Static versus dynamic heterogeneous parallel schemes to solve the symmetric tridiagonal eigenvalue problem

Miguel O. Bernabeu
Dept. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera S/N. 46022 Valencia
Spain

Antonio M. Vidal
Dept. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera S/N. 46022 Valencia
Spain

Abstract: Computation of the eigenvalues of a symmetric tridiagonal matrix is a problem of great relevance. Many linear algebra libraries provide subroutines for solving it. But none of them is oriented to be executed in heterogeneous distributed memory multicomputers. In this work we focus on this kind of platforms. Two different load balancing schemes are presented and implemented. The experimental results show that only the algorithms that take into account the heterogeneity of the system when balancing the workload obtain optimum performance. This fact justifies the need of implementing specific load balancing techniques for heterogeneous parallel computers.

Key–Words: Symmetric tridiagonal eigenvalue problem, heterogeneous parallel computing, load balancing

1 Introduction

Computation of the eigenvalues of a symmetric tridiagonal matrix is a problem of great relevance in numerical linear algebra and in many engineering fields, mainly due to two reasons: first, this kind of matrices arises in the discretisation of certain engineering problems and secondly, and more important, this operation is the main computational kernel in the computation of the eigenvalues of any symmetric matrix when tridiagonalisation techniques are used as a previous step.

Nowadays, there are a large amount of eigenvalue computation algorithms that exploit the tridiagonality properties of the matrix. Four main techniques can be found in the specialised literature to solve this problem: QR iteration, homotopy method, bisection and multisection methods and divide and conquer techniques. None of them is clearly superior to the rest since every one presents exclusive advantages, for example: computing all matrix eigenvalues or just a defined subset of them, precision of the results or simultaneous eigenvector computation. See [1] for an exhaustive comparison.

In [2] Badía and Vidal proposed two parallel bisection algorithms for solving the symmetric tridiagonal eigenproblem on distributed memory multicomputers, including a deep study of the two step bisection algorithm. In that work, special emphasis was put in load balancing since this is the main difficulty when parallelising the bisection algorithm for the computation of the eigenvalues of a symmetric tridiagonal matrix.

Both, ScaLAPACK subroutines and those presented in [2] achieve good performance in homogeneous distributed memory multicomputers. We can define an homogeneous distributed memory multicomputer as a distributed memory multicomputer where all the processors are equal in computing and communication capabilities. In this work we focus on heterogeneous distributed memory multicomputers, those formed by processors with different computing and communication capabilities. These kind of platforms are expected to be the best solution in order to achieve great performance/cost ratio, to reuse obsolete computational resources or simply to obtain the maximum performance from several powerful computers with different architectures but able to work coordinately.

Parallel numerical linear algebra libraries, like ScaLAPACK, do not take into account the possible heterogeneity of the hardware and, for that reason, their performance considerably decrease when working in this kind of systems. There is a big gap in this context and, nowadays, the design of parallel linear algebra libraries for heterogeneous architectures is a must. The study of computational kernels and basic algorithms is the previous step to achieve this objec-

tive. In this work we present algorithms for the computation of the eigenvalues of a symmetric tridiagonal matrix which attain good performance in heterogeneous multicomputer, analysing different load balancing strategies and different problem instances.

The rest of the paper is organised as follows: in the next section we present the mathematical description of the problem and the sequential algorithm implemented in the solution. Section 3 describes the heterogeneous computational model used. In Section 4, the different parallel schemes used are described. In Section 5, experimental results are presented. Finally, the main conclusions of the work are given in Section 6.

2 Problem description and proposed solution

2.1 Problem definition

Let T be a symmetric tridiagonal matrix $T \in \mathbb{R}^{nxn}$, defined as follows

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

The eigenvalues of T are the n roots of its characteristic polynomial $p(z) = det(zI - T)$. The set of these roots is called the spectrum and is denoted by $\lambda(T)$.

It is possible to compute specific eigenvalues of a symmetric matrix by using the LDL^T factorization and exploiting the Sylvester inertia theorem. If

$$A - \mu I = LDL^T \quad A = A^T \in R^{nxn}$$

is the LDL^T factorization of $A - \mu I$ with $D = diag(d_1, \ldots, d_n)$, then the number of negative d_i equals the number of $\lambda(A)$ that are smaller than μ [6].

