Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 263–270. http://www.scpe.org
ISSN 1895-1767

ROBUST PARALLEL IMPLEMENTATION OF A LANCZOS-BASED ALGORITHM FOR AN STRUCTURED ELECTROMAGNETIC EIGENVALUE PROBLEM

MIGUEL O. BERNABEU*, MÁRIAM TARONCHER†, VÍCTOR M. GARCÍA‡, AND ANA VIDAL§

Abstract. This paper describes a parallel implementation of a Lanczos-based method to solve generalised eigenvalue problems related to the modal computation of arbitrarily shaped waveguides. This efficient implementation is intended for execution mainly in moderate-low cost workstations (2 to 4 processors). The problem under study has several features: the involved matrices are sparse with a certain structure, and all the eigenvalues needed are contained in a given interval. The novel parallel algorithms proposed show excellent speed-up for small number of processors.

Key words. large eigenvalue problem, structured matrices, microwaves

1. Introduction and examples. This paper is focused on the parallelisation of a Lanczos-based method for the solution of the following generalised eigenvalue problem: Given a symmetric pencil $\mathbf{A}x = \lambda\mathbf{B}x$, find all the generalised eigenvalues (and the corresponding eigenvectors) comprised in a given interval. This interval contains a large number of eigenvalues.

An efficient sequential method was already proposed in [1]. However, when the number of desired eigenvalues is very large, the execution time is still too long. A first parallel algorithm was recently introduced in [2], using MPI and a distributed-memory approach. The results presented in that paper show that the method parallelises extremely well.

A code based in the proposed technique will be included in a CAD tool for design of passive waveguide components. However, this CAD tool will usually run in low cost workstations or, at most, small PC clusters. For these small systems, a different approach should be chosen.

Therefore, the main goal of this paper is to explore different parallel programming approaches for the implementation of the sequential technique described in [1], in low cost workstations and small clusters.

Three different approaches have been examined: First, we have designed an OpenMP version of the Lanczos algorithm to take advantage of two-processor machines. Next, we implemented a version for distributed memory machines using MPI (Message Passing Interface), to execute it on clusters of PCs. Finally, a mixed approach was proposed in order to achieve optimum performance on clusters of two-processors.

A number of modifications have been carried out lately in the algorithm, to improve the reliability of the code, these shall be described as well. The main corrections have been i) the inclusion of ARPACK [7] routines for the extraction of all the generalised eigenvalues in a small subinterval, ii) the correction of the algorithm for balancing workload, and iii) the improvement of the linear solver, formerly based in the LU-Schur complement and now based on the LDL_t decomposition.

The paper is organised as follows: first, we will briefly outline the sequential problem (described in [1]), including the algorithmic modifications. Then, the new parallelisation schemes will be completely described, taking into account the different proposed options: i. e. MPI, OpenMP, MPI+OpenMP and so on. Finally, some numerical results are shown, and then the conclusions of this work are given.

2. Problem Description and Sequential Algorithm.

2.1. The electromagnetic problem. In this study, the efficient and accurate modal computation of arbitrary waveguides is based on the Boundary Integral - Resonant Mode Expansion (BI-RME) method (see the detailed formulation in [1, 3]). This technique provides the modal cut-off frequencies of an arbitrary waveguide from the solution of two eigenvalue problems. The first one is a generalised eigenvalue problem that models the transversal electric (TE) family of modes of the arbitrary waveguide. The structure of the corresponding matrices $\mathbf{A}$ and $\mathbf{B}$, shown in Fig. 2.1, presents a very sparse nature that is conveniently exploited in this work.

*Dpt. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, mbernabeu@dsic.upv.es

† Dpt. de Comunicaciones, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, matacal@iteam.upv.es

‡Dpt. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, vmgarcia@dsic.upv.es

§Dpt. de Comunicaciones, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, avidal@dcom.upv.es

Both matrices have a non-zero main diagonal, and a small $N \times N$ block in the right, bottom corner. Furthermore, the B matrix has two thin nonzero stripes $\mathbf{R}$ (with dimensions $N \times M$) and $\mathbf{R_t}$ ($M \times N$), in the last N rows and the last N columns. The size of the matrices is $(M+N) \times (M+N)$, but since M is far larger than N the matrices are very sparse (see [1]). This situation is given when a large number of cut-off frequencies is demanded. The transversal magnetic (TM) family of modes can be also formulated as a generalised eigenvalue problem (see [1]) with matrices $\mathbf{A}$ and $\mathbf{B}$ very similar to those explained before for the TE modes.

