A Parallel Implementation of the Trace Minimization Eigensolver

Eloy Romero* and Jose E. Roman

Instituto ITACA, Universidad Politécnica de Valencia,
Camino de Vera, s/n, 46022 Valencia, Spain.
Tel. +34-963877356, Fax +34-963877359
{eromero,jroman}@itaca.upv.es

Abstract. In this paper we describe a parallel implementation of the trace minimization method for symmetric generalized eigenvalue problems proposed by Sameh and Wisniewski. The implementation includes several techniques proposed in a later article of Sameh, such as multi-shifting, preconditioning and adaptive inner solver termination, which accelerate the method and make it much more robust. A Davidson-type variant of the algorithm has been also considered. The different methods are analyzed in terms of sequential and parallel efficiency.

Topics. Numerical algorithms for CS&E, parallel and distributed computing.

1 Introduction

Let A and B be large, sparse, symmetric (or Hermitian) matrices of order n. We are concerned with the partial solution of the generalized eigenvalue problem defined by these matrices, that is, the computation of a few pairs (λ, x) that satisfy

$$Ax = \lambda Bx, \tag{1}$$

where λ is a real scalar called the eigenvalue and x is an n-vector called the eigenvector. This problem arises in many scientific and engineering areas such as structural dynamics, electrical networks, quantum chemistry, and control theory. In this work, we focus on the particular case that the wanted part of the spectrum corresponds to the smallest eigenvalues. Also, we are mainly concerned with matrix pairs where B is positive (semi-)definite, although this condition can be relaxed under some circumstances.

Many different methods have been proposed for solving the above problem, including subspace iteration, Krylov projection methods such as Lanczos or Krylov-Schur, and Davidson-type methods such as Jacobi-Davidson. Details of these methods can be found in [1–4]. Subspace iteration and Krylov methods perform best when computing largest eigenvalues, but usually fail to compute the smallest or interior eigenvalues. In that case, it can be useful to

* Candidate to the Best Student Paper Award.

combine the method with a spectral transformation technique, that is, to solve $(A - \sigma B)^{-1}Bx = \theta x$ instead of Eq. 1. The problem with this approach is its high computational cost, since linear systems are to be solved at each iteration of the eigensolver, and they have to be solved very accurately. Preconditioned eigensolvers such as Jacobi-Davidson try to reduce the cost, by solving systems only approximately.

This work presents an implementation of the trace minimization eigensolver with several optimizations, including dynamic multishifting, approximate solution of the inner system and preconditioning. This method, described in section 2, can be seen as an improvement on the subspace iteration method that is able to compute smallest eigenvalues naturally. It can also be derived as a Davidson-type algorithm.

The implementation is being integrated as a solver in SLEPc, the Scalable Library for Eigenvalue Problem Computations. SLEPc is a software library for the solution of large, sparse eigenvalue problems on parallel computers, developed by the authors and other colleagues. An overview is provided in section 3, together with implementation details concerning the new solver, including the optimizations (preconditioning, multishifting and adaptive inner solver termination).

The paper is completed with section 4 showing the parallel performance of the implementations and the impact of the optimizations, and section 5 with some conclusions.

2 Trace Minimization Eigensolver

The trace minimization method for solving symmetric generalized eigenvalue problems was proposed by Sameh and Wisniewski [5], and developed further in [6]. The main idea of the method is to improve the update step of subspace iteration for generalized eigensystems as explained below.

Let Eq. 1 be the order n generalized eigenproblem to solve, and assume that X_k is a B-orthogonal basis of an approximate eigenspace associated to the smallest p eigenvalues, being $1 \leq p \ll n$. In subspace iteration, the sequence of computed approximations X_k is generated by the recurrence

$$X_{k+1} = A^{-1}BX_k, \quad k \geq 0 \ , \tag{2}$$

where the initial solution X_0 is an $n \times p$ full rank matrix. During the process, B-orthogonality of the columns of X_k is explicitly enforced. It can be shown that X_k eventually spans a subspace that contains the required eigenvectors [4].

This procedure requires the solution of p linear systems of equations with coefficient matrix A at each step k (one system per each column of X_k). Moreover, this has to be done very accurately (otherwise the global convergence is compromised), which is significantly expensive. Trace minimization tries to overcome that difficulty by exploiting a property stated in the following theorem.

