Parallel Eigensolvers for a Discretized Radiative Transfer Problem*

Paulo B. Vasconcelos[1], Osni Marques[2], and Jose E. Roman[3]

[1] Faculdade de Economia da Universidade do Porto,
Rua Dr. Roberto Frias s/n, 4200-464 Porto, Portugal
pjv@fep.up.pt
[2] Lawrence Berkeley National Laboratory,
1 Cyclotron Road, MS 50F-1650, Berkeley, CA 94720-8139, USA
oamarques@lbl.gov
[3] Instituto ITACA, Universidad Politécnica de Valencia,
Camino de Vera s/n, 46022 Valencia, Spain
jroman@dsic.upv.es

Abstract. In this work we consider the numerical computation of eigenpairs of a matrix derived from integral operators. The matrix is associated to a radiative transfer problem in stellar atmospheres that is formulated by means of a weakly singular Fredholm integral equation defined on a Banach space. We examine direct and iterative parallel strategies for the eigensolution phase, using state-of-the-art numerical methods implemented in publicly available software packages.

Keywords: High performance computing, eigenvalue computations, Fredholm integral equation, weakly singular kernel.
AMS subject classification: 68W10, 65F15, 45B05, 65R20, 65D20, 32A55.

1 Introduction

The solution of many problems in natural sciences and engineering would be unthinkable without the extensive use of approximations and numerical methods. There is a great wealth of such methods, based on either direct or iterative (projection-based) algorithms. Yet, without parallel processing many practical problems would not be solvable or would require an unacceptably large amount of time.

In the present contribution we focus on the numerical computation of eigenpairs of integral operators associated to a radiative transfer problem. This problem is formulated by a weakly singular Fredholm integral equation defined on a

* This work was partially supported by the Portuguese and Spanish governments via an Integrated Action (resp. ref. E-41/07 and ref. HP2006-0004), and partly by the Director, Office of Science, Advanced Scientific Computing Research Program, of the U.S. Department of Energy under contract No. DE-AC02-05CH11231.

Banach space. The numerical approach used to compute the (cluster of) eigenvalues and associated invariant subspace basis for the integral operator is based on its projection into a finite dimensional subspace. By evaluating the projected problem on a specific basis function, an algebraic eigenvalue problem is obtained; further, the corresponding coefficient matrix is banded.

This work examines the numerical computation of eigenpairs of the matrix of the integral operator using state-of-the-art numerical methods implemented in publicly available software packages. Numerical results using both direct and iterative parallel strategies are presented and discussed.

Section 2 provides a brief description of the problem and the necessary mathematical formalism, from the discretization method used to the projected approximate eigenvalue problem. Section 3 summarizes the computing platforms used for the performance analysis, along with a description of the installed software in each one of the platforms. In Sections 4 and 5, implementation details on the data generation as well as on the numerical methods used are addressed. The former section deals with numerical experiments with the ScaLAPACK library, therefore focusing on direct methods capable of delivering all (or a subset of) eigenpairs of the matrix problem. The latter section is driven by the SLEPc library: a number of state-of-the-art iterative methods are tested, providing insights in the more appropriate methods as well as in parameters specification. Both ScaLAPACK and SLEPc are available in the ACTS Collection of the US Department of Energy (DOE) [1]. The work finishes with conclusions and guidelines for the solution of similar problems in the context of high performance computing.

2 Problem Description

We seek to solve

$$T\varphi = \lambda\varphi \tag{1}$$

where $T : X \to X$ is an integral operator defined in $X = L^1([0, \tau^*])$, that satisfies

$$(Tx)(\tau) = \frac{\varpi}{2} \int_{\tau^*} E_1(|\tau - \tau^*|)\, x(\tau')\, d\tau', \quad 0 < \tau \leq \tau^* \tag{2}$$

where τ is the optical depth of the stellar atmosphere, τ^* is the optical thickness of the stellar atmosphere, $\varpi \in [0, 1]$ is the albedo, and $E_1 = \frac{\varpi}{2} \int_1^\infty \frac{exp(-\tau\mu)}{\mu} d\mu$ is the first exponential-integral function. We refer the reader to [2, 3] for details on the formulation of the problem. In a previous work two of the authors have investigated techniques for the solution of the associated radiative transfer equation [4]. Here we apply direct and iterative parallel strategies to investigate the spectral properties of the problem.

