An Efficient Parallel Algorithm for the Symmetric Tridiagonal Eigenvalue Problem

Maria Antónia Forjaz[1] and Rui Ralha

Departamento de Matemática
Universidade do Minho
Campus de Gualtar
4710-057 Braga, Portugal
Tel +351 253604340, Fax +351 253678982
maf@math.uminho.pt, r_ralha@math.uminho.pt

Abstract. An efficient parallel algorithm, *farmzeroinNR*, for the eigenvalue problem of a symmetric tridiagonal matrix is implemented in a distributed memory multiprocessor with 112 nodes [forjaz2000]. The basis of our parallel implementation, is an improved version of the *zeroinNR* method [ralha93]. It is consistently faster than simple bisection and produces more accurate eigenvalues than the QR method. As it happens with bisection, *zeroinNR* exhibits great flexibility and allows the computation of a subset of the spectrum with some prescribed accuracy. Results were carried out with matrices of different types and sizes up to 10^4 and show that our algorithm is efficient and scalable.

1 Introduction

The computation of the eigenvalues of symmetric tridiagonal matrices is one of the most important problems in numerical linear algebra. The reason for this is the fact that in many cases the initial matrix, if not already in tridiagonal form, is reduced to this form using either orthogonal similarity transformations, in the case of dense matrices, or the Lanczos method, in the case of large sparse matrices.

Essentially we can consider three different kinds of methods for this problem: the QR method and their variations [parlett80], [demmel97], the divide-and-conquer methods[2] [cuppen81], [dongarra87], and the bisection-multisection methods [wilkinson65], [ralston78], [parlett80], [bernstein84]. The bisection method is a robust method but is slower than the other methods for the computation of the complete set of eigenvalues. However, because of the excellent opportunities it offers for parallel processing, several parallel algorithms have been proposed which use bisection to isolate each eigenvalue and then some additional technique

[1] Candidate to the Best Student Paper Award

[2] Available as LAPACK routine sstevd; a good choice if we desire all eigenvalues and eigenvectors of a tridiagonal matrix whose dimension is larger than about 25 [demmel97, pg. 217].

with better convergence rate to compute the eigenvalue to the prescribed accuracy [phillipe87], [jessup90], [kalambouskis90], [baserman92], [demmel93]. One of such methods, dubbed *zeroinNR*, has been proposed in [ralha93] and uses an original implementation of the Newton-Raphson's method for this purpose.

2 A Sequential Algorithm: *zeroinNR*

Let A be a real, symmetric tridiagonal matrix, with diagonal elements $a_1, \ldots, a_n$ and off-diagonal elements $b_1, \ldots, b_{n-1}$. The sequence of leading principal minors of A is given by

$$\begin{cases} p_0(\lambda) = 1 \\ p_1(\lambda) = a_1 - \lambda \\ p_i(\lambda) = (a_i - \lambda)p_{(i-1)}(\lambda) - b_i^2 p_{(i-2)}(\lambda), \quad i = 2, 3, \ldots, n. \end{cases} \tag{1}$$

It is well known that *the number of variations of sign in this sequence equals the number of eigenvalues of A which are strictly smaller than λ.*

To avoid overflow problems, the sequence (1) can be modified to the form

$$\begin{cases} q_0(\lambda) = 1 \\ q_1(\lambda) = a_1 - \lambda \\ q_i(\lambda) = p_i(\lambda)/p_{i-1}(\lambda), \ i = 2, 3, \ldots, n \end{cases} \tag{2}$$

and the terms of the new sequence can be obtain by the following expressions,

$$\begin{cases} q_0(\lambda) = 1 \\ q_1(\lambda) = a_1 - \lambda \\ q_i(\lambda) = (a_i - \lambda) - b_i^2/q_{i-1}(\lambda), \ i = 2, 3, \ldots, n \end{cases} \tag{3}$$

where the number of negative terms $q_i(\lambda), \ i = 0, \ldots, n$, is equal to the number of eigenvalues strictly smaller than λ. This is the basis for the bisection method implemented in [barth67], which is known to have excellent numerical properties in the sense that it produces very accurate eigenvalues. The drawback of bisection is its linear convergence rate[3] that makes the method slower than others, at least for the computation of the complete system. Different authors have proposed modifications of the simple bisection method in order to accelerate its convergence. One such proposal, dubbed the *zeroinNR* method, has been given in [ralha93] and essentially uses Newton-Raphson's method to find an eigenvalue after it has been isolated by bisection. The correction $p_n(x_k)/p'_n(x_k)$, in the iterative formula of the Newton-Raphson method,