Theorem 1 (Sameh and Wisniewski [5]). *Let A and B be $n \times n$ real symmetric matrices, with positive definite B, and let $\mathcal{X}$ be the set of all $n \times p$ matrices X for which $X^T B X = I_p$, $1 \leq p \leq n$. Then*

$$\min_{X \in \mathcal{X}} \operatorname{tr}(X^T A X) = \sum_{i=1}^{p} \lambda_i \ , \tag{3}$$

where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of $Ax = \lambda Bx$. The equality holds if and only if the columns of the matrix X, which achieves the minimum, span the eigenspace corresponding to the smallest p eigenvalues.

The trace of a square matrix, $\operatorname{tr}(\cdot)$, is defined as the sum of its diagonal elements, and it can be shown to be equal to the sum of its eigenvalues [4].

The trace minimization algorithm updates the current approximation X_k by subtracting a correction Δ_k that it obtained by solving the following constrained minimization problem,

$$\begin{aligned} &\text{minimize } \operatorname{tr}\left[(X_k - \Delta_k)^T A (X_k - \Delta_k)\right], \\ &\text{subject to } X_k^T B \Delta_k = 0 \ . \end{aligned} \tag{4}$$

As shown in [5], X_{k+1} satisfies

$$\operatorname{tr}(X_{k+1}^T A X_{k+1}) \leq \operatorname{tr}(X_k^T A X_k) \ , \tag{5}$$

if X_{k+1} is a B-orthonormal basis of the subspace spanned by $X_k - \Delta_k$.

By the method of Lagrange multipliers, the solution of the minimization problem, Eq. 4, can be computed by solving a set of saddle-point systems of linear equations,

$$\begin{pmatrix} A & BX_k \\ X_k^T B & 0 \end{pmatrix} \begin{pmatrix} \Delta_k \\ L_k \end{pmatrix} = \begin{pmatrix} AX_k \\ 0 \end{pmatrix} \ , \tag{6}$$

being $2L_k$ the Lagrange multipliers.

Nevertheless, we are interested only in Δ_k, so Eq. 6 can be reduce further, resulting in a set of constrained positive semi-definite systems,

$$(PAP)\Delta_k = PAX_k, \quad X_k^T B \Delta_k = 0 \ , \tag{7}$$

where P is the projector onto the orthogonal complement of BX_k,

$$P = I - BX_k (X_k^T B^2 X_k)^{-1} X_k^T B \ . \tag{8}$$

These systems can be solved by means of an iterative linear solver such as conjugate gradient (CG) or generalized minimal residual (GMRES) method, with a zero vector as initial solution so that the constraint is automatically satisfied.

It may seem that the projector of Eq. 8 will introduce an excessive overhead in the computation. However, in practice the overhead is not so high since p is normally small and the projector is applied implicitly, i.e., without building matrix P explicitly. Furthermore, PAP is better conditioned than A, thus reducing the required number of linear solver iterations.

Algorithm 1 summarizes the basic trace minimization method.

Algorithm 1 (Trace minimization)

Input: matrices A and B
number of desired eigenvalues p
dimension of subspace $s \geq p$
Output: resulting eigenpairs

Choose an $n \times s$ full rank matrix V_1 such that $V_1^T B V_1 = I_s$
For $k = 1, 2, \ldots$

1. Compute $W_k \leftarrow AV_k$ and $H_k \leftarrow V_k^T W_k$.
2. Compute all eigenpairs of H_k, (Θ_k, Y_k).
3. Compute the Ritz vectors, $X_k \leftarrow V_k Y_k$.
4. Compute the residual vectors, $R_k \leftarrow W_k Y_k - B X_k \Theta_k$.
5. Test for convergence.
6. Solve the inner system of Eq. 7 approximately.
7. $V_{k+1} \leftarrow B$-orthonormalization of $X_k - \Delta_k$.

End for

In step 2 of Algorithm 1, the eigenvalues have to be arranged in ascending order and the eigenvectors need to be orthonormalized. Note that the search subspace has dimension s, which can be for instance twice the number of wanted eigenpairs. At the end of the computation, the first p diagonal elements of Θ_k contain the computed Ritz values, and the first p columns of X_k contain the corresponding Ritz vectors.

It can be shown that the columns of X_k in Algorithm 1 converge to the eigenvectors with an asymptotic rate bounded by λ_i/λ_{s+1} [6, Th. 2.2]. Convergence can also be proved under the assumption that the inner systems in Eq. 7 are solved inexactly [6, §3.1].

Davidson-type version The main drawback of Algorithm 1, inherited from subspace iteration, is that the dimension of the subspace of approximants is constant throughout the computation. In [6], Sameh and Tong propose a variant with expanding subspaces, in which the number of columns of V_k grows, as well as the dimension of the projected matrix, H_k. This variant is related to the Generalized Davidson method, because it uses the preconditioned residuals for expanding the search subspace. These residuals will replace the right hand sides of the inner systems, Eq. 7. Integrating these two ideas with Algorithm 1 results in the Davidson-type trace minimization method, Algorithm 2.