An integral equation of this type can be solved by a discretization mechanism, for instance by projecting it into a finite dimensional subspace. In particular, $T\varphi = \lambda\varphi$ can be approximated by

$$T_m\varphi_m = \lambda_m\varphi_m \tag{3}$$

in the finite dimensional subspace given by an m-dimensional subspace of X spanned by m linearly independent functions $(e_{m,j})_{j=1}^{m}$, X_m. Also, $0 = \tau_{m,0} < \tau_{m,1} < \ldots < \tau_{m,m} = \tau^*$. A nonuniform grid can be used, taking into account the boundary layer at 0. However, a symmetric matrix is obtained if a uniform grid is used instead. Moreover, a symmetrization operation can be applied to the non-symmetric operator matrix.

The projection approximation of T, $T_m : X \to X_m$, is defined by

$$T_m x = \pi_m T x = \sum_{j=1}^{m} \langle x, T^* e^*_{m,j} \rangle e_{m,j}. \tag{4}$$

being π_m the projection in the finite dimensional space. This leads to the eigenvalue problem

$$Ax = \theta x \tag{5}$$

of dimension m where A, large sparse and symmetric, corresponds to the restriction of T_m to X_m. The coefficients of A are given by:

$$\begin{cases} \varpi \left[1 + \frac{1}{d_{i,i-1}} (E_3\left(d_{i,i-1}\right) - \frac{1}{2})\right], & i > j \\ \frac{\varpi}{2d_{i,i-1}} \left[-E_3\left(d_{i,j}\right) + E_3\left(d_{i-1,j}\right) + E_3\left(d_{i,j-1}\right) - E_3\left(d_{i-1,j-1}\right)\right], & i \neq j \end{cases} \tag{6}$$

being $d_{i,j} = |\tau_{m,i} - \tau_{m,j}|$, $E_3(\tau) = \int_1^\infty \frac{\exp(-\tau\mu)}{\mu^3} d\mu$, and $E_3(0) = \frac{1}{2}$. For further details, we refer the reader to [2].

For computational purposes, the E_3 function is evaluated according to [5]. Noteworthily, the values of A decay significantly from the diagonal and for practical purposes the matrix can be considered banded.

Our goal is to approximate $T_m \varphi_m = \lambda_m \varphi_m$ by solving an associated symmetric eigenvalue problem $Ax = \theta x$ for large values of m.

3 Test Cases and Computing Platforms

The performance analysis was performed using test problems of different size, which were obtained by varying the resolution of the discretization mesh.

Our tests were performed on three platforms: bassi and jacquard, located at DOE's National Energy Research Center (NERSC), and odin, located at Universidad Politécnica de Valencia. Bassi is an IBM p575 POWER 5 system with 122 8-processor nodes and 32 GB of memory per node. Jacquard is an AMD Opteron cluster with 356 dual-processor nodes, 2.2 GHz processors, 6 GB of memory per node, interconnected with a high-speed InfiniBand network. Odin is a cluster of 55 nodes with dual Pentium Xeon processor at 2 GHz with 1 GB of memory per node and interconnected with a high-speed SCI network with 2-D torus topology. On bassi we used the basic linear algebra subroutines (BLAS) available in the ESSL library, while on jacquard we used the BLAS available in the ACML library. The software installation on odin includes SLEPc 2.3.3, PETSc 2.3.3, PRIMME 1.1, Hypre 2.0, and MUMPS 4.7.3.

For all cases showed in this paper we used $\varpi = 0.75$ (although not shown, tests with larger values of ϖ required similar computing times). We used a relative error $\varepsilon \leq 10^{-12}$ for the stopping criterion of the iterative method in order to obtain solutions as "accurate" as the direct method, and also to perform a more realistic comparison of the algorithms' computational performance.