$$x_{k+1} \leftarrow x_k - \frac{p(x_k)}{p'(x_k)}, \tag{4}$$

is obtained without explicitly calculating the values of the polinomial $p_n(x_k)$, and its derivative $p'_n(x_k)$, therefore avoiding overflow and underflow in such

[3] The bisection method converges linearly, with one bit of accuracy for each step.

computations. For this purpose the following algorithm has been derived. From (2) we have,

$$p_i = q_i\, p_{i-1}$$

and by differentiation

$$p_i' = q_i'\, p_{i-1} + q_i\, p_{i-1}'$$

and carrying out the division by p_i, we obtain the following expression

$$\frac{p_i'}{p_i} = \frac{q_i'}{q_i} + \frac{p_{i-1}'}{p_i} \tag{5}$$

which relates the arithmetic inverses of the Newton-Raphson correction for the polynomials p_{i-1} and p_i, and their quotient q_i.

From the recursive expression, (3), we have that,

$$q_i' = -1 + b_i^2\, \frac{q_{i-1}'}{q_{i-1}^2}, \quad i = 2, 3, \ldots, n$$

and carrying out the division by q_i,

$$\frac{q_i'}{q_i} = \frac{-1}{q_i}\left(-1 + \frac{b_i^2}{q_{i-1}}\, \frac{q_{i-1}'}{q_{i-1}}\right), \quad i = 2, 3, \ldots, n$$

Using the notation

$$\Delta q_i = q_i'/q_i, \quad \Delta p_i = p_i'/p_i$$

the complete computation of,

$$\Delta p_n = p_n'(x)/p_n(x)$$

is expressed in the following equations,

$$\begin{aligned} q_1 &= a_1 - x \\ \Delta q_1 &= \Delta p_1 = -1/q_1 \\ &\left.\begin{aligned} q_i &= a_i - x - b_i^2/q_{i-1} \\ \Delta q_i &= (-1 + b_i^2/q_{i-1} * \Delta q_{i-1})/qi \\ \Delta p_i &= \Delta q_i + \Delta p_{i-1} \end{aligned}\right\} i = 2, \ldots, n \end{aligned} \tag{6}$$

where $\Delta q_i = q_i'(x_k)/q_i(x_k)$ and $\Delta p_i = p_i'(x_k)/p_i(x_k)$.

It is important to observe that in the computation of Δp_n using the formulae (6), the values $q_i, i = 1, \ldots, n$, are obtained, and its signs can be used to derive a method that combines bisection and Newton-Raphson's iteration. We will refer to this method as the *zeroinNR* algorithm.

So, given an interval $[\alpha, \beta]$ which contains an eigenvalue, and given an approximation $x_k \in [\alpha, \beta]$, the *zeroinNR* method will produce, in each step, an approximation x_{k+1} to the eigenvalue.

The *zeroinNR* method although not as fast as the QR method (according to [ralha93], *zeroinNR* is about two to four times slower than QR for the computation of all eigenvalues, depending on the characteristics of the spectrum) is consistently faster than simple bisection (generally, twice as fast) and retains the excellent numerical properties of simple bisection. In the present work we have introduced some modifications in the original *zeroinNR* method which actually make it faster. Numerical tests were carried out in a transputer based machine using double precision arithmetic. The methods were implemented in Occam 2, the official transputer's language.

We were able to find out the errors in the computed eigenvalues since we have used matrices for which analytic expressions for the eigenvalues are known. We conclude that, for small matrices, the accuracy of *zeroinNR* is comparable to that of the QR method as implemented in the MatLab system [matlab5], but as the size of the matrices grows, the *zeroinNR* method provides more accurate eigenvalues than QR method.

