Solving the Inverse Toeplitz Eigenproblem Using ScaLAPACK and MPI*

J.M. Badía[1] and A.M. Vidal[2]

[1] Dpto. Informática., Univ Jaume I. 12071, Castellón, Spain.
badia@inf.uji.es
[2] Dpto. Sistemas Informáticos y Computación. Univ. Politécnica de Valencia.
46071, Valencia, Spain.
avidal@dsic.upv.es

Abstract. In this paper we present a parallel algorithm for solving the inverse eigenvalue problem for real symmetric Toeplitz matrices. The algorithm is implemented using the parallel library ScaLAPACK, based on the communication model MPI. Experimental results are reported on a SGI Power Challenge and on a cluster of PC's connected by a Myrinet network. The results show that the algorithm is portable and scalable.

1 Introduction

The sequential and parallel linear algebra libraries LAPACK and ScaLAPACK, respectively, have become basic tools to tackle any kind of numerical problems in Physics and in Engineering.

ScaLAPACK [1] is a library of linear algebra routines for message-passing distributed memory computers. As in LAPACK, ScaLAPACK routines are based in block-oriented algorithms to minimize the data movement between different levels of the memory hierarchy. The main ScaLAPACK components are BLACS and PBLAS. BLACS is a message-passing library, which includes a set of routines to perform the communications that appear in parallel linear algebra algorithms. There are implementations of the BLACS library using MPI [2], PVM and other message-passing environments. PBLAS includes parallel versions of Level 1, 2 and 3 BLAS routines implemented for distributed memory using the message-passing programming model.

In this work, we analyze the parallel solution of a complex linear algebra problem using the ScaLAPACK library and the message-passing environment MPI. The problem is the inverse eigenvalue problem for Real Symmetric Toeplitz (RST) matrices.

* This paper was partially supported by the project CICYT TIC96-1062-C03: "Parallel Algorithms for the computation of the eigenvalues of sparse and structured matrices".

J. Dongarra et al. (Eds.): PVM/MPI'99, LNCS 1697, pp. 372-379, 1999.

Let $t = [t_0,\ t_1,\ \cdots,\ t_{n-1}]$ where t_i are real numbers. We say $T(t)$ is a real symmetric Toeplitz matrix, generated by *t*, if

$$T(t) = \left(t_{|i-j|}\right)_{i,j=1}^{n}.$$

This type of matrices appears in relation with the solution of several problems in different areas of Physics and Engineering. Moreover, the problem is specially appropriate for a parallel implementation due to its big computational cost.

In this paper we present a parallel implementation for solving the previous problem. Moreover, we report the experimental performance on two machines, a SGI Power Challenge multiprocessor and a cluster of PC's linked by a Myrinet network. In the last case, we compare the experimental results obtained with two different versions of the MPI environment. These versions have been specifically implemented for this gigabit network. A more detailed description of the method and of the experimental results can be found in [3].

The rest of the paper has the following structure: in section 2 we briefly describe the problem to be solved; in section 3 we present the sequential method utilized; the parallelization problem is studied in section 4, and last, in section 5 we report the obtained experimental results.

2 The Inverse Eigenvalue Problem

Let us denote of the eigenvalues of the matrix $T(t)$ by $\lambda_1(t) \le \lambda_2(t) \le \ldots \le \lambda_n(t)$.

We say an eigenvector, $x = [x_1,\ x_2,\ \ldots,\ x_n]$, of matrix $T(t)$ is *symmetric* if

$$x_j = x_{n-j+1},\quad 1 \le j \le n$$

and is *skew-symmetric* if

$$x_j = -x_{n-j+1},\quad 1 \le j \le n.$$

Moreover, we will call *even* (*odd*) eigenvalues to those eigenvalues associated with the symmetric (skew-symmetric).

In [4] it is shown that if $r = \lceil n/2 \rceil$, $s = \lfloor n/2 \rfloor$, and *T* is a RST matrix of order *n*, then $\Re^n$ has an orthonormal basis consisting of r symmetric and s skew-symmetric eigenvectors of *T*.