The test cases that have been used for analyzing the performance of the solvers correspond to matrix dimensions (m) of 4K, 8K, 16K, 32K, 64K, 128K and 216K. The average number of non-zero elements per row is about 75. For example, the matrix of order 4K has 290,668 non-zero elements.

4 Solution Strategy with a Direct Eigensolver

4.1 Numerical Components

In this section we focus on ScaLAPACK's *pdsyevx* [6] for the solution of eigenvalue problem (5). This routine can compute all eigenvalues and (optionally) the corresponding eigenvectors, or only those eigenvalues specified by a range of values or a range of indices. The calculations consist of the following steps: reduction of the input (symmetric matrix) to tridiagonal form, computation of the eigenvalues of the tridiagonal using bisection, computation of the eigenvectors of the tridiagonal using inverse iteration, and (if eigenvectors are required) multiplication of the orthogonal transformation matrix from the reduction to tridiagonal form by the eigenvectors of the tridiagonal. Noticeably, *pdsyevx* may fail to produce orthogonal eigenvectors for tighly clustered eigenvalues. Also, it does not reorthogonalize eigenvectors that are on different processes (the extent of reorthogonalization is determined by the memory available).

ScaLAPACK assumes that the global data has been distributed to the processes with a one or two-dimensional block-cyclic data distribution. In the present work we have generated A using a 1-D block column distribution, i.e. each processor generates a block of columns of A. Since in our case A is banded, the 1-D column distribution is more natural and easier to implement. It would be desirable to take advantage of these properties but ScaLAPACK implements only LU and Cholesky factorizations for band matrices.

4.2 Performance Results

Figures 1 and 2 show the timings for the generation of A, and the eigensolution phase with *pdsyevx*, for matrices of dimension 4K, 8K, 16K and 32K, on up to 128 processors. As expected, the generation of A scales almost ideally when the number of processors is increased. Concerning the eigensolution phase, the timings on bassi refer to the computation of all eigenvalues but no eigenvectors, while on jacquard to the computation of the five largest eigenvalues and corresponding eigenvectors. In both cases, the scaling of *pdsyevx* showed to be satisfactory for the dimensions and number of processors that we have considered. Although the generation of A following a 2-D block cyclic distribution might lead to a

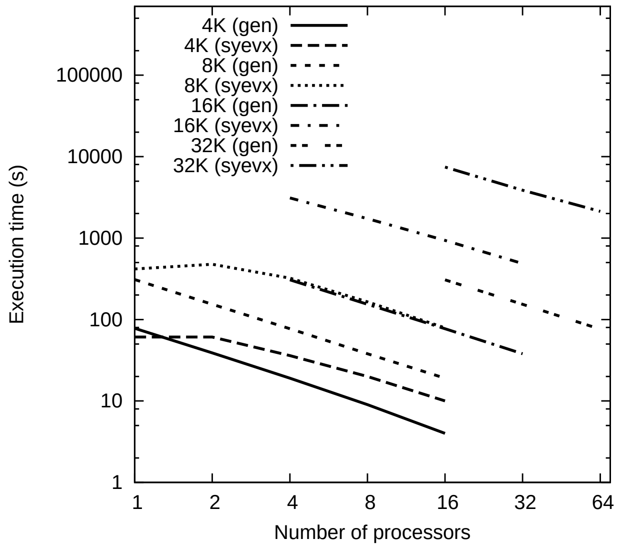

Fig. 1. Execution times for the matrix generation and eigensolution (*pdsyevx*) phases on bassi. The timings are for the computation of all eigenvalues but no eigenvectors.

better performance, we anticipate that for much larger matrices a direct method would be penalizing. This is also because the matrices become more banded (a property that currently we cannot take advantage of). It is well known that for most cases the reduction to tridiagonal form dominates the costs. However, for some pathological cases, where the eigenvalues are highly clustered, the costs for computing eigenvalues and orthogonalizing them may also be significant [7]. For our problems, the largest eigenvalues become very clustered as the dimension increases. This means that the corresponding eigenvectors would probably have to be orthogonalized more often, therefore increasing the computational costs.