This can be appreciated in Figure 1, where the absolute erros of a matrix of size 1000, are plotted. We have used the tridiagonal matrix with $a_i = 2$ and $b_i = 1$ which eigenvalues are given by

$$\lambda_i = 2 + 2\cos\left(\frac{i\pi}{n+1}\right), \quad i = 1, \ldots, n.$$

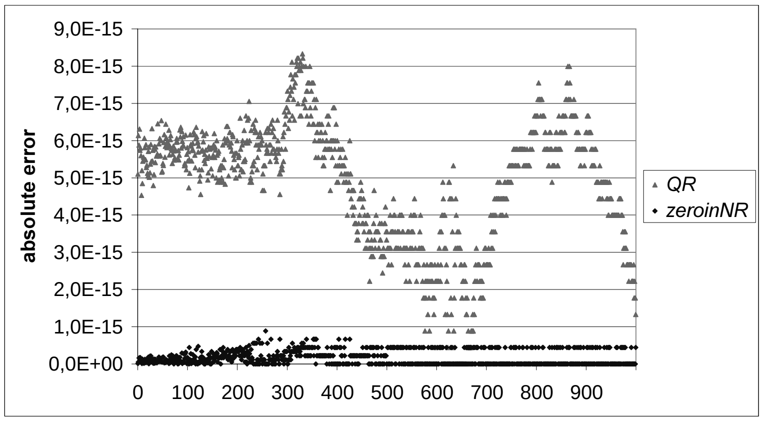

Fig. 1. Absolute errors of the eigenvalues of a matrix ($n = 1000$) computed with *zeroinNR* and QR.

3 An Efficient Parallel Algorithm: *farmzeroinNR*

The sequential *zeroinNR* method can be readily adapted to parallel processing since several disjoint intervals can be treated simultaneously by different processors. We have developed a parallel organization under a *processor farm* model and we will refer to this parallel implementation as the *farmzeroinNR* method. The typical architecture for this model is a pipeline of processors (workers), where the master sends tasks to workers and gets back the results produced.

Each time a processor produces two disjoint intervals containing eigenvalues, as the result of a bisection step, it keeps only one of them and passes back to the *master* the second interval which is kept in a queue of tasks. As soon as there is an available *worker* somewhere in the line, a new task is fed into the pipeline. Because of this mechanism, the algorithm achieves dynamic load balancing.

A dynamic distribution of tasks results from the fact already mentioned, as soon as a *worker* finishes a task, it will get a new one from the queue (which is managed by *master*), if such queue is not empty. The advantage of such dynamic workload distribution gets more important as n grows. It must be noted that, because some tasks take longer to finish than others, workers may not execute the same number of tasks, but will spend about the same time working.

The pseudocode to the *master* and *worker* processors are given in Algorithm 1 and Algorithm 2, respectively.

```
$eig \leftarrow 0$
for $k \leftarrow 1 .. p-1$ do
    { $worker[k] \leftarrow initial\_interval[k]$
$procs \leftarrow 0$
while $eig < n$ do
    case $input\_channel$ is_a
        $interval$ ⟶
            if $procs > 0$ do
                $output \leftarrow$ interval to workers
                $procs \leftarrow procs - 1$
            else ⟶
                { $queue \leftarrow interval$
        $eigenvalue$ ⟶
            $eig \leftarrow eig + 1$
            if queue not empty do
                { $output \leftarrow$ interval to workers
            else ⟶
                { $procs \leftarrow procs + 1$
send signal to terminate
```

Algorithm 1: *FarmzeroinNR master* processor pseudocode.

It must be noted that messages exchanged between the *master* and some *worker* in the pipeline need to be routed through the processors that lay in between. For the global performance of the system it is important that messages reach their destination as quickly as possible, therefore communication must be given priority over the computation.

To compute eigenvectors, once we have computed (selected) eigenvalues, we can use inverse iteration. Convergence is fast but eigenvectors associated with close eigenvalues may not be orthogonal. The LAPACK's routine sstein uses re-

```
while not receive signal to terminate do
    $interval \leftarrow input\_channel$
    if interval has more than one eigenvalue do
        $intervals \leftarrow$ bisection method(interval)
        $output \leftarrow$ intervals (to the master)
    else $\longrightarrow$
        $eig \leftarrow$ extract isolate eigenvalue
        $output \leftarrow$ eig (to the master)
```

Algorithm 2: *FarmzeroinNR worker* processor pseudocode.

orthogonalization of such eigenvectors. This does not solve the problem when there is a cluster with many close eigenvalues [demmel97, pg. 231], and recent progress on this problem appears to indicate that inverse iteration may be *repaired* to provide accurate, orthogonal eigenvectors without spending more than $O(n)$ flops per eigenvector. This will make bisection, or *zeroinNR* and *repaired* inverse iteration the algorithm of choice in all cases, no matter how many eigenvalues and eigenvectors are desired.