Sequence $(d_1, \ldots, d_n)$ can be computed by using the following recurrence, where $d_i = q_i(c), i = 1, \ldots, n$ for a given c:

$$\begin{cases} q_0(c) = 1, \quad q_1(c) = a_1 - c \\ q_i(c) = (a_i - c) - \frac{b_{i-1}^2}{q_{i-1}(c)} \quad i : 2, 3, \ldots, n \end{cases}$$

Thanks to this result it is possible to define a function $neg_n(c)$ that for any value of c computes the number of eigenvalues smaller than c. With this function it is easy to implement a bisection algorithm that isolates eigenvalues of T.

The bisection algorithm needs, for initialisation purpose, an initial interval $[a, b]$ which contains all the eigenvalues of matrix T. The Gershgorin circle theorem can be used to calculate it.

So, based on the initial interval $[a, b]$ and through the bisection algorithm is possible to isolate the m subintervals $]lb_i, ub_i]$ which contain a number of eigenvalues $v \leq max_v al$ and will be used as the input for the next step of the algorithm.

The importance of this step lies in the fact that it will help us to discard parts of the real line where no eigenvalue is located and therefore to reduce the number of iterations of the extraction methods used in the following step. In addition, the isolation step will be used to balance the workload of the parallel algorithms presented in Section 4, since the initial problem is divided into m subproblems with similar workload, susceptible to be solved in parallel.

The second step of the algorithm gets as input the m subintervals $]lb_i, ub_i]$ obtained in the previous step to compute m eigenvalues of matrix T.

There are several alternatives for the eigenvalues extraction:

1. To apply again the bisection method described in previous subsection.
2. To use a fast convergence method like Newton or Laguerre [1].
3. To use standard computational kernels like LAPACK and let them choose the best method for eigenvalue extraction. These subroutines are expected to efficiently implement the sequential solution of the problem.

In this work, we have chosen to use LAPACK subroutines, specifically the driver subroutine $dstevr$ which can compute the eigenvalues of a tridiagonal symmetric matrix contained into an interval $]vl, vu]$.

2.2 Test matrices

The bisection algorithm above described is problem-dependent, because the number of iterations to reach each eigenvalue with a specified precision could be different. Thus, the behaviour of the algorithm depends on the distribution of the eigenvalues along the spectrum. In addition, the presence of clusters of eigenvalues or hidden eigenvalues considerably increases the extraction time, see [2].

Therefore, in order to perform a correct experimental analysis of the algorithms implemented, a suitable set of test matrices should be chosen. In our case,

we have chosen two kinds of matrices that present different eigenvalue distribution characteristics, so it can affect the performance of the algorithm.

Table 1 shows matrices used.

3 Heterogeneous computational model

3.1 PC model description

The following theoretical machine model is called PC (Power-Communications). Let be a set of p processors interconnected via a communication network. This model is expected to evaluate the power of each processor as well as the communication capabilities of the network.

First of all, a power vector P_t that summarizes the relative power of each processor (related to the global machine power) is defined. This relative power depends on the operation and on the problem size, so there is a vector P_t for each pair of operation and problem size. However, we consider here that the power vector does not depend on time.

Secondly, the communication model used defines the time needed to send n bytes from processor i to processor j as $T_{ij}(n) = \beta + n\tau$, where β stands for network latency and τ is the bandwidth inverse. In order to summarize the model two matrices B_c and T_c are defined. The (i, j) entry of each matrix represents the β or τ applicable to the communication from processor i to j. We consider also here that both matrices do not depend on time.

3.2 Model implementation

The cluster used to evaluate the model and to run the parallel algorithms consists of four machines with six processors. They are:

- Intel Pentium IV at 3.0 GHz with 1 MB of cache and 1 GB of main memory.
- Intel Xeon two-processor at 2.2 GHz with 512 KB of cache and 4 GB of main memory.
- Intel Xeon two-processor at 2.2 GHz with 512 KB of cache and 4 GB of main memory.
- Intel Pentium IV at 1,6 GHz with 256 KB of cache and 1GB of main memory.