Algorithm 2 (Davidson-type trace minimization)

Input: matrices A and B
number of desired eigenvalues p
block size $s \geq p$
maximum subspace dimension $m \geq s$
Output: resulting eigenpairs

Choose an $n \times s$ full rank matrix V_1 such that $V_1^T B V_1 = I_s$
For $k = 1, 2, \ldots$

1. Compute $W_k \leftarrow AV_k$ and $H_k \leftarrow V_k^T W_k$.
2. Compute s eigenpairs of H_k, (Θ_k, Y_k).
3. Compute the Ritz vectors $X_k \leftarrow V_k Y_k$.
4. Compute the residuals $R_k \leftarrow W_k Y_k - B X_k \Theta_k$.
5. Test for convergence.
6. Solve the systems $[P(A - \sigma_{k,i} B)P] d_{k,i} = P r_{k,i}$ s.t. $X_k^T B d_{k,i} = 0$.
7. If $dim(V_k) \leq m - s$
   then $V_{k+1} \leftarrow B$-orthonormalization of $[V_k, \Delta_k]$,
   else $V_{k+1} \leftarrow B$-orthonormalization of $[X_k, \Delta_k]$.

End for

In Algorithm 2, the number of columns of V_k is s initially, but grows as the iteration proceeds. The order of H_k grows accordingly, but only s eigenpairs are computed in step 2, and therefore the number of columns of X_k is constant. The other main difference with respect to Algorithm 1 is step 6, in which the system to be solved is different. Vectors $r_{k,i}$ and $d_{k,i}$ denote the ith column of R_k and Δ_k, respectively. The role of $\sigma_{k,i}$ and other considerations will be discussed in subsection 3.2. Finally, step 7 expands the working subspace, except if the maximum dimension has been reached, in which case V_{k+1} is set to have $2s$ columns only.

Apart from a much better behaviour in terms of convergence of Algorithm 2 with respect to Algorithm 1, there is another significant benefit, namely the reduction of the cost of enforcement of the constraint. The orthogonality requirement $X_k^T B d_{k,i} = 0$ is now an implicit deflation of the s Ritz vectors, and s can be much smaller than in the original algorithm. In fact, our implementation defaults to $s = p$ in Algorithm 2.

3 Implementation and Parallelization

The implementation has been developed in the context of the SLEPc library, being our intent to include the eigensolver in forthcoming releases.

3.1 Overview of SLEPc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [7][1], is a software library for the solution of large, sparse eigenvalue and singular value problems on parallel computers. It can be used for the solution of problems formulated in either standard or generalized form, both Hermitian and non-Hermitian, with either real or complex arithmetic.

[1] http://www.grycap.upv.es/slepc/

SLEPc provides a collection of eigensolvers including Krylov-Schur, Arnoldi, Lanczos, Subspace Iteration and Power/RQI. It also provides built-in support for different types of problems and spectral transformations such as the shift-and-invert technique.

SLEPc is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation, [8]), a parallel framework for the numerical solution of partial differential equations, whose approach is to encapsulate mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical message-passing codes. All the PETSc software is freely available and used around the world in many application areas.

PETSc is object-oriented in the sense that all the code is built around a set of data structures and algorithmic objects. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The three basic abstract data objects are index sets, vectors and matrices. Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers. Many different iterative linear solvers are provided, including CG and GMRES, together with various preconditioners such as Jacobi or Incomplete Cholesky. PETSc has also the provision to interface with third-party software such as HYPRE.

SLEPc extends PETSc with all the functionality necessary for the solution of eigenvalue problems. SLEPc inherits all the good properties of PETSc, including portability, scalability, efficiency and flexibility. SLEPc also leverages well-established eigensolver packages such as ARPACK, integrating them seamlessly. Some of the outstanding features of SLEPc are the following:

- Easy programming with PETSc's object-oriented style.
- Data-structure neutral implementation.
- Run-time flexibility, giving full control over the solution process.
- Portability to a wide range of parallel platforms.
- Usable from code written in C, C++ and Fortran.

3.2 Implementation Details

The current prototype implementation incorporates several improvements described in [6]. Most of them refer to the solution of the linear system in step 6 in both algorithms, which is the most expensive operation.