In addition, there exist a set of properties of Toeplitz matrices which allows us to calculate the odd and even eigenvalues separately. This calculation is performed from *T* by constructing two matrices with half order each, thus providing a substantial saving in the computational time.

In [5] a method for solving the inverse Toeplitz eigenvalue problem, equivalent to the Newton method, is proposed. This algorithm is improved in [6] by adequately exploiting the previous properties of Toeplitz matrices.

Now, we state the problem to be solved in similar terms to those in [6]:

Given n real numbers $\mu_1 \le \mu_2 \le \ldots \le \mu_r$ and $\nu_1 \le \nu_2 \le \ldots \le \nu_s$, find an n-vector t such that

$$\mu_i(t) = \mu_i,\ 1 \le i \le r,\ \textit{and}\ \ \nu_i(t) = \nu_i,\ 1 \le i \le s,$$

where the values $\mu_i(t)$ and $\nu_i(t)$ are, respectively, the even and odd eigenvalues of the RST matrix $T(t)$.

3 Sequential Algorithm

In this section we briefly describe algorithmically, the sequential method proposed in [6] for solving the inverse eigenvalue problem for a RST matrix.

First, we denote by $p_1(t)$, …, $p_r(t)$ the symmetric eigenvectors of $T(t)$, and by $q_1(t)$, …, $q_s(t)$ the skew-symmetric eigenvectors. In addition we denote the target spectrum as

$$\Lambda = [\mu_1,\ \mu_2,\ \ldots, \mu_r,\ \nu_1,\ \nu_2,\ \ldots, \nu_s], \tag{1}$$

where the even and odd spectra have been separated and reordered in increasing order.

Let t^0 be an n-vector, and let Λ be the target spectrum such as it is defined in (1). The basic idea of the algorithm is the following: by using t^0 as an starting generator, compute a sequence t^m, m=1, 2,..., as the solution of the equations

$$\begin{aligned} p_i\left(t^{m-1}\right)^T T\left(t^m\right) p_i\left(t^{m-1}\right) &= \mu_i, \quad 1 \le i \le r \\ q_j\left(t^{m-1}\right)^T T\left(t^m\right) q_j\left(t^{m-1}\right) &= \mu_j, \quad 1 \le j \le s. \end{aligned} \tag{2}$$

The solution of these equations can be rewritten as the solution of a linear system of size $n=r+s$.

If we denote by $\Lambda(t)$ the spectrum of $T(t)$ and by Λ the target spectrum, we can express the distance between both spectra as

$$\sigma(t;\Lambda) = \left\| \Lambda(t) - \Lambda \right\|_2. \tag{3}$$

We will say that the above method has converged if

$$\sigma\left(t^m;\Lambda\right) < \varepsilon_0. \tag{4}$$

for some integer m, where ε_0 defines the desired precision for the result.

We will define a basic iterative algorithm `alg1` with the following specification:

```
ALGORITHM [D',t,fails] = alg1(t₀, n, D, eps₀)
```

Given an initial generator `t₀` of size `n`, a target spectrum `D`, and a real value `eps₀` which defines the desired precision in (4), `alg1` returns the generator `t` of a RST matrix, its computed spectrum `D'` and a boolean value `fails` which indicates if the algorithm has failed to converge (5).

In each iteration of the previous algorithm, the initial task is to construct the matrix of the linear system, which allows us to solve (2). This is performed from the eigenvectors of the RST matrix computed in the previous iteration. Then, the linear system is solved, and its solution provides a new generator for RST matrix. The spectrum of this matrix is computed and the convergence is reached if (4) is verified.