A Gigabit Ethernet network is used to interconnect the six machines with 1 Gbit/s of theoretical bandwidth. Note that communications between each CPU of the two-processors boards have been evaluated with the same model.

Despite the algorithms have been implemented and evaluated in this machine, they only depend on the theoretical model. So, they can be executed in any other distributed memory multicomputer with similar predictable performance, under the condition of being evaluated with the previous model.

3.3 Evaluation

Tables 2, 3, 4 and 5 show the result of the evaluation of the cluster described before, following the PC model. Tables 2 and 3 show the power vector P_t obtained in the computation of eigenvalues of uniform spectrum matrices and Wilkinson matrices with different sizes. Tables 4 and 5 show the matrices B_c and T_c obtained in the same experiments.

As it can be observed, the variations of the vector power P_t with the size of the problem are very small. This is a characteristic of this problem, because only two vectors (main diagonal and subdiagonal) have to be stored in memory. This may be different with problems which require more memory space.

4 Heterogeneous parallel schemes

4.1 Available alternatives

Among different techniques proposed in the literature (see [2]) to parallelise the bisection method, probably the most effective is the computation of groups of eigenvalues simultaneously in different processors. However, this division could not be arbitrarly done, since the performance of the parallel algorithm will be determined by the correctness of the load balancing.

The problem of the load balancing is already known in homogeneous parallel computing but it affects more to the performance of the parallel algorithms when the power and the communication capabilities of the processors are not equal.

Different approaches can be taken to solve the problem in our case:

1. To ignore the difference of power and communication capabilities and perform an equitable workload distribution. With this approach, the spectrum is divided into subintervals containing the same number of eigenvalues.
2. Based on a heterogeneous machine model, like the one presented on Section 3, perform a distribution of the workload proportional to the power and communication features of each processor. Now, the spectrum is divided into subintervals

Kind	Elements	Eigenvalues
Uniform spectrum matrices	$a_i = 0$ $b_i = \sqrt{i(n-i)}$	$\{-n+2k-1\}_{k=1}^{n}$ Uniformly distributed
Wilkinson matrices	$a_i = \begin{cases} \frac{m}{2} - i + 1 & i : 1, \ldots, \frac{m}{2} \\ i - \frac{m}{2} & i : \frac{m}{2} + 1, \ldots, m \end{cases}$ $m = \begin{cases} n & with\ even\ n \\ n+1 & with\ odd\ n \end{cases}$	Most eigenvalues grouped in clusters of two.

Table 1: Test Matrices

P0	P1	P2	P3	P4	P5
0.2245	0.1740	0.1692	0.1605	0.1598	0.1120

Table 2: Relative power vector P_t for uniform spectrum matrices eigenvalue computation

P0	P1	P2	P3	P4	P5
0.2219	0.1747	0.1681	0.1635	0.1607	0.1113

Table 3: Relative power vector P_t for Wilkinson matrices eigenvalue computation

	P0	P1	P2	P3	P4	P5
P0	0	2.5074E-05	4.6149E-05	4.5745E-05	4.0809E-05	5.1629E-05
P1	2.4774E-05	0	4.6042E-05	4.5852E-05	4.0348E-05	5.317E-05
P2	4.6268E-05	4.6073E-05	0	2.4727E-05	4.2951E-05	5.2753E-05
P3	4.564E-05	4.5583E-05	2.4898E-05	0	4.1091E-05	5.2756E-05
P4	4.0829E-05	4.0706E-05	4.1424E-05	4.0211E-05	0	7.8713E-05
P5	5.0371E-05	4.9166E-05	5.1836E-05	4.6695E-05	1.0234E-4	0

Table 4: Latency matrix B_c (s)

	P0	P1	P2	P3	P4	P5
P0	0	6.4244E-09	1.4045E-08	1.1898E-08	2.1316E-08	5.5421E-08
P1	6.3307E-09	0	1.2879E-08	1.3066E-08	2.1954E-08	5.5108E-08
P2	1.2966E-08	1.3761E-08	0	6.3347E-09	2.0886E-08	5.2581E-08
P3	1.2548E-08	1.3327E-08	6.3294E-09	0	2.0014E-08	5.4863E-08
P4	2.1369E-08	2.1076E-08	2.269E-08	2.0776E-08	0	1.0859E-07
P5	2.1369E-08	5.4601E-08	5.6684E-08	5.1962E-08	1.0991E-07	0

Table 5: Inverse bandwidth matrix T_c (s)

with a number of eigenvalues proportional to the relative power of each processor.