Shifting strategy An important acceleration in the original algorithm, that has been incorporated also in the Davidson-type version, comes from *shifts*. Instead of Eq. 7, this technique solves the systems

$$[P(A-\sigma_{k,i}B)P]d_{k,i} = PAx_{k,i}, \quad X_k^T B d_{k,i} = 0 \ , \tag{9}$$

where $d_{k,i}$ and $x_{k,i}$ are the ith columns of Δ_k and X_k, respectively, and $\sigma_{k,i}$ is the associated shift at step k.

If the desired eigenvalues are poorly separated from the remaining part of the spectrum, the unshifted method converges too slowly. Choosing an appropriate value of $\sigma_{k,i}$ can improve the separation of eigenvalues and accelerate the convergence. The shift heuristic strategy implemented in both algorithms is detailed in [6]. This technique of *multiple dynamic shifts* consists in computing a different shift $\sigma_{k,i}$ for each required eigenvalue in every outer iteration. The heuristic can be summarized as follows (i_0 is the number of converged eigenpairs):

– For $i_0 + 1$:
  if $\theta_{i_0+1} + \|r_{i_0+1}\|_{B^{-1}} \leq \theta_{i_0+2} - \|r_{i_0+2}\|_{B^{-1}}$ (test for cluster)
    then $\sigma_{i_0+1} \leftarrow \theta_{i_0+1}$,
    else $\sigma_{i_0+1} \leftarrow \max\{\theta_{i_0+1} - \|r_{i_0+1}\|_{B^{-1}}, \lambda_{i_0}\}$.
– For every $j > i_0 + 1$:
  if $\sigma_{j-1} = \theta_{j-1}$ and $\theta_j < \theta_{j+1} - \|r_{j+1}\|_{B^{-1}}$
    then $\sigma_j \leftarrow \theta_j$,
    else $\sigma_j \leftarrow \max\{\theta_l : \theta_l < \theta_j - \|r_j\|_{B^{-1}}\} \cup \theta_{i_0+1}$.

For practical reasons, employing 2-norms instead of B^{-1}-norms is recommended in [6]. However, this simplification does not result in a safe lower bound for some B matrices with $\lambda_{max}(B) > 1$, because a vector r satisfying $\theta_j - \|r\|_{B^{-1}} \leq \theta_j - \|r\|_2$ exists if $\lambda_{min}(B^{-1}) < 1$, due to the property

$$\|x\|_2\sqrt{\lambda_{min}(B^{-1})} \leq \|x\|_{B^{-1}} \leq \|x\|_2\sqrt{\lambda_{max}(B^{-1})} \ . \tag{10}$$

To overcome this difficulty, our implementation can make use of an approximation of the smallest eigenvalue of B (provided by the user or computed by the Gershgorin circle theorem) to obtain a better estimation of the shifts, because

$$\|r\|_{B^{-1}} = \sqrt{r^T B^{-1} r} = \|r\|_2\sqrt{z^T B^{-1} z} \leq \|r\|_2\sqrt{\lambda_{max}(B^{-1})} = \|r\|_2\lambda_{min}(B)^{-\frac{1}{2}}.$$

Linear solver and preconditioning When shifting strategies are used, the inner system of Eq. 9 may be indefinite or ill-conditioned. In order to make its resolution feasible with an iterative solver, it is necessary to use an effective preconditioner. In PETSc, it is very easy to precondition a linear system whose coefficient matrix is available explicitly. However, the preconditioning of Eq. 9 is not trivial.

Eq. 9 is a projected linear system with an orthogonality constraint, much like the Jacobi-Davidson correction equation. In [9], Sleijpen et al. describe how to solve

$$(I - uu^T)(A - \sigma B)(I - uu^T)t = r, \quad \text{s.t.} \quad t \perp u,$$

using a Krylov solver and an approximation K of $A - \sigma B$. Adapting this idea to the context of trace minimization results in solving Eq. 9 using a Krylov solver with a left preconditioner K for $A - \sigma_{k,i}B$, the operator $(PKP)P(A - \sigma_{k,i}B)P$ and right hand side vector $(PKP)(PA)x_{k,i}$. The transformation

$$F = \left[I - K^{-1}BX(X^T BK^{-1}BX)^{-1}X^T B\right] K^{-1}$$

is employed to optimize the application of the operator because

– if $z = Fy$ and $Py = y$, then $Pz = z$ and $PKPz = y$; and
– if the initial solution fed into the Krylov solver, v_0, belongs to the range of the operator, then all the subsequently generated vectors are in that subspace.

As a result, the matrix-vector product $z = (PKP)P(A-\sigma_{k,i}B)Pv$ in the Krylov solver can be calculated as $z = F(A - \sigma_{k,i}B)v$.

