍA AND A. M. VIDAL, *Parallel Computation of the Eigenstructure of Toeplitz-plus-Hankel matrices on Multicomputers*, in *Parallel Scientific Computing*, (Lecture Notes in Computer Science, 879), J. Dongarra and J. Wasniewski, Eds. Springer-Verlang. (1994), p. 33-40.

[6] G. BAKER, *Accelerated Bisection Techniques for Tri- and Quintadiagonal Matrices*, Int. J. for Num. Meth. in Eng., (1992), pp. 203-218.

[7] R. H. BARLOW AND D. J. EVANS, *A parallel organization of the bisection algorithm*, Comput. J., 22 (1977), pp. 267-269.

[8] R. H. BARLOW, D. J. EVANS AND J. SHANEHCHI, *Parallel multisection applied to the eigenvalue problem*, Comput. J., 26 (1983), pp. 6-9.

[9] W. BARTH, R. S. MARTIN AND J. H. WILKINSON, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method*, Numer. Math., 9 (1967), pp. 386-393.

[10] A. BASERMAN AND P. WEIDNER, *A Parallel Algorithm for Determining all Eigenvalues of Large Real Symmetric Tridiagonal Matrices*, Parallel Computing, 18 (1992), pp. 1129-1141.

[11] H. J. BERNSTEIN, *An Accelerated Bisection Method for the Calculation of Eigenvalues of A Symmetric Tridiagonal Matrix*, Numer. Math., 43 (1984), pp. 153-160.

[12] H. J. BERNSTEIN AND M. GOLDSTEIN, *Parallel Implementation of Bisection for the Calculation of Eigenvalues of Tridiagonal Symmetric Matrices*, Computing, 37 (1986), pp. 85-91.

[13] D. BINI AND V. PAN, *Practical Improvement of the Divide-and-Conquer Eigenvalue Algorithms*, Computing, 48 (1992), pp. 109-123.

[14] J. R. BUNCH, C. P. NIELSEN AND D. C. SORENSEN, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31-48.

[15] M. T. CHU, *A simple application of the homotopy method to symmetric eigenvalue problems*, Lin. Alg. Appl., 105 (1984), pp. 225-236.

[16] J. M. CONROY AND L. J. PODRAZIK, *A parallel inertia method for finding eigenvalues on vector and SIMD architectures*, SIAM J. Sci. Stat. Comp., 2 (1995), pp. 500-505.

[17] J. J. M. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, SIAM J. Sci. Stat. Comp., 2 (1981), pp. 139-154.

[18] U. DE ROS, *Soluzione parallela su una rete di transputer del problema degli autovalori per una matrice tridiagonale simmetrica*, Ph. D. Thesis, Politecnico di Milano, Facoltà di Ingegneria, Dipartimento di Matematica, 1993.

[19] J. DEMMEL, J. DU CROZ, et al., *Guidelines for the Design of Symmetric Eigenroutines, SVD and Iterative Refinement for Linear Systems*, MCS-TM-111, (Lapack Working Note #4), Argonne National Laboratory, (1988).

[20] J. J. DONGARRA AND D. C. SORENSEN, *A Fully Parallel Algorithm for the Symmetric Eigenvalue Problem*, SIAM J. Sci. Stat. Comp., 2 (1987), pp. s139-s154.

[21] M. DOWELL AND P. JARATT, *The Pegasus method for computing the root of an equation*, BIT, 12 (1972), pp. 503-508.

[22] D. J. EVANS, *Design of numerical algorithms for Supercomputers*, in *Scientific Computing on Supercomputers*, J. T. Devreese, Ed. (1989), p. 101-151.

[23] A. GEIST, A. BEGUELIN, et al., *PVM3 User's Guide and Reference Manual*, Oak Ridge National Laboratory, 1994.

[24] J. W. Givens, *A method of computing eigenvalues and eigenvectors suggested by classical results on symmetric matrices*, U.S. Nat. Bur. Standards Applied Mathematics Series, (1953), pp. 117-122.

[25] J. W. GIVENS, *Numerical computations of the characteristic values of a real symmetric matrix*, ORNI-1574, Oak Ridge National Laboratory, (1954).

[26] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, 1989.

[27] G. HENRY AND R. VAN DE GEIJN, *Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality*, SIAM J. Sci. Comp., 4 (1996), pp. 870-883.

[28] H. M. HUANG, *A parallel algorithm for symmetric tridiagonal eigenvalue problems*, CAC Document No. 109, Center for Advanced Computation, University of Illinois at Urbana-Champaign, (1974).