Here we will consider only the TE case.

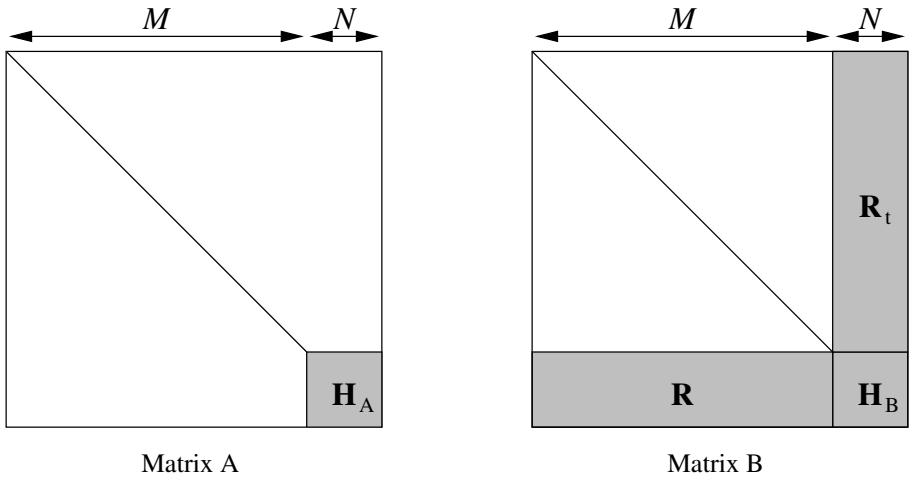

FIG. 2.1. *Structured matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *for the TE problem in a ridge waveguide.*

2.2. The sequential algorithm.

2.2.1. Shift-and-Invert Lanczos' algorithm. The standard techniques for generalised eigenvalue problems is the QZ algorithm. However, as was described in [1], in this case is not efficient since it does not use the structure of the matrices.

The technique proposed in [1] by the authors is based on Lanczos algorithm [6]. This algorithm, in its most basic form, allows the computation of a reduced number of extremal eigenvalues (the largest or smallest in magnitude). However, given a real number (usually called *shift*) σ, Lanczos' algorithm can be applied to the matrix $\mathbf{W} = (\mathbf{A} - \sigma\mathbf{B})^{-1}\mathbf{B}$. Lanczos' algorithm applied to this matrix will deliver as result the eigenvalues of the original problem closer to the shift σ. (This is called the "Shift-and-Invert" version of the Lanczos' algorithm.) The application of the Lanczos' method to this problem requires the solution of several linear systems, with $\mathbf{A} - \sigma\mathbf{B}$ as coefficient matrix. However, the structure of the matrices $\mathbf{A}$ and $\mathbf{B}$ allows a very efficient solution of these systems, using the Schur complement method. This method, described in [1] for this problem, was based in the LU decomposition; one of the algorithmic improvements mentioned above has been to change the LU-based technique to a LDL_t based algorithm, described next.

2.2.2. LDL_t **decomposition.** Let us now find out how is the $\mathbf{LDL_t}$ decomposition of the matrices in our case. For a matrix $(\mathbf{A} - \sigma\mathbf{B})$ with $\mathbf{A}$ and $\mathbf{B}$ as above, we can write

$$\begin{aligned}\mathbf{A} - \sigma\mathbf{B} &= \begin{pmatrix} \mathbf{U}_\sigma & \mathbf{R}_{\sigma t} \\ \mathbf{R}_\sigma & \mathbf{H}_\sigma \end{pmatrix} = \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{F} & \mathbf{T} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{D}_l & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_s \end{pmatrix} \cdot \begin{pmatrix} \mathbf{D_t} & \mathbf{F_t} \\ \mathbf{0} & \mathbf{T_t} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{D} \cdot \mathbf{D}_l \cdot \mathbf{D_t} & \mathbf{D} \cdot \mathbf{D}_l \cdot \mathbf{F_t} \\ \mathbf{F} \cdot \mathbf{D}_l \cdot \mathbf{D}_t & \mathbf{F} \cdot \mathbf{D}_l \cdot \mathbf{F_t} + \mathbf{T} \cdot \mathbf{D}_s \cdot \mathbf{T_t} \end{pmatrix} \end{aligned} \tag{2.1}$$

where the structure of matrix $\mathbf{A} - \sigma\mathbf{B}$ is identical to that of matrix $\mathbf{B}$ (Figure 2.1).