5 Solution Strategy with an Iterative Eigensolver

An iterative eigensolver allows for the solution of larger problems, since one can better exploit the characteristics of the associated matrices (such as sparsity) and no direct transformation of the matrices is needed (such as reduction to tridiagonal form). Also, an iterative eigensolver is usually cheaper than a direct eigensolver when only a subset of eigenvalues and eigenvectors is required. The price to be paid is the convergence rate when there are tightly clustered eigenvalues, as well as the complexity of the different numerical algorithms that have to be used in order to get a scalable solution. Fortunately, there are soft-

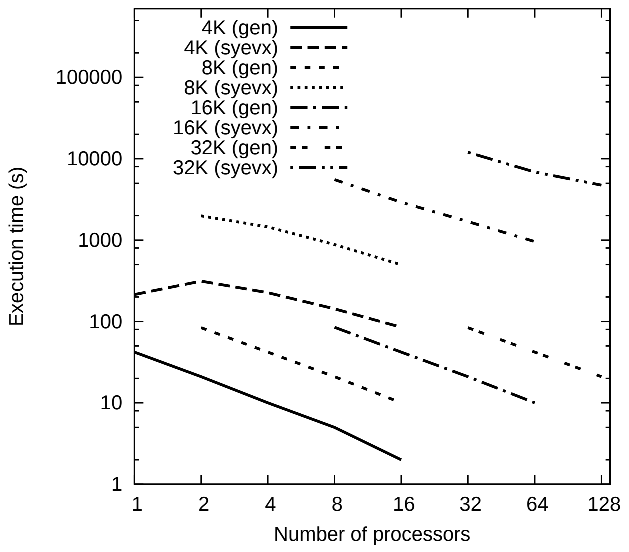

Fig. 2. Execution times for the matrix generation and eigensolution (*pdsyevx*) phases on jacquard. The timings are for the computation of the five largest eigenvalues and corresponding eigenvectors.

ware tools available that implement those algorithms, and their integration is relatively easy as explained below.

5.1 Numerical Components

PETSc[4], the Portable Extensible Toolkit for Scientific Computation [8], is a parallel framework for the numerical solution of problems arising in applications modeled by partial differential equations. Its design follows an object-oriented approach in order to be able to manage the complexity of numerical methods for very large and sparse problems on parallel computers. It is designed to provide enough flexibility to make software reuse feasible in many different contexts, but also with other goals in mind such as numerical robustness, computational efficiency, portability to different computing platforms, interoperability with other software, etc.

In PETSc all the code is built around a set of objects that encapsulate data structures and solution algorithms. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The data objects include management of index sets, vectors and sparse matrices

[4] PETSc is available at `http://www.mcs.anl.gov/petsc`.

in different formats, as well as basic support for structured and unstructured meshes. Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers.

For solving linear systems of equations, PETSc provides a variety of iterative methods such as Conjugate Gradient and GMRES, that can be combined with different preconditioners such as the incomplete LU factorization. Additionally, direct methods for linear systems are also available via complete factorizations that are handled in a similar way to preconditioners. In both cases, PETSc allows the use of external libraries that are seamlessly integrated in the framework, thus complementing the offered functionality. Examples of such libraries are MUMPS [9] for direct solvers and Hypre for preconditioners, the latter providing different methods such as the algebraic multigrid preconditioner, BoomerAMG [10].

SLEPc[5], the Scalable Library for Eigenvalue Problem Computations [11, 12], is a software library for the solution of large, sparse eigenvalue problems on parallel computers. It can be used for the solution of eigenproblems formulated in either standard or generalized form ($Ax = \lambda x$ or $Ax = \lambda Bx$), both Hermitian and non-Hermitian, with either real or complex arithmetic, as well as other related problems such as the singular value decomposition.

SLEPc is built on top of PETSc, and extends it with all the functionality necessary for the solution of eigenvalue problems. It provides uniform and efficient access to a growing number of eigensolvers. Most of these solvers belong to the class of Krylov projection methods, see [13]. In particular, SLEPc implements several variants of the Arnoldi and Lanczos methods, as well as the recently proposed Krylov-Schur method [14] that incorporates a very efficient restarting mechanism. In addition to these solvers, SLEPc seamlessly integrates third-party eigensolver software such as PRIMME [15]. The user is able to easily switch among different eigensolvers by simply specifying the method at run time. Apart from the solver, many other options can be specified such as the number of eigenvalues to compute, the requested tolerance, or the portion of the spectrum of interest.