3. To implement a dynamical workload distribution algorithm based on the master-slave programming paradigm. With this approach, the spectrum is divided into a number of subintervals $m \gg p$ that are assigned to the processors on demand.

4.2 Implemented algorithms

Based on the algorithm presented in Section 2 and on the approaches for the solution of the load balancing problem described before, we have implemented a sequential and five parallel algorithms.

Sequential algorithm. A1: This version implements sequentially the bisection algorithm described in

Section 2.

ScaLAPACK algorithm. A2: This version computes the eigenvalues of the matrix T by means of calling ScaLAPACK subroutine $pdstebz$. This subroutine uses the bisection method for isolating and extracting the eigenvalues.

Static algorithm. A3: In this version we statically assign to the processor $i = 0, \ldots, p-1$ the calculation of eigenvalues $[i\frac{n}{p}+1, (i+1)\frac{n}{p}]$ according to an ascending classification of them.

This algorithm have been implemented by means of p concurrent calls to the LAPACK subroutine $dstevr$ that takes as input parameters two integers that define the subset of desired eigenvalues.

We have used this algorithm to model the cluster and to obtain the data shown in Section 3.3. Also it could be a better comparative reference for the rest of the algorithms as it has been implemented with a similar style and without the optimisation of ScaLAPACK library.

Proportional Static algorithm. A4: This version uses a similar strategy to the one described in the previous algorithm but the number of eigenvalues assigned to each processor depends on its relative power.

Dynamic algorithm. A5: This version implements, in parallel, both steps of bisection algorithm described in Section 2. The first step of the algorithm consists of dividing the interval $[a, b]$ which contains all the eigenvalues into p subintervals of length equal to $\frac{b-a}{p}$. Each of this subintervals is assigned to a processor that applies the isolation step described in 2. Finally, the results are gathered by the master process.

In order to fulfil the $m \gg p$ constraint, parameter max_val of the isolation algorithm has been adjusted to 1. Thus m is equal to problema size n.

Finally, the extraction step has been implemented with the master-slave technique described before. Note that the most powerful processor has been chosen to allocate the master process. The master process also assigns intervals to this most powerful processor. In this way it acts as a slave process too, in order to take advantage of its greater power.

Modified Dynamic algorithm. A6: This version is similar to the previous one, but the $m \gg p$ constraint has been relaxed. Instead of $max\_val = 1$, we have assigned values between 1 and 100. We have done it for two reasons; first, to diminish the drawbacks produced by clusters of eigenvalues in the isolation step and, second, to study the impact in the execution time of the number of eigenvalues computed in the extraction step.

5 Experimental Analysis

Tables 6, 7 and 8 show the execution time of the six algorithms presented. For both kind of matrices, the Proportional Static algorithm (A4) and the Modified Dynamic algorithm (A6) present the smaller execution times, followed by ScaLAPACK (A2) and by Dynamic algorithm (A5) that present similar results. Finally the Static algorithm (A3) has the poorest performance of all tested algorithms.

n	Uniform	Wilkinson
5000	20.14	8.63
7000	38.80	16.50
9000	63.35	27.07
11000	94.13	39.94
13000	130.69	55.38
15000	172.80	74.31

Table 6: Execution time (s), on P0, for the sequential algorithm (A1) on both kinds of matrices

n	A2	A3	A4	A5	A6
5000	5.15	6.39	4.44	5.21	4.43
7000	9.98	12.41	8.61	9.78	8.54
9000	16.29	20.22	13.94	15.88	13.95
11000	24.45	30.15	20.66	23.68	20.78
13000	33.85	42.12	28.76	33.00	28.68
15000	45.67	56.50	37.96	43.72	38.04

Table 7: Execution time (s) for the 5 parallel algorithms on uniform spectrum matrices

Tables 9 and 10 show the speedup of the two best parallel versions, A4 and A6, with regard to the ScaLAPACK version (algorithm A2). Both algorithms present similar performance, with a slightly better speedup when they are applied on Wilkinson matrices.