4 Performance Analysis

As already mentioned, a typical architecture for the processor farm model consists of a bidirectional array, forming a single pipeline (SP), with the master placed at one end of the array (Fig. 2).

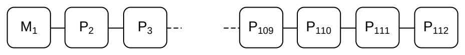

Fig. 2. Single pipeline, with 112 nodes.

It is predictable that as the number of workers increases, the communication overhead becomes more significant and processors that are further away from the master take longer to communicate with him. Furthermore, the activity in the links of the processors which are closer to the master grows with the number of processors and some congestion is to be expected if the computational complexity of each task is not sufficiently large. In an attempt to overcome the problems just mentioned, we decided to test the parallel algorithm with a modified topology, referred to as multiple pipeline (MP), which consists of seven pipelines, each one with 16 transputers; the masters of such pipelines are themselves connected in a single pipeline (Fig. 3).

At the beginning, the interval that contains all the eigenvalues is decomposed in 7 subintervals of equal width which are distributed among the different pipelines.

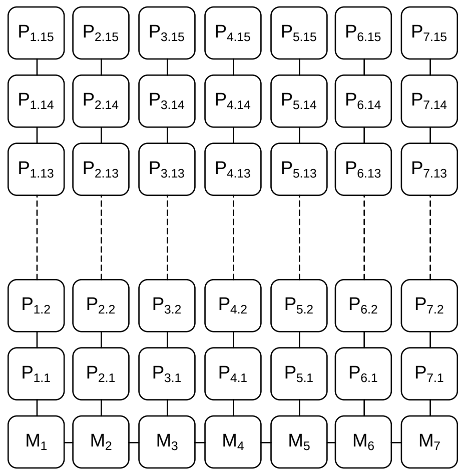

Fig. 3. Multiple pipeline, with 112 nodes.

Although this may reduce to some extent communication overhead and waiting times, it has an important disadvantage which is an eventual deterioration of the load balancing, which becomes critical when some of the subintervals contain a much larger number of eigenvalues than others. Therefore, the spectral distribution of the matrix is an important factor to be considered when comparing the performance of the SP and MP architectures. For this reason we have used four different types of matrices (see Table 1 where $a_i, i = 1, \ldots, n$, represents the diagonal elements $b_i, i = 1, \ldots, n-1$, represents the sub-diagonal elements) with different spectral distributions (see Figures 4, and 5),and sizes n ranging from one thousand to ten thousand.

Matrix	Elements	Analitical Formula
I	$a_i = a$ $b_i = b$	$\left\{ a + 2b\cos\frac{k\pi}{n+1} \right\}_{k=1}^{n}$
II	$a_1 = a - b$ $a_i = a,\ i = 1, \ldots, n$ $a_n = a + b$ $b_i = b,\ i = 1, \ldots, n$	$\left\{ a + 2b\cos\frac{(2k-1)\pi}{2n} \right\}_{k=1}^{n}$
III	$a_i = 0$ $b_i = \sqrt{i(n-i)}$	$\left\{ -n + 2k - 1 \right\}_{k=1}^{n}$
IV	$a_i = -[(2i-1)(n-1) - 2(i-1)^2]$ $b_i = i(n-i)$	$\left\{ -k(k-1) \right\}_{k=1}^{n}$

Table 1. Matrix Types.