It is easy to check that we can take $\mathbf{D}$ as the identity matrix (since $\mathbf{U}_\sigma$ is diagonal), so that equating parts of this equation we arrive to the following procedure to compute the $\mathbf{LDL_t}$ decomposition:

1. Take $\mathbf{D}_l$ equal to $\mathbf{U}_\sigma$.
2. $\mathbf{F} = \mathbf{R}_\sigma \cdot \mathbf{D}_l^{-1}$ (trivial, since $\mathbf{D}_l$ is diagonal).
3. $\mathbf{T}$ and $\mathbf{D}_s$ are obtained computing the $\mathbf{LDL_t}$ decomposition of $\mathbf{H}_\sigma - \mathbf{F} \cdot \mathbf{D}_l \cdot \mathbf{F_t}$, through the LAPACK routine *dsytrf* .

2.2.3. Main interval decomposition. As we have mentioned before, the shift-and-invert version of the Lanczos' algorithm computes a subset of the spectrum centred in the shift point. The number of eigenvalues required will determine the number of iterations of the Lanczos' algorithm and its spatial cost [7]. Obviously, we cannot apply the Lanczos' algorithm to the main interval $[\alpha, \beta]$ where all the desired eigenvalues lie. The original problem should be split into many smaller ones to ensure the optimal performance of the Lanczos' algorithm.

As shown in [1], it is possible to use the Inertia Theorem to know in advance how many eigenvalues contain a given interval $[\alpha, \beta]$. For such interval, the $\mathbf{LDL_t}$ decompositions of $\mathbf{A} - \alpha\mathbf{B}$ (equal to $\mathbf{L}^\alpha\mathbf{D}^\alpha{\mathbf{L_t}}^\alpha$) and $\mathbf{A} - \beta\mathbf{B}$ (equal to $\mathbf{L}^\beta\mathbf{D}^\beta{\mathbf{L_t}}^\beta$) can be computed with a moderated cost (as described above). Then, the number of eigenvalues in the interval is simply the number $\nu(\mathbf{D}^\beta) - \nu(\mathbf{D}^\alpha)$, where $\nu(\mathbf{D})$ denotes the number of negative elements in the diagonal $\mathbf{D}$. It must be taken into account that the diagonal returned by *dsytrf* may not be completely "diagonal"; instead, it can be diagonal with $1 * 1$ and $2 * 2$ blocks, as a consequence of the special pivoting strategy. In this case, the eigenvalues of this special diagonal matrix can be easily found (solving the characteristic equation for the 2*2 blocks), which allows to compute the inertia quite efficiently anyway.

Thus, we can divide the original $[\alpha, \beta]$ interval into m subintervals $[\alpha_i, \beta_i]$ of different length, but containing nearly the same number of eigenvalues, and where the number of eigenvalues in each subinterval is known exactly. Therefore, the CPU time needed to compute the eigenvalues of every subinterval is expected to be nearly constant.

2.2.4. Sequential algorithm. The full sequential algorithm is as follows: The interval where lie the desired eigenvalues, $[\alpha, \beta]$, is divided in many small subintervals. Then, in each subinterval, a shift (possibly the middle point) is selected, and then the "Shift-and Invert" Lanczos algorithm is applied independently to each subinterval. This will compute all the eigenvalues in each subinterval, independently of the other subintervals. The number of subintervals and its width are chosen so that the number of eigenvalues on each subinterval is not too large.

This allows to obtain all the eigenvalues in the full interval in a reasonable time and without memory problems (see [1] for all the details).

ALGORITHM 1. **Sequential overall algorithm**.