SLEPc also provides built-in support for different spectral transformations such as the shift-and-invert technique. When such a transformation is applied, the matrix inverses such as $(A-\sigma B)^{-1}$ are not computed explicitly but handled implicitly via a linear system solver provided by PETSc.

For the application described in section 2, the algebraic problem is a standard eigenproblem, that can be symmetric or non-symmetric depending on whether the discretization grid is uniform or not. In the following, we will consider only the symmetric case.

As mentioned in section 2, the sparsity pattern of the matrix is quite special. In fact, it is not really sparse but banded, with a dense band. This will have some implications when treating the matrix from a sparse perspective. In particular, some operations may be quite inefficient with respect to the purely sparse case.

For moderate problem sizes, the Krylov-Schur method does a very good job in computing the largest eigenvalues, i.e. those closest to ϖ, and the correspond-

[5] SLEPc is available at `http://www.grycap.upv.es/slepc`.

Table 1. Performance of SLEPc and PRIMME on several test cases: m is the matrix size, k is the dimension of the subspace employed by the solver. For the Krylov-Schur and Jacobi-Davidson methods, the required iterations (its) and elapsed time (in seconds) is shown.

	Krylov-Schur			Jacobi-Davidson		
m	k	its	time	k	its	time
4K	48	230	27	48	68	14
8K	96	145	103	48	79	44
16K	192	119	789	48	89	181

ing eigenvectors. However, as the problem size grows, it is increasingly difficult for the method to have the solution converged. This difficulty is illustrated in Table 1, with the execution time and number of iterations required to compute 5 eigenpairs of the smallest test cases. In order for the Krylov-Schur method to get the solution in a reasonable number of iterations, it is necessary to increase the dimension of the subspace used by the solver. The table also shows results for the Jacobi-Davison method implemented in PRIMME, which is able to compute the solution faster and without having to increase the subspace dimension. Unfortunately, for larger test cases both methods fail to compute the solution in a reasonable time.

The problem of slow convergence is due to the fact that eigenvalues get more and more clustered around ϖ as the the order of the matrix grows, and it is well known that convergence of Krylov eigensolvers gets worse when the separation of eigenvalues is poor. Fortunately, the shift-and-invert spectral transformation is a workaround for this problem that fits very well in this application. The idea is to reformulate the eigenproblem as

$$(A - \varpi I)^{-1} x_i = \theta_i x_i . \tag{7}$$

This transformation does not alter the eigenvectors, x_i, and eigenvalues are modified in a simple way, $(\lambda_i - \varpi)^{-1} = \theta_i$, where λ_i are the eigenvalues of the original eigenproblem. If the Krylov-Schur eigensolver is applied to the transformed problem, the eigenvalues closest to ϖ will be retrieved first, as before, the difference being a more favourable separation of eigenvalues.

To illustrate the benefits of shift-and-invert, especially for large matrices, we show some performance data in Table 2. The first eigenvalue is very close to ϖ, and the next ones are very tightly clustered, with a separation of order 10^{-9} in the two largest test cases. With a basis size of 12 vectors, Krylov-Schur requires more than 20,000 restarts to attain the requested tolerance. As mentioned before, increasing the basis size would alleviate this bad convergence, but this is not viable for the largest test cases. In contrast, shift-and-invert performs extremely well, with only three restarts independently of the matrix size, and a total of just 23 linear solves. In the sequel, we will consider only runs of Krylov-Schur with shift-and-invert using a basis size of 12 vectors.

Table 2. Performance of SLEPc on several test cases: m is the dimension of the matrix, λ_1 is the largest eigenvalue, sep is the average distance among the 5 largest eigenvalues. For the Krylov-Schur method (K-S) with and without shift-and-invert (S-I), the table shows the required iterations (its) and elapsed time (with MUMPS in the latter case).