[29] I. C. F. IPSEN AND E. R. JESSUP, *Solving the Symmetric Tridiagonal Eigenvalue Problem on the Hypercube*, SIAM J. Sci. Stat. Comp., 2 (1990), pp. 203-229.
[30] W. KAHAN, *Accurate eigenvalues of a symmetric tridiagonal matrix*, CS41, Computer Science Dpt., Stanford University, (1966).
[31] W. KAHAN AND J. VARAH, *Two working algorithms for the eigenvalues of a symmetric tridiagonal matrix*, CS43, Stanford University, (1966).
[32] T. Z. KALAMBOUKIS, *The Symmetric Tridiagonal Eigenvalue Problem on a Transputer Network*, Parallel Computing, 15 (1990), pp. 101-106.
[33] D. J. KUCK AND A. H. SAMEH, *Parallel Computation of Eigenvalues of Real Matrices*, in IFIP Congress. N. Holland, Eds. (1971), pp. 1266-1272.
[34] V. KUMAR, A. GRAMA, et al., *Introduction to Parallel Computing. Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Reedwood City, California, 1994.
[35] K. LI AND T. Y. LI, *An algorithm for symmetric tridiagonal eigenproblems - divide and conquer whith homotopy continuation*, SIAM J. Sci. Statist. Comput., 14 (1993), pp. 735-751.
[36] T. Y. LI AND N. H. RHEE, *Homotopy algorithm for symmetric eigenvalue problems*, Numer. Math., 55 (1989), pp. 265-280.
[37] T. Y. LI AND Z. ZENG, *The Laguerre iteration in solving the symmetric tridiagonal eigenproblem, revisited*, SIAM J. Sci. Stat. Comp., 5 (1993), pp. 1145-1173.
[38] T. Y. LI, H. ZHANG AND X. H. SUN, *Parallel homotopy algorithm for symmetric tridiagonal eigenvalue problems*, SIAM J. Sci. Stat. Comp., 12 (1991), pp. 464-485.
[39] S. S. LO, B. PHILIPPE AND A. SAMEH, *A Multiprocessor Algorithm for the Symmetric Tridiagonal Eigenvalue Problem*, SIAM J. Sci. Stat. Comp., 2 (1987), pp. s155-s165.
[40] M. LU AND X. QIAO, *Applying parallel computer systems to solve symmetric tridiagonal eigenvalue problems*, Parallel Computing, 18 (1992), pp. 1301-1315.
[41] I. MARTÍN, F. TIRADO AND L. VÁZQUEZ, *Some Aspects about the Scalability of Scientific Applications on Parallel Architectures*, Parallel Computing, 9 (1996), pp. 1169-1196.
[42] M. OETTLI, *The Homotopy Method Applied to the Symmetric Eigenproblem*, Ph. D. Thesis, Swiss Federal Institute of Technology Zurich, 1995.
[43] B. N. PARLETT, *The symmetric Eigenvalue Problem*, Prentice-hall, Inc., 1980.
[44] R. PAVANI AND U. DE ROS, *A parallel algorithm for the symmetric eigenvalue problem*, in International Congress on Industrial and Applied Mathematics. (Hamburg, 1995).
[45] V. PEREYRA AND G. SCHERER, *Eigenvalues of Symmetric Tridiagonal Matrices: A Fast, Accurate and Reliable Algorithm*, J. Inst. Math. Appl., 12 (1973), pp. 209-222.
[46] R. M. S. RALHA, *Parallel Solution of the Symmetric Tridiagonal Eigenvalue Problem on a Transputer Network*, in SEMNI 93. (La Coruña, España, 1993), pp. 1026-1033.
[47] H. D. SIMON, *Bisection is not optimal on vector processors*, SIAM J. Sci. Stat. Comp., 1 (1989), pp. 205-209.
[48] C. TREFFTZ, C. C. HUANG, et al., *A scalable eigenvalue solver for symmetric tridiagonal matrices*, Michigan State University, (1994).
[49] W. F. TRENCH, *Numerical Solution of the Eigenvalue Problem for Efficiently Structured Hermitian Matrices*, Lin. Alg. Appl., 154-156 (1991), pp. 415-432.
[50] W. F. TRENCH, *Numerical Solution of the Eigenvalue Problem for Hermitian Toeplitz Matrices*, SIAM J. Matrix Anal. Appl., 2 (1989), pp. 135-146.
[51] Y. WALLACH, *Alternating Sequential Parallel Processing*, (Lecture Notes in Computer Science, 127), Springer-Verlag, Berlin-Heidelberg-New York. (1982).
[52] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965.
[53] J. H. WILKINSON, *Calculation of the Eigenvalues of a Symmetric Tridiagonal Matrix by the Method of Bisection*, Numer. Math., 4 (1962), pp. 362-367.