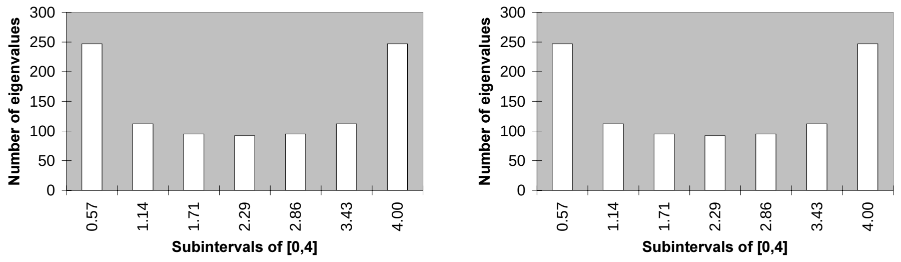

Fig. 4. Spectral distributions for matrix I (left) and matrix II (right), with $n = 1000$.

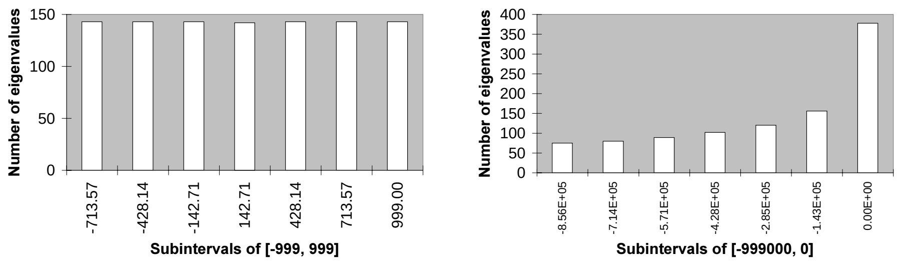

Fig. 5. Spectral distributions for matrix III (left) and matrix IV (right), with $n = 1000$.

We have computed the efficiency in the usual way, i.e.,

$$E = \frac{T_1}{112\ T_{112}}$$

where T_1 represents the time taken by a single transputer executing the sequential implementation of *zeroinNR*, and T_{112} is the time taken by *farmzeroinNR* with 112 processors. In Table 2 such ratios are given, representing by $E(\mathsf{SP})$ and $E(\mathsf{MP})$ the efficiency obtained for the single pipeline and multiple pipeline implementations, respectively.

	Matrix I		Matrix II		Matrix III		Matrix IV	
n	E(SP)	E(MP)	E(SP)	E(MP)	E(SP)	E(MP)	E(SP)	E(MP)
1000	55%	45%	60%	72%	61%	71%	60%	35%
5000	80%	56%	92%	80%	92%	89%	90%	38%
7000	93%	64%	91%	80%	91%	85%	94%	39%
10000	95%	56%	99%	85%	99%	91%	97%	39%

Table 2. Efficiency of *farmzeroinNR*, for matrices of type I, II, III and IV.

As it can be appreciated from this table, the MP implementation is less efficient than the SP implementation, except for the case of Matrices II and III of size $n = 1000$. In general, we have obtained better efficiency values with the SP architecture and we conclude that, for n sufficiently large, the communication overhead is not as important as the unbalance in the distribution of tasks introduced by the MP architecture. This is particularly clear in the case of matrix IV since for the larger values of n the efficiency for SP is about 2.4 times better than the efficiency for MP. The explanation for this can be found in Figure 5 (right side): the number of eigenvalues received by each one of the seven pipelines presents, in the case of matrix IV, a large variation, from about 70 to about 370. Another important aspect that must be taken into account is that in the MP implementation there are only 105 *workers*, since 7 processors are playing the role of *master*. However, even if we had used the modified formula

$$E = \frac{T_1}{105\, T_{112}}$$

to compute the efficiency for the MP implementation, the values produced in this way would still be lower than those obtained for the SP architecture in most cases.

5 Conclusions

We have carried out a parallel implementation of an efficient algorithm, dubbed *zeroinNR*, for the eigenvalue problem of a symmetric tridiagonal matrix, on a distributed memory system. The sequential *zeroinNR* method, although not as fast as QR, is consistently faster than simple bisection and retains the excellent numerical properties of this method. We have numerical evidence to support the claim that our method produces eigenvalues with smaller errors than those produced by QR. For the parallel implementation we used a farm model with two different topologies: a single pipeline (SP) of 112 processors and a multiple pipeline implementation (MP) consisting of seven pipelines, each one with 16 processors. The MP architecture reduces the communication overhead to some extent but is not able to retain fully the excellent load balancing of the SP implementation. This trade-off is not clear since it depends on the spectral distribution of each particular matrix. We have used matrices of different types to study this trade-off and conclude that for matrices sufficiently large, the parallel algorithm under the SP architecture performs better than the MP architecture. It must be emphasized that the parallel algorithm under the SP architecture is very efficient: for matrices of size $n = 10000$ we got efficiency values which are in all cases tested larger than 95%.