			K-S		K-S with S-I	
m	λ_1	sep	its	time	its	time
4K	0.749999813794	$1.12 \cdot 10^{-6}$	21,349	298	3	0.21
64K	0.749999999272	$4.37 \cdot 10^{-9}$	N/A	N/A	3	3.54
128K	0.749999999818	$1.09 \cdot 10^{-9}$	N/A	N/A	3	7.09

The spectral transformation enhances convergence dramatically, and it is provided by SLEPc in a straightforward manner. The downside is that the code has to deal with an inverted matrix, $(A - \varpi I)^{-1}$. Of course, this matrix is not computed explicitly. Instead, linear solves are performed whenever a matrix-vector product is required by the eigensolver.

Two alternatives exist for solving linear systems: direct and iterative methods. And both alternatives are provided in the context of PETSc, as mentioned before. For the shift-and-invert spectral transformation, it is more natural to use a direct method, providing full accuracy for the computed vector. However, that approach could lead to a non-scalable code or represent an exceedingly high cost. The iterative solver alternative can be a cheap and effective solution, provided that the requested tolerance is sufficient and that the convergence of the inner iteration is guaranteed by a good preconditioner.

Since only a subset of the eigenpairs are computed with SLEPc, the eigensolution stage will be relatively fast and the higher cost will reside in the computation of the matrix elements. Thus, the matrix generation stage has to be effectively parallelized in order to achieve good overall performance.

In PETSc, parallel matrices are distributed by blocks of contiguous rows. That is, a given processor owns a range of matrix rows and stores them in a sparse format. The internal format may vary depending on which solver is going to be used, but this is transparent to the code that sets the matrix elements. For instance, direct solvers such as MUMPS use a particular storage scheme.

In order to exploit the data distribution scheme during the parallel matrix generation, the different processors will compute only the matrix elements that belong to their locally owned rows. With this simple rule, there is no communication involved in the creation of the matrix, and parallel performance should be close to optimal provided that the matrix rows are equally distributed. However, it turns out that the number of non-zero elements is not uniform across the different rows, and this may lead to load imbalance. This imbalance is not severe, though, as will be illustrated in the performance results below. On the other hand, there is no obvious way of predicting how elements decay in a given row of the matrix, so the number of non-zero elements cannot be estimated prior to the actual computation of matrix elements.

In addition to the scheme discussed in the previous paragraph, our actual implementation incorporates the following enhancements:

1. Symmetry is exploited, that is, whenever a matrix element $a_{i,j}$ is computed, it is set also in the position corresponding to the symmetric element, $a_{j,i}$. This reduces the cost of the computation roughly by half. The drawback is that when the computation is done in parallel, the assembly of the matrix will imply communication among the processors.
2. The computation of each $E_3(d_{i,j})$ can be amortized because it is used in many different matrix elements. It is difficult to determine a priori which are those elements. Nevertheless, there is a locality effect that makes reuse more likely in close matrix elements. In order to exploit this fact, we implemented a simple caching mechanism that stores recently computed values of E_3. With a cache size of only 12 values, the percentage of cache hits in a typical run is 75%, so three quarters of the computation is avoided.

The above optimizations are a detriment to parallel efficiency, but we opted for prioritizing sequential performance and still retain good parallel behaviour.

5.2 Performance Results

For analyzing the parallel performance of the SLEPc-based code, the test problems have been solved on odin, requesting only 5 eigenvalues with a basis size of 12 vectors. In this section, only results are shown corresponding to the largest test case. Despite the large matrix size, the eigensolver does a very good job in getting the solution converged, because of the effectiveness of shift-and-invert in this case.

If a direct linear solver is used for the systems associated to the shift-and-invert transformation, then the computation with one processor is really fast. For instance, with MUMPS the overall eigencomputation takes 7 seconds. However, the situation is worse in parallel: 22.6 seconds with 2 processors, and this time cannot be reduced with more processors, see the horizontal line in Figure 3. Therefore, we discard the use of direct linear solvers in this setting, because they are not scalable, at least in this type of platforms. It should be stressed that the non-scalability comes from the problem properties, because MUMPS is not optimized for the case of banded, dense matrices.