As the Newton method does not converge globally, the algorithm `alg1` does not necessary converge to the solution of the problem. We say that the Algorithm 1 has failed if

$$\sigma\left(t^{k};\Lambda\right)\geq\sigma\left(t^{k-1};\Lambda\right)\geq\varepsilon_{0}. \tag{5}$$

In [6] an improvement of `alg1` is proposed. The algorithm is organized in two stages: During the first stage, if the algorithm fails, a new target spectrum such as

$$(1-\rho)\Lambda+\rho\Lambda\left(t^{k-1}\right) \tag{6}$$

is proposed, where ρ is a real value in (0,1). During this first stage the value of ρ is modified until the convergence to the target spectrum is reached linearly, with a less restrictive stopping criterion. From this new point the second stage starts and the algorithm `alg1` is utilized in order to converge to the target spectrum quadratically. This algorithm can be specified as follows:

Algorithm 2. Given an initial generator t_0 of size `n`, a target spectrum `D`, real values eps_0, eps_1 and `alfa`, associated to the convergence criteria of the linear and quadratic stage, and a real value `dro`, the following algorithm returns the generator `t` of a RST matrix, its computed spectrum `D'` and a boolean value `fails` which indicates if the algorithm failed to converge.

```
ALGORITHM [D',t,fails] = alg2(t_0,n,D,eps_0,eps_1,alfa,dro)
BEGIN
  [D', t, fails] = alg1(t_0, n, D, eps_0)
  IF fails THEN  (* linear stage *)
    [D(t_0), P] = compute_spectrum(t_0, n)
    ro = 0.1; error = ||D(t_0) - D||
    WHILE (error > eps_1) AND (ro < 1) DO
      Dmod = (1 - ro) * D + ro * D(t_k)
      [D(t_k+1), t_k+1, fails] = alg1(t_k, n, Dmod, alfa*error)
      IF fails THEN
        ro = ro + dro; t_k = t_0
      ELSE
        t_k = t_k+1
      ENDIF
      error = ||D(t_k) - D||
    ENDWHILE
    IF (ro > 1) THEN
      fails = TRUE
    ELSE  (* quadratic stage *)
```

```
      [D', t, fails] = alg1(t_k, n, D, eps_0)
    ENDIF
  ENDIF
END.
```

In the previous algorithm the subroutine `[D(t), P] = compute_spectrum(t, n)` returns the eigenvalues `D(t)` and eigenvectors `P` of a matrix $T(t)$ of size *nxn*. This subroutine exploits the properties mentioned in section 2, halving the cost of computing the spectrum, and maintains separated the even and odd spectra.

3 Parallel Algorithm

In order to parallelize the above method we have used the numerical linear algebra library ScaLAPACK. We try to obtain a portable and scalable algorithm for distributed memory systems. We have also used the message-passing interface MPI to exploit the excellent implementations of this environment. The parallel algorithm is implemented in a mesh of processors using an SPMD model, and the matrices are distributed using a block cyclic scheme.

Algorithm 2 is based on four subroutines. The first subroutine computes the spectrum of a RST matrix using the spectra of two smaller symmetric matrices. To perform this task we have used the ScaLAPACK routine `PDSYEV`. As the eigenvectors of the two matrices are independently distributed in the mesh, we have redistributed them in order to have a global distribution of the eigenvectors of the full matrix.

The two first parts in figure 1 show that the spectrum of a matrix of size 17, for example, has a different distribution if we distribute the full matrix or if we distribute separately the even and odd spectra. To obtain a global distribution by concatenating both spectra in each processor, we have to perform a redistribution of the eigenvectors. This is performed minimizing the communications, by sending the last odd eigenvectors from the last processors in order to complete the same number of blocks of the even spectrum in all processors. After performing this redistribution, the eigenvectors location is reported in the lower part of figure 1. We have also to redistribute the eigenvalues to maintain the correspondence during all the iterative process.

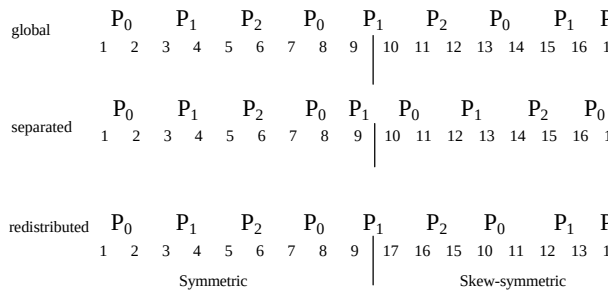

Fig. 1. Global, separated and redistributed odd and even spectra.