References

[wilkinson65] J. H. Wilkinson: The Algebraic Eigenvalue Problem. Oxford University Press, (1965).

[barth67] W. Barth and R. S. Martin and others: Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method. Num.Mathematics, **9** (1967) 386–393.

[barlow78] R. H. Barlow, D. J. Evans: A Parallel Organization of the Bisection Algorithm. The Computer Journal. **22, n°3** (1978).

[ralston78] A. Ralston and P. Rabinowitz: A First Course in Numerical Analysis. McGraw-Hill, (1978).

[parlett80] B. N. Parlett: The Symmetric Eigenvalue Problem. PrenticeHall, Englewood Cliffs, NJ, USA, (1980).

[cuppen81] J. J. Cuppen: A Divide and Conquer Method for the Symmetric eigenvalue problem. Numer. Math. **36, 177–195** (1981).

[bernstein84] H. J. Bernstein: An Accelerated Bisection Method for the Calculation of Eigenvalue of A Symmetric Tridiagonal Matrix. Numer. Math. **43, 153–160** (1984)).

[dongarra87] J. J. Dongarra and D. C. Sorense: A Fully Parallel Algorithms for the Symmetric Eigenproblem. SIAM J. SCI. STAT. COMPUT. **8, n°2** (March, 1987).

[phillipe87] S. S. Lo, B. Phillipe and A. Sameh: A Multiprocessor Algorithm for the Symmetric Eigenproblem. SIAM J. SCI. STAT. COMPUT. **8, 155–165** (1987).

[jessup90] I. C. F. Ipsen and E. R. Jessupe: Solving the Symmetric Tridiagonal Eigenvalue Problem on the Hypercube. SIAM J. SCI. STAT. COMPUT. **11, n°2, 203-229** (1990).

[kalambouskis90] T. Z. Kalambouskis: The symmetric tridiagonal eigenvalue problem on a transputer network. Parallel Computing. **15, 101–106**. North Holland (1990).

[ralha90] R. Ralha: Parallel Computation of Eigenvalues and Eigenvectors using Occam and Transputers. PhD thesis, University of Southampton, (1990).

[baserman92] A. Baserman and P. Weidner: A Parallel Algorithm dor Determining all Eigenvalue of Large Real Symmetric Tridiagonal Matrices. Parallel Computing. **18, 1129–1141**.(1920).

[scalapack97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker and R. C. Whaley: ScaLAPACK User's Guide. Software, Enviroements and Tools 4. SIAM, Philadelphia, PA, (1997).

[lapack95] E. Anderson, Z. Bai, C. Bischop, J. Demmel, J. Dongarra, S. Hammarling, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and S. Sorensen : LAPACK User's Guide (2nd edition). SIAM, Philadelphia, PA, (1995).

[demmel93] J. Demmel, M. Heath and H. van der Vorst: Parallel Numerical Linear Algebra. In A. Iserles, Acta Numerica, Volume 2. Cambridge University Press, UK, (1993).

[ralha93] R. Ralha: Parallel Solution of the Symmetric Tridiagonal Eigenvalue Problem on a Transputer Network, Proceedings of the Second Congress of Numerical Methods in Engineering, Spanish Society of Numerical Methods in Engineering, (1993).

[badia96] José Manuel Badía Contelles: Algoritmos Paralelos para el Cálculo de los Valores Propios de Matrices Estructuradas. PhD thesis, Universidad Politecnica de Valencia, (1996).

[demmel97] J. Demmel: Applied Numerical Linear Algebra. SIAM, Philadelphia, (1997).

[matlab5] Using MatLab, The Math Works Inc. (1999).

[forjaz2000] Maria Antónia Forjaz: Algoritmos Paralelos para o Cálculo de Valores e Vectores Próprios em Sistemas de Multiprocessadores de Memória Distribuída. PhD thesis, Universidade do Minho, (2000) (submitted).