As discussed before, the alternative is to use an iterative linear solver for the shift-and-invert transformation. In our application, GMRES combined with the algebraic multigrid preconditioner works very well. With one processor the computation is significantly slower (80 seconds), but scalability is remarkably good up to 32 processors and it catches up soon. Figures 3 and 4 show the execution time and the speed-up, respectively. Up to 32 processors, the execution time decreases with constant slope, and speed-up is close to the optimal one. The data for more processors are not shown in these figures, because the number of iterations of the linear system solver changes significantly (e.g. with 40 processors it makes a total of 115 linear iterations, whereas only 70 are necessary up to

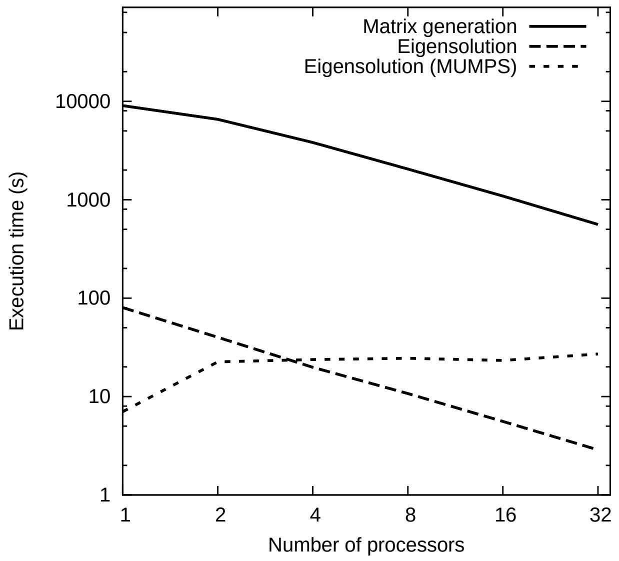

Fig. 3. Execution time in log-log scale for the matrix generation and eigensolution stages with SLEPc corresponding to the 128K test case on odin computer.

32 processors), and thus those times are not comparable. It seems that the preconditioner starts to lose effectiveness when the piece of the matrix assigned to each processor becomes too small.

Regarding the generation of the matrix, Figure 3 shows a time curve that starts with a significantly flatter slope. This is due to the fact that the enhancements discussed before make matrix generation sequentially very efficient. The consequence is a moderate speed-up, as illustrated in Figure 4. Overall, the efficiency of the computation is reasonably good.

In order to carry out a fair comparison of the performance for more than 32 processors, we run test cases with increasing problem size, so that the number of matrix rows assigned to each processor remains constant. Table 3 presents the obtained execution times, showing that the time for the eigencomputation phase is more or less constant. The total of accumulated linear solve iterations do not vary too much in this case. We can conclude that the computation of eigenvalues with SLEPc and BoomerAMG scales linearly with the problem size.

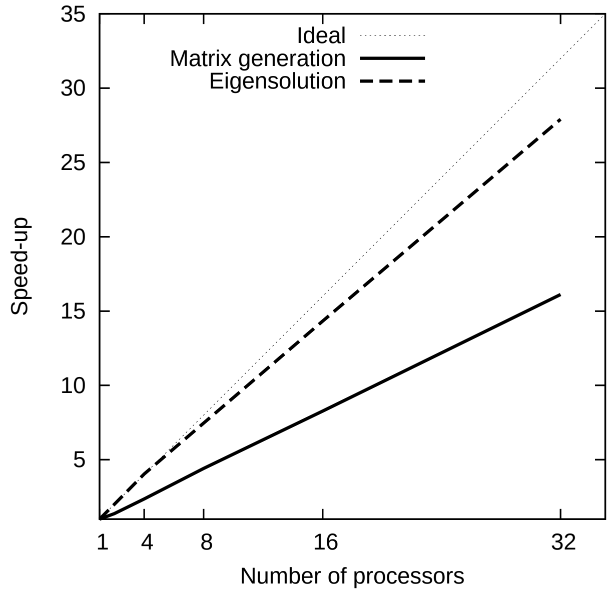

Fig. 4. Speed-up in log-log scale for the matrix generation and eigensolution stages with SLEPc corresponding to the 128K test case on odin computer.