INPUT: matrices $\mathbf{A}$ *and* $\mathbf{B}$*, the main subinterval* $[\alpha, \beta]$ *and the maximum number of eigenvalues per subinterval*
OUTPUT: eigenvalues of the pair $(\mathbf{A}, \mathbf{B})$ *contained in* $[\alpha, \beta]$ *ant its corresponding eigenvectors*

1. *Apply the Inertia Theorem to the full interval* $[\alpha, \beta]$ *to divide it into m smaller subintervals* $[\alpha_i, \beta_i]$

2. *for every subinterval* $[\alpha_i, \beta_i]$
4. $\sigma = (\beta_i - \alpha_i)/2$
5. *Apply Lanczos' shift-and-invert method to extract the eigenvalues closer to* σ *and its eigenvectors*
6. *end for*

7. *return eigenvalues and eigenvectors*

In the latest versions of the code, the robustness has been improved by using the ARPACK [7] routines for the symmetric generalised eigenvalue problem *dsaupd*. This routine is faster and safer than our previous versions of the Lanczos algorithm.

3. Parallel implementations.

3.1. Algorithmic approach. Clearly, the basic idea for the parallel implementation is to distribute the subintervals among the available processors; in each subinterval, the extraction of the eigenvalues will still be carried out in a sequential way.

Once we have computed the bounds of every $[\alpha_i, \beta_i]$ subinterval, m/p subintervals are assigned to each processor. This assignation is performed at the beginning of the algorithm, and there is no more communication among the processors until all of them have ended its work and results have been gathered.

As we have mentioned in Section 2.2.3, the CPU time needed to extract the eigenvalues of every subinterval is expected to be nearly constant. Thus, just distributing them among the available processors the work load balance is expected to be close to the optimal.

ALGORITHM 2. **Parallel overall algorithm**.

INPUT: matrices $\mathbf{A}$ *and* $\mathbf{B}$*, the main interval* $[\alpha, \beta]$ *and the maximum number of eigenvalues per subinterval*
OUTPUT: eigenvalues of the pair $(\mathbf{A}, \mathbf{B})$ *contained in* $[\alpha, \beta]$ *ant its corresponding eigenvectors*

Let us suppose that m *is multiple of* p

1. At the master processor
2. Apply the Inertia Theorem to the full interval $[\alpha, \beta]$ *to divide it into* m *smaller subintervals* $[\alpha_i, \beta_i]$
3. Assign m/p *subintervals to each processor*
4. End at the master processor

5. For each processor
6. for every assigned subinterval $[\alpha_i, \beta_i]$
7. $\sigma = (\beta_i - \alpha_i)/2$
8. Apply Lanczos' shift-and-invert method to extract the eig eigenvalues closer to σ *and its eigenvectors*
9. end for

10. Send eigenvalues and eigenvectors to the master processor
11. End for each processor

12. At the master processor
13. Gather the results from all the processors
14. End at the master processor

3.2. Implementation details. The new proposed algorithms have been implemented in Fortran 90, making use of the Intel Fortran Compiler for Linux. OpenMP and MPI standards have been used for the shared-memory version and distributed-memory version, respectively. In addition, BLAS and LAPACK [8] have been used whenever it was possible.

We have implemented three versions of Algorithm 2:

1. MPI version of algorithm 2
2. OpenMP version of algorithm 2
3. MPI+OpenMP version of algorithm 2

In the MPI version, all the processes read the input data from disk (matrices and main interval). Then, the main interval is divided with the technique described in the previous section. Next, a distributed algorithm is executed to assign the subintervals that should be solved by each process. Once it is done, every process solves its corresponding subintervals sequentially. Then, the results are gathered by the master process. This version is oriented to distributed memory machines, although it should work as well in shared memory machine.

In the OpenMP version, only the main thread reads the input data from disk. Then, the $[\alpha, \beta]$ interval is divided by the main thread, again. Next, the subintervals are assigned and distributed among the threads. This version is designed to run on shared memory machines.

Finally, the MPI+OpenMP version combines both techniques. In the first level of parallelism, a set of p MPI processes are spawned and they execute the MPI algorithm described before. Then, in the step where each MPI process solves its m/p subintervals, a second level of parallelism is introduced. Instead of sequentially solving those intervals, a group of p' OpenMP threads are created and the m/p intervals are divided among them in the same way described in the OpenMP version. There are $p * p'$ processors working on the solution of the problem. Note that this version is a combination of the two previous ones , and has been designed to run on a cluster of SMP machines.