The second routine utilized in our parallel algorithm constructs the coefficient matrix of the linear system from the eigenvectors of the TRS matrix. We gather all the eigenvectors in the first row of processors in order to ease the construction of the matrix. Then, we scatter the constructed matrix among all processors in order to solve the system of equations.

Finally, the third and fourth basic routines utilized in our parallel algorithm are the ScaLAPACK routines `PDGETRF` and `PDGETRS`. These routines solve in parallel a general linear system of equations. A complete scheme of computations and communications in the algorithm is reported in figure 2.

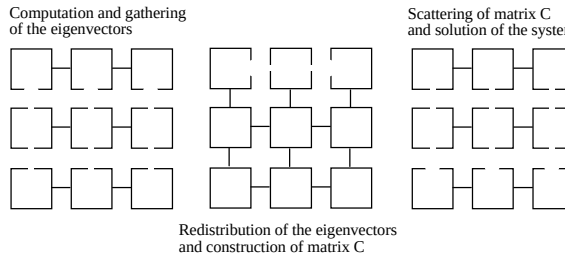

Fig. 2. Computations and communications in the parallel algorithm.

We have performed all communications using the BLACS routines and the auxiliary matrix redistribution routines included in ScaLAPACK, [1]. In both cases all the communications are based on the MPI environment.

4 Experimental Analysis

4.1 Sequential Algorithm

In the case of the sequential version of algorithm 2, we have performed a set of tests similar to the one presented in [6]. In our case the tests have been developed with a Fortran version of the program and with larger matrices (n=1200). In all performed tests the algorithm converges, except when the spectrum contains large clusters of eigenvalues. In this case a heuristic convergence strategy is proposed in [6].

We have tested the algorithm with three classes of starting generators t^0 and the best generators are of the class proposed in [6]. Regarding to this, we have observed that using a slight modification of the starting generator used by Trench the convergence always occurs during the quadratic stage of the algorithm or from a value of ρ=0.1 obtained during the linear stage.

We have completed the analysis of the sequential case testing the behaviour of the algorithm with 15 classes of spectra with different distributions of the eigenvalues. The algorithm converged in all cases except when large clusters of eigenvalues are present. A more detailed description of the algorithms can be found in [3].

4.2 Parallel Algorithm

First we test the parallel algorithm on a shared memory multiprocessor, the SGI Power Challenge, using 10 MIPS R10000 processors. In this machine the communications are "emulated" on the shared memory, so we cannot fully interpret the results as those of a message-passing model.

Figure 3 shows that the speedups obtained with small matrices are even smaller than 1 for 10 processors. However, executing the algorithm with large matrices we can even achieve superspeedups. This behaviour of the algorithm is due to the large communications overhead involved. With matrices of large dimension, the effect of the communications is reduced with respect to the computational cost, thus allowing better speedups.

We have also studied the enormous influence in the results of the configuration of the mesh. The best results are always obtained with meshes of type 1xP, while poorer results are obtained with meshes of type Px1. This behaviour of the algorithm is due to its communications scheme. As shown in figure 2, the larger is the number of column processors in the mesh, the smaller is the cost of gathering and scattering the matrices in each iteration of the algorithm.

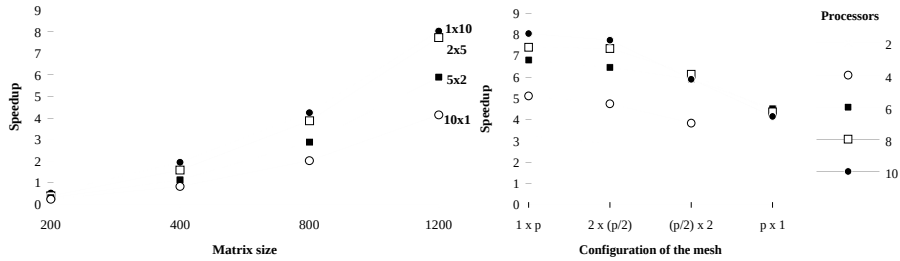

Fig. 3. Speedup vs. matrix size with p=10 and Speedup vs. configuration of the mesh with n=1200. Results obtained in the Power Challenge.