The F transformation is precomputed as

$$F = \left[I - K^{-1}BX(X^TBK^{-1}BX)^{-1}X^TB\right]K^{-1} \tag{11}$$

$$= K^{-1} - K^{-1}BX(X^TBK^{-1}BX)^{-1}X^TBK^{-1} \tag{12}$$

$$= K^{-1} - \tilde{J}\tilde{J}^T, \tag{13}$$

where $\tilde{J} = K^{-1}J$, $BX = JR$, and $J^TK^{-1}J = I$. In the implementation, in each external iteration a K^{-1}-orthonormal basis of BX is built and then pre-multiplied by K^{-1}. Consequently, the product by F only requires an application of $\tilde{J}\tilde{J}^T$ and K^{-1} on the vector, together with a subtraction.

Stopping criterion Another issue that has a significant impact on overall performance is the stopping criterion for the inner system solver. We adapt the strategy proposed in [5] to the shifted inner system, Eq. 9, as developed in [6]. Like preconditioning, monitoring the convergence of the linear system can avoid awkward breakdowns and unnecessary iterations.

The linear solver is configured to stop when either the error estimate is less than a certain tolerance, $\tau_{k,i}$, or the iterations of the method exceed a certain maximum. The tolerances are calculated as

$$\tau_{k,i} = \begin{cases} \sqrt{tol} & \text{if } k = 1 \\ (\theta_{k,i} - \sigma_{k,i})/(\theta_{k-1,s} - \sigma_{k,i}), & \text{if } k > 1 \text{ and } \theta_{k,i} \neq \sigma_{k,i} \\ (\theta_{k-1,i} - \sigma_{k,i})/(\theta_{k-1,s} - \sigma_{k,i}), & \text{if } k > 1 \text{ and } \theta_{k,i} = \sigma_{k,i}, \end{cases} \tag{14}$$

where *tol* is the tolerance demanded to the eigensolver. In some cases, $\tau_{k,i}$ is too close to 1, resulting in a poor update of the associated eigenvector that slows down its converge. To avoid this, our implementation allows the user to specify a maximum value of the tolerance.

The maximum number of allowed iterations is also set by the user, since the optimal value is dependent on the preconditioner and the conditioning of the inner system. However too small values may prevent convergence of the method.

Orthogonalization In terms of optimizations for parallel efficiency, the implementation makes use of an efficient Gram-Schmidt orthogonalization procedure available in SLEPc and explained in [10]. The default option is set up to Classical Gram-Schmidt with selective refinement.

4 Performance Analysis

This section summarizes the experiments carried out in order to evaluate the parallel performance of the implementations and the impact of the optimizations in the global convergence.

Table 1. Test cases from Matrix Market for testing optimizations (p: positive definite, s: positive semi-definite, i: indefinite). The column *speed-up* shows the gain factor of Algorithm 2 with respect to Algorithm 1.

		A		B		
	size	nonzeros	definiteness	nonzeros	definiteness	speed-up
BCSST02	66	2,211	p	66	p	0.81
BCSST08	1,074	7,017	p	1,074	p	2.80
BCSST09	1,083	9,760	p	1,083	p	2.21
BCSST10	1,086	11,578	p	11,589	s	1.31
BCSST11	1,473	17,857	p	1,473	p	0.90
BCSST12	1,473	17,857	p	10,566	s	1.54
BCSST13	2,003	42,943	p	11,973	s	1.47
BCSST19	817	3,835	i	817	p	–
BCSST21	3,600	15,100	i	3,600	p	–
BCSST22	138	417	i	138	p	–
BCSST23	3,134	24,156	i	3,134	p	–
BCSST24	3,562	81,736	i	3,562	p	–
BCSST25	15,439	133,840	i	15,439	p	–
BCSST26	1,922	16,129	i	1,922	p	–
BCSST27	1,224	28,675	p	28,675	i	1.26
BCSST38	8,032	355,460	p	10,485	i	–

Table 2. Test cases from UF Matrix Collection for analyzing parallel performance.

	size	A nonzeros	B nonzeros
DIAMON5	19,200	9,347,698	3,115,256
BS01	127,224	6,715,152	2,238,384
GYRO	17,361	1,021,159	340,431

The matrices used for the tests were taken from the Matrix Market and the University of Florida Sparse Matrix Collection. All test cases correspond to real symmetric generalized eigenproblems arising in real applications. The test cases used in the analysis of the various optimizations are listed in Table 1. For parallel performance tests, larger matrices were used, see Table 2.