6 Conclusions

In the present work one sequential and five parallel algorithms have been presented for the extraction of the

n	A2	A3	A4	A5	A6
5000	2.48	2.98	2.03	3.48	2.08
7000	4.70	5.68	3.86	6.72	3.92
9000	7.68	9.36	6.33	11.15	6.35
11000	11.45	13.96	9.38	17.42	9.45
13000	15.95	19.63	13.04	24.58	13.15
15000	21.21	25.99	17.39	32.85	17.35

Table 8: Execution time (s) for the 5 parallel algorithms on Wilkinson matrices

n	5000	7000	9000	11000	13000	15000
A4	1.16	1.16	1.17	1.18	1.18	1.2
A6	1.16	1.17	1.17	1.18	1.18	1.2

Table 9: Speedup of algorithm A4 and A6 with regard to algorithm A2 (ScaLAPACK) on uniform spectrum matrices

n	5000	7000	9000	11000	13000	15000
A4	1.22	1.22	1.21	1.22	1.22	1.22
A6	1.2	1.2	1.21	1.21	1.21	1.22

Table 10: Speedup of algorithm A4 and A6 with regard to algorithm A2 (ScaLAPACK) on Wilkinson matrices

eigenvalues of a symmetric tridiagonal matrix. Three of them have been specifically designed to be executed in heterogeneous distributed memory multicomputers.

The parallel algorithms implemented are based on the bisection method. Basically, two strategies have been used: a static strategy, trying to get a good load balancing through a distribution of processes proportional to the power of processors, and a dynamic strategy, based on the master-slave paradigm. For the implementation of the dynamic algorithms we have used a bisection algorithm with two steps: isolation and extraction. The bisection technique has been chosen to implement the isolation step. For the extraction step LAPACK subroutines have been used.

Finally, these are the main conclusions of the work:

- The algorithms that take into account the heterogeneity of the system when balancing the workload (A4, A5 and A6) always obtain better execution time than those that do not that (A2 and A3). This fact justifies the need of implementing specific load balancing techniques for heterogeneous architectures.
- The execution time of the Dynamic algorithm (A5) is always larger than that of the Modified Dynamic algorithm (A6). This is due to the extra effort made in the isolation step. In addition, the presence of clusters of eigenvalues (Wilkinson matrices) increases the number of iterations needed in this step.
- The fact that Proportional Static algorithm (A4) and Modified Dynamic algorithm (A6) present almost the same execution time validates both strategies to get a good load balancing. However, the effort necessary to reach a good workload balance in A4 (compute the power vector for each kind of matrix and problem size) could be a huge amount of extra work. Therefore, the authors consider the Modified Dynamic algorithm (A6) is the most suitable solution for heterogeneous environments.
- The fact that algorithms A4 and A6 present better performance than ScaLAPACK subroutines justifies the need of designing and implementing numerical linear algebra libraries for heterogeneous parallel architectures.

References:

[1] J.M.Badía, A.M.Vidal. “Cálculo de los valores propios de matrices tridiagonales simétricas mediante la iteración de Laguerre”. Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería. Vol. 16, num 2, pp 227–149 (2000)

[2] J.M.Badía, A.M.Vidal. “Parallel bisection algorithms for solving the symmetric tridiagonal eigenproblem”. In the book High Performance Algorithms for Structured Matrix Problems from the series Advances in the Theory of Computation and Computational Mathematics. Nova Science Publishers (1998)

[3] J. L. Bosque, L. P. Perez. “HLogGP: a new parallel computational model for heterogeneous clusters”. CCGRID 2004: 403-410 (2004)

[4] Anderson, E., Bai, Z., Bishof, C., Demmel, J., Dongarra, J.; LAPACK User Guide; Second edition. SIAM (1995)

[5] Blackford, L.S., Choi, J., Clearly, A.; ScaLAPACK User’s Guide. SIAM (1997)

[6] G. H. Golub, C. F. van Loan; Matrix Computations; John Hopkins University Press, 3rd edition (1996)