An experimental analysis has also been performed on a cluster of PCs connected by a Myrinet network and using two different versions of MPI environment. The speedups obtained are smaller than those in the Power Challenge. Notice that in this case we are using an architecture that has a distributed memory and a smaller ratio between computational cost and communications cost. However, in the case of large matrices we also obtain large speedups in this class of architecture.

Finally, in figure 4 we compare the performance obtained using two versions of the MPI environment implemented on the top of a Myrinet network. The GM version has been implemented by the vendor of the network and offers better performances for our problem that the BIP version, implemented in the Laboratory for High Performance Computing in Lyon.

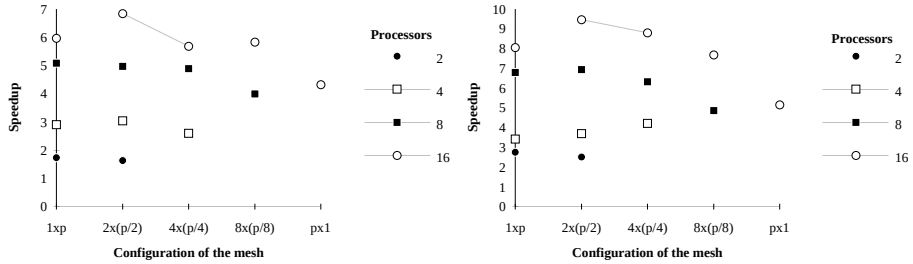

Fig. 4. Evolution of the speedups using different configurations of the mesh (n=1200). Results in a cluster of PCs and using two versions of the MPI environment (BIP and GM).

5 Conclusions

In this paper we show a scalable and portable parallel algorithm to solve the inverse eigenproblem of real symmetric Toeplitz matrices. We have obtained an efficient implementation of the algorithm that allows us to solve large problems with a enormous sequential cost. This parallel algorithm can be used to test the convergence of the method using different classes of starting generators and a larger test matrix set.

The results show the convergence of the algorithm in all cases except when the spectrum contains large clusters of eigenvalues. The experimental analysis performed shows scalable speedups, both in a shared memory multicomputer, a SGI Power Challenge, and in a cluster of Pentium II processors connected by a Myrinet network.

The parallelization of the method implies the redistribution of a lot of data in each iteration. The communications cost limits the performance in the case of small matrices. Moreover, we have seen that, given the communications scheme of the algorithm, the best results are obtained with horizontal meshes (1xP), while poorer results are obtained when we increment the number of rows of processors.

References

1. Blackford, L.S., et al.: ScaLAPACK Users' Guide., (1997), Philadelphia: SIAM Press.
2. Snir, M., et al.: MPI. The complete reference. Scientific and Engineering Computation, ed. J. Kowalik. (1996), Cambridge: The MIT Press.
3. Badia, J.M. and A.M. Vidal: Parallel Solution of the Inverse Eigenproblem for Real Symmetric Toeplitz Matrices. (1999). Technical Report DI01-04/99. Dpt. Informática, Univ. Jaume I: Castellon.
4. Cantoni, A. and F. Butler: Eigenvalues and eigenvectors of symmetric centrosymmetric matrices. Lin. Alg. Appl., (1976) (13): p. 275--288.
5. Laurie, D.P.: A Numerical Approach to the Inverse Toeplitz Eigenproblem. SIAM J. Sci. Sta. Comput., (1988). **9**(2): p. 401--405.
6. Trench, W.F.: Numerical Solution of the Inverse Eigenvalue Problem for Real Symmetric Toeplitz Matrices. SIAM J. Sci. Comput., (1997). **18**(6): p. 1722--1736.