The tests were executed on two clusters: *Odin*, made up of 55 bi-processor nodes with 2.80 GHz Pentium Xeon processors, arranged in a 2D torus topology with SCI interconnect, and *MareNostrum*, consisting of 2,560 JS21 blade computing nodes, each with 2 dual-core IBM 64-bit PowerPC 970MP processors running at 2.3 GHz, interconnected with a low latency Myrinet network.

Although in [6] CG is used as the solver for the linear system, in our experience only a few simple problems can be solved using that configuration. Unless otherwise stated, all tests reported in this section use GMRES with an algebraic multigrid preconditioner. This preconditioner is one of those provided by HYPRE [11]. With that configuration, all test cases in Tables 1 and 2 are solved without convergence problems.

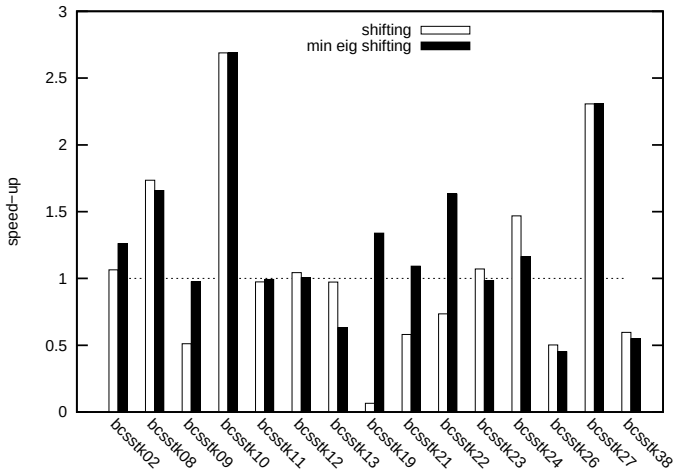

Fig. 1. Gain factor of dynamic shifting (`shifting`) and dynamic shifting with corrected norm (`min eig shifting`) with respect to the version without optimizations, in terms of the total number of inner iterations.

Original version vs Davidson-type version As it was pointed out previously, Algorithm 2 not only reduces the time spent in the inner iteration, but also the total number of inner iterations. In order to support that conclusion, both algorithms have been compared using the test cases listed in Table 1. The last column of this table shows the gain factor of Algorithm 2 with respect to Algorithm 1, except for those problems with one of the two matrices indefinite, which cannot be solved with for Algorithm 1. Furthermore, the frequency of use of the most expensive operations is very similar in both variants, so the parallel speed-up and scalability should be very similar (this can be confirmed in Figures 4 and 6). For these reasons, the optimization and parallelism analyses will focus on the Davidson-type version.

Analysis of optimizations The benefits of the different improvements described in section 3.2 have been analyzed using the test cases listed in Table 1. The results are shown in Figures 1, 2, and 3, as gain factors of the different strategies with respect to the version without optimizations, in terms of the total number of inner iterations. It can be observed from the figures that the optimizations are effective in most problems (but not all), making the computation up to 5 times faster in several cases, or even more.

The shifting strategy (Figure 1) accelerates the convergence in 66% of the problems. The version that employs a shifting strategy with the correction provided by the smallest eigenvalue of B (see section 3.2) is usually to be preferred, although it may be worse sometimes depending on the definiteness of matrix B. The tolerance strategy (Figure 2) presents better speed-up, specially setting its

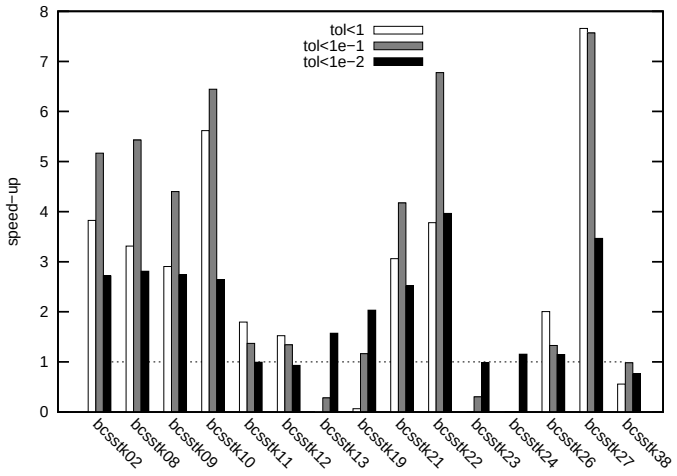

Fig. 2. Gain factor of the tolerance strategy with an upper bound of 1 (`tol<1`), 0.1 (`tol<1e-1`) and 0.01 (`tol<1e-2`), in terms of the total number of inner iterations.