4. Experimental results.

4.1. Description of the test environment. In order to test the performance of the three implemented versions of algorithm 2, we have chosen two different environments: an SMP Cluster and an SGI Computation Server.

The SMP Cluster consists of two Intel Xeon bi-processors running at 2.2 GHz with 4GB of RAM each one, interconnected through a Gigabit-Ethernet network.

The SGI Computation Server is an SGI Altix 3700. This machine is a cluster of 44 Itanium II tetraprocessors, although it has been designed as a ccNUMA machine [5] and therefore can be programmed as a SMP machine.

As mentioned previously, the algorithms have been designed to be included into a CAD tool of complex passive waveguide components. This kind of tools are expected to run in moderate-low cost workstations, so the SMP Cluster is the perfect testing environment. Despite of this, we have also chosen a more complex and powerful machine, the SGI Server, in order to test the algorithm performance using more expensive machines. Obviously, we will only use 4 of the 44 processors available for fair comparison purposes with the cheaper machine.

4.2. Experimental results. The following tables show the execution times of the implementations listed in section 3 for both test environments.

For the testbed, we have considered a single ridge waveguide described in [1].

TABLE 4.1
Execution time (s) for MPI implementation at the SMP Cluster.

$M+N$	$p=1$	$p=2$	$p=4$
5000	71.68	40.92	20.45
8000	199.26	121.22	67.98
11000	426.32	257.13	140.06
14000	772.10	413.06	221.21
17000	1247.71	655.40	367.26
20000	1685.27	1003.56	540.88

TABLE 4.2
Execution time (s) for OpenMP implementation at the SMP Cluster.

$M+N$	$p=1$	$p=2$
5000	71.68	38.11
8000	199.26	109.78
11000	426.32	246.32
14000	772.10	419.12
17000	1247.71	646.51
20000	1685.27	963.91

4.3. Analysis of the experimental results. The previous results show that the method described in section 2 parallelises extremely well in affordable machines. The key points for this good behaviour are the

TABLE 4.3
Execution time (s) for MPI+OpenMP implementation at the SMP Cluster.

$M+N$	$p=1$	$p=4$
5000	71.68	20.53
8000	199.26	61.59
11000	426.32	134.88
14000	772.10	216.84
17000	1247.71	333.86
20000	1685.27	534.69

TABLE 4.4
Execution time (s) for OpenMP implementation at the SGI Server.

$M+N$	$p=1$	$p=2$	$p=3$	$p=4$
5000	44.14	25.44	18.66	14.95
8000	161.99	86.46	69.25	55.67
11000	321.68	185.16	148.37	133.35
14000	598.13	337.35	249.26	247.38
17000	893.64	494.42	405.15	351.61
20000	1259.16	665.58	556.76	532.72

TABLE 4.5
Execution time (s) for MPI implementation at the SGI Server.

$M+N$	$p=1$	$p=2$	$p=3$	$p=4$
5000	44.14	23.69	16.17	13.09
8000	161.99	86.34	60.85	49.24
11000	321.68	172.88	117.61	91.53
14000	598.13	310.42	217.38	170.07
17000	893.64	498.16	304.24	241.64
20000	1259.16	658.08	446.46	349.44

application of Inertia Theorem to ensure a good work-load balance, as well as the absence of communications during the execution of the algorithm.

The different versions of the developed algorithms differ in the parallel programming standard used: MPI, OpenMP, or both of them. Both standards offer good performance and the final choice depends more on the machine architecture rather than on the sequential algorithm characteristics.

TABLE 4.6
Speed-up @ the SMP Cluster. Comparative study between OpenMP and MPI versions ($p=2$).

$M+N$	OpenMP	MPI
5000	1.88	1.75
8000	1.82	1.64
11000	1.73	1.66
14000	1.84	1.87
17000	1.93	1.90
20000	1.75	1.68

Table 4.6 shows the speed-up of MPI and OpenMP versions in a two-processor board (one of the nodes of the SMP Cluster). OpenMP results are slightly better than MPI ones. This result was expected because OpenMP can take more advantage of the shared memory architecture of the machine.