6 Conclusions

In this contribution we considered the numerical computation of eigenelements of integral operators associated to a radiative transfer problem, using direct and iterative strategies available in publicly available software packages. The generation of the associated matrices in parallel leads to a significant reduction in computing time. Concerning the eigensolution phase, the algorithm implemented ScaLAPACK's *pdsyevx* showed good scalability for the number of processors used, for computing all eigenvalues only, or a subset of eigenvalues and corresponding eigenvectors. Similarly, SLEPc showed good scalability for the number of processors used for computing a subset of eigenvalues and corresponding eigenvectors. However, a closer look at Figures 1-2 and 3-4 reveals that a direct method becomes more costly as the problem size increases, greatly surpassing the (by itself costly) generation of the matrix. This results from the $O(n^3)$ complexity of the algorithm (in particular the reduction to tridiagonal form) and also because we cannot exploit the band structure of our problems. A strong point in favor of a direct method would be the proper determination of eigenvalue multiplicities. However, we have also been able to deal with such cases with iterative methods. Therefore, for our applications, iterative methods become the method of choice when a subset of eigenvalues is wanted.

Table 3. Scalability of SLEPc. For p processors, a problem of order $m = 4000 \cdot p$ is solved. T_{gen} and T_{solve} are the execution times (in seconds) for matrix generation and eigencomputation, respectively. *its* are the accumulated GMRES iterations.

p	m	T_{gen}	T_{solve}	*its*
1	4K	21	2.70	92
2	8K	40	2.85	92
4	16K	75	2.91	91
8	32K	145	2.91	89
16	64K	294	2.83	82
32	128K	630	2.94	73
54	216K	1012	3.37	86

References

1. Drummond, L.A., Marques, O.A.: An overview of the advanced computational software (ACTS) collection. ACM Transactions on Mathematical Software **31**(3) (2005) 282–301
2. Ahues, M., d'Almeida, F.D., Largillier, A., Titaud, O., Vasconcelos, P.: An L^1 refined projection approximate solution of the radiation transfer equation in stellar atmospheres. Journal of Computational and Applied Mathematics **140** (2002) 13–26
3. Rutily, B.: Multiple scattering theoretical and integral equations. In Cotanda, C., Ahues, M., Largillier, A., eds.: Integral Methods in Science and Engineering: Analytic and Numerical Techniques. Birkhauser (2004) 211–231
4. Marques, O.A., Vasconcelos, P.B.: Evaluation of linear solvers for astrophysics transfer problems. In: High Performance Computing for Computational Science - VECPAR 2006. Volume 4395 of Lecture Notes in Computer Science. (2007) 466–475
5. Abramowitz, M., Stegun, I.A., eds.: Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables. Volume 55 of Applied Mathematics Series. National Bureau of Standards, Washington, D.C. (1964) Reprinted by Dover, New York.
6. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1997)
7. I. S. Dhillon: A New O(N^2) Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem PhD. Thesis, University of California, Berkeley, 1997.
8. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.3, Argonne National Laboratory (2007)
9. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Computer Methods in Applied Mechanics and Engineering **184**(2–4) (2000) 501–520
10. Henson, V.E., Yang, U.M.: BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Applied Numerical Mathematics: Transactions of IMACS **41**(1) (2002) 155–177

11. Hernandez, V., Roman, J.E., Vidal, V.: SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Transactions on Mathematical Software **31**(3) (2005) 351–362
12. Hernandez, V., Roman, J.E., Tomas, A., Vidal, V.: SLEPc users manual. Technical Report DSIC-II/24/02 - Revision 2.3.3, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia (2007)
13. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H., eds.: Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA (2000)
14. Stewart, G.W.: A Krylov–Schur algorithm for large eigenproblems. SIAM Journal on Matrix Analysis and Applications **23**(3) (2001) 601–614
15. Stathopoulos, A.: Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue. SIAM Journal on Scientific Computing **29**(2) (2007) 481–514