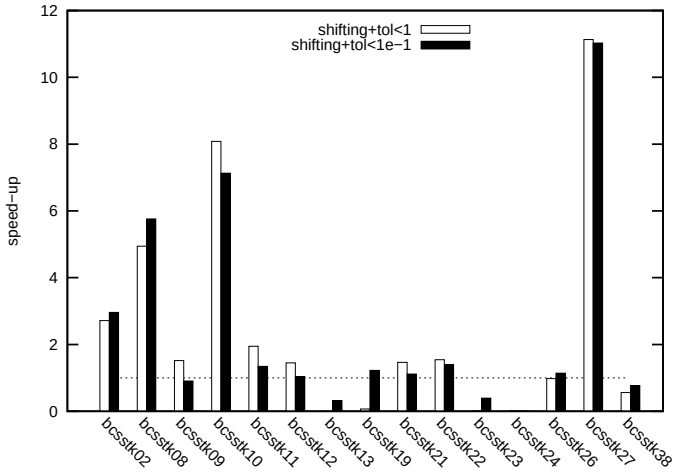

Fig. 3. Gain factor of dynamic shifting combined with tolerance strategy, in terms of the total number of inner iterations.

upper bound to 0.1. Finally, the combination of the two techniques is plotted in Figure 3.

Parallel analysis We start the parallel performance analysis with a scalability study, that is, to measure the parallel speed-up for different number of processors when the problem size is increased proportionally. Figure 4 presents the scala-

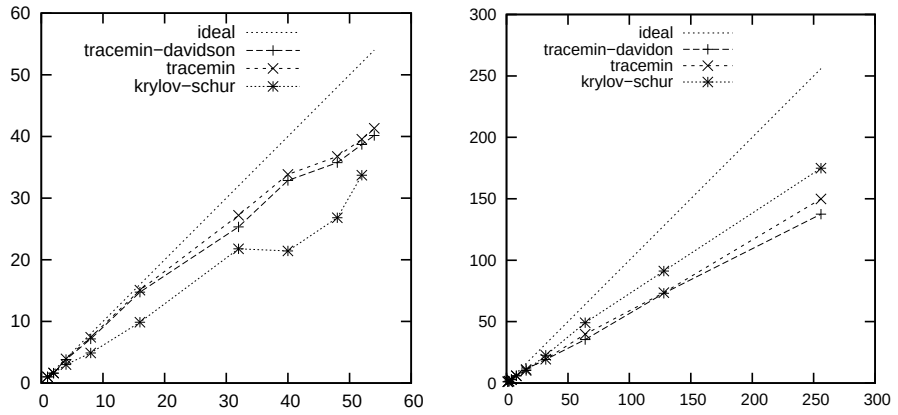

Fig. 4. Scalability of Davidson-type trace minimization and Krylov-Schur methods using a tridiagonal eigenproblem in odin (left) and MareNostrum (right).

bility results in both clusters for the Davidson-type trace minimization method, and also for the Krylov-Schur method (with shift-and-invert spectral transformation), using diagonally dominant random tridiagonal A and B matrices. In this case, the Jacobi preconditioner was employed when solving the inner systems.

The plots in Figure 4 indicate that both algorithms are reasonably scalable, considering the fact that they consist in a nested inner-outer iteration requiring a lot of communication among processors. In Odin, trace minimization seems to scale better than Krylov-Schur, but the roles are reversed in MareNostrum. This different behaviour could be explained by noticing that trace minimization spends more time in vector dot products (VecMDot) and norms (VecNorm) than in vector additions (VecMAXPY) and matrix-vector multiplications (MatMult), see Table 3. Vector products and norms scale worse, since they require a multi-reduction operation involving all processors, whereas VecMAXPY operations are trivially parallelizable and MatMult operations scale usually well. Maybe, the multi-reduction operation is less efficient in MareNostrum due to the hardware configuration.

We also presents results from some realistic problems, in particular those listed in Table 2. The speed-up of Algorithm 2 for all three problems is shown in Figure 5 for both platforms. For reference, the figure plots also the speed-up for

Table 3. Percentage of execution time corresponding to the three most time consuming operations, with 128 processors in MareNostrum.