Table 4.7 shows the speed-up of MPI and MPI+OpenMP versions in a cluster of 2 two-processor boards. In this kind of environments with two levels of parallelism (shared memory at each node and distributed memory

TABLE 4.7
Speed-up @ the SMP Cluster. Comparative study between MPI+OpenMP and MPI versions ($p = 4$).

$M + N$	MPI+OpenMP	MPI
5000	3.49	3.51
8000	3.24	2.93
11000	3.16	3.04
14000	3.56	3.49
17000	3.74	3.40
20000	3.15	3.12

for the global view of the machine) the combination of MPI and OpenMP standards show better results than the use of MPI only. Again, OpenMP is taking advantage of shared memory features of the machine while MPI is not doing so.

TABLE 4.8
Comparative analysis between OpenMP and MPI versions @ SGI Server ($M + N = 20000$).

version	$p = 2$	$p = 3$	$p = 4$
MPI version	1,91	2,82	3,60
OpenMP version	1,89	2,26	2,36

Table 4.8 shows the speed-up of MPI and OpenMP versions at the SGI Server. In this machine, the MPI version scales better than OpenMP version. This rather surprising result is due to the scheduling policy. When the batch system runs the parallel algorithm, it can schedule the p threads/processes to different boards. With the MPI algorithm this does not create problems, since each process owns all the necessary data to perform its part of the algorithm. However, for the OpenMP implementation it is different, because all the threads need to access master thread's memory. This would create accesses to memory placed in a different board, which shall slow down the algorithm. Obviously, the problem worsens as the number of threads increases.

5. Conclusions. Three parallel implementations of a Lanczos-based method for solving a generalised eigenvalue problem have been successfully developed. The problem has got several distinct characteristics: matrices are sparse and structured, and the search of eigenvalues is reduced to a fixed interval.

The proposed technique parallelises very well and any of the implementations present very good speed-up even for a small number of processors.

OpenMP is the best choice for parallel programming of two-processors boards (and any SMP environment). For NUMA systems, it is concluded that OpenMP may present some problems and its use should be studied carefully.

Multi level programming (MPI + OpenMP) is the best choice for hybrid machines (those with two levels of parallelism), since this paradigm can take advantage of both shared and distributed memory features of the machine.

Finally, we can conclude that execution times in both machines are not too different, while speed-up is clearly better at the SMP Cluster. So, in this case, the performance-cost ratio is clearly better for the SMP Cluster.

Acknowledgement. Contract/grant sponsor: partially supported by Ministerio de Educación y Ciencia, Spanish Government, and FEDER funds, European Commission; contract/grant number: TIC2003-08238- C02-02, and by Programa de Incentivo a la Investigacion UPV-Valencia 2005 Project 005522

REFERENCES

[1] GARCÍA V. M., VIDAL A., BORIA V. E., VIDAL A. M.: *Efficient and accurate waveguide mode computation using BI-RME and Lanczos method,* Int. Journal for Numerical Methods in Engineering. DOI:10.1002/nme.1520 (2005).

[2] VIDAL A. M., VIDAL A., BORIA V. E., GARCÍA V. M.: *Parallel computation of arbitrarily shaped waveguide modes using BI-RME and Lanczos method,* Submitted to Int. Journal for Numerical Methods in Engineering. (2006).

[3] CONCIAURO G., BRESSAN M., ZUFFADA, C.: *Waveguide modes via an integral equation leading to a linear matrix eigenvalue problem,* IEEE Transactions on Microwave Theory and Techniques. (1984).

[4] Snir M., Otto S., Huss-Lederman S., Walker D. and Dongarra J.: MPI: The Complete Reference; MIT Press, (1996).
[5] Grbic A.: Assessment of Cache Coherence Protocols in Shared-memory Multiprocessors; Phd Thesis, University of Toronto (2003).
[6] A. Ruhe.: *Generalized Hermitian Eigenvalue Problem; Lanczos Method,* In Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide. SIAM, Philadelphia, first edition, (2000).
[7] R. B. Lehoucq, D. C. Sorensen, and C. Yang: *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, Philadelphia, (1998).
[8] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen: *LAPACK Users' Guide.* SIAM, Philadelphia, (1999).

Edited by: Dana Petcu
Received: May 31, 2007
Accepted: June 18, 2007