	VecMAXPY	MatMult	VecMDot	VecNorm
Krylov-Schur	32%	19%	10%	10%
Trace Minimization	29%	26%	24%	8%

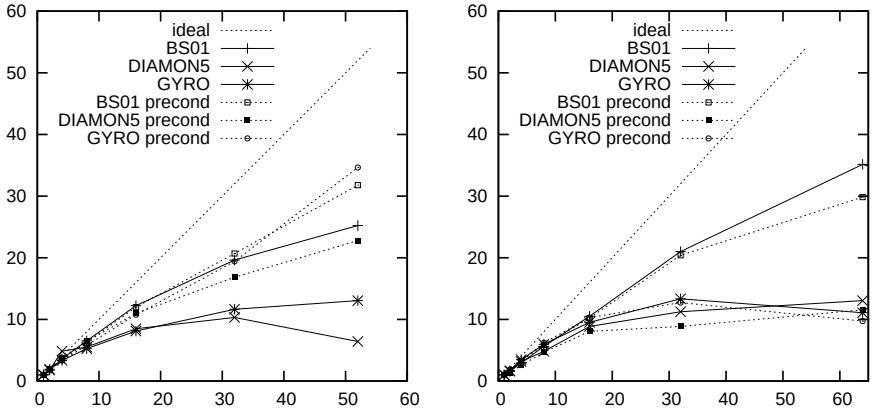

Fig. 5. Speed-up of Davidson-type trace minimization, as well as only the preconditioner application, for the BS01, DIAMON5, and GYRO problems in Odin (left) and MareNostrum (right).

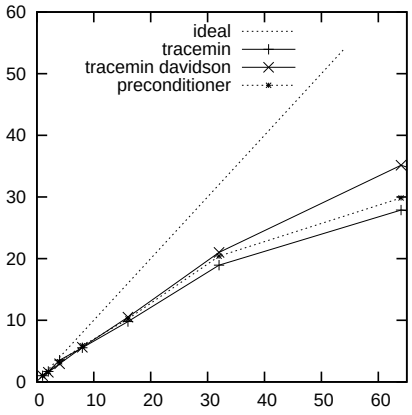

Fig. 6. Speed-up of the original trace minimization and Davidson-type methods, as well as only the preconditioner application, for the BS01 problem in MareNostrum.

the preconditioner application only, showing a strong correlation between the speed-up of the method and the preconditioner. We can conclude that a bad speed-up in the eigensolver can be attributed to a non-scalable preconditioner or a matrix with a disadvantageous sparsity pattern.

Finally, in Figure 6, we compare the speed-up of the Davidson-type implementation and the original trace minimization with the largest test case, showing a slight advantage of the former.

5 Conclusions

A parallel implementation of the trace minimization method for generalized symmetric eigenvalue problems has been developed and analyzed, focusing on its Davidson-type version. This implementation will be made available as a new solver in the SLEPc library. Our tests with several problems show that the proposed optimizations, such as multishifting, preconditioning and adaptive inner solver termination, accelerate the method considerably and make it more robust.

The parallel performance is comparable to that of Krylov-Schur, but the main advantage is that trace minimization can find the smallest eigenvalues in problems where Krylov-Schur cannot. This development is the prelude to the implementation in SLEPc of more advanced preconditioned eigensolvers such as Jacobi-Davidson.

Acknowledgements. The authors thankfully acknowledge the computer resources and assistance provided by the Barcelona Supercomputing Center (BSC).

References

1. Parlett, B.N.: The Symmetric Eigenvalue Problem. Prentice-Hall, Englewood Cliffs, NJ (1980) Reissued with revisions by SIAM, Philadelphia, 1998.
2. Saad, Y.: Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms. John Wiley and Sons, New York (1992)
3. 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)
4. Stewart, G.W.: Matrix Algorithms. Volume II: Eigensystems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2001)
5. Sameh, A.H., Wisniewski, J.A.: A trace minimization algorithm for the generalized eigenvalue problem. SIAM J. Numer. Anal. **19**(6) (1982) 1243–1259
6. Sameh, A., Tong, Z.: The trace minimization method for the symmetric generalized eigenvalue problem. J. Comput. Appl. Math. **123**(1-2) (2000) 155–175
7. Hernandez, V., Roman, J.E., Vidal, V.: SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans. Math. Soft. **31**(3) (2005) 351–362
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. Sleijpen, G.L.G., van der Vorst, H.A., Meijerink, E.: Efficient expansion of subspaces in the Jacobi–Davidson method for standard and generalized eigenproblems. Electron. Trans. Numer. Anal. **7** (1998) 75–89
10. Hernandez, V., Roman, J.E., Tomas, A.: Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. Parallel Comput. **33**(7–8) (2007) 521–540
11. 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
12. Sleijpen, G.L.G., der Vorst, H.A.V.: A Jacobi–Davidson iteration method for linear eigenvalue problems. SIAM Rev. **42**(2) (2000) 267–293