

A Parallel Implementation of the Davidson Method for Generalized Eigenproblems

Eloy ROMERO¹ and Jose E. ROMAN

Instituto ITACA, Universidad Politécica de Valencia, Spain

Abstract. We present a parallel implementation of the Davidson method for the numerical solution of large-scale, sparse, generalized eigenvalue problems. The implementation is done in the context of SLEPc, the Scalable Library for Eigenvalue Problem Computations. In this work, we focus on the Hermitian version of the method, with several optimizations. We compare the developed solver with other available implementations, as well as with Krylov-type eigensolvers already available in SLEPc, particularly in terms of parallel efficiency.

Keywords. Generalized eigenvalue problem, Davidson-type methods, Message-passing parallelization

1. Introduction

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

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

where the scalar λ is called the eigenvalue, and the n -vector x is called the eigenvector. If B is a positive definite matrix, the eigenvalues and the eigenvectors are real. Otherwise the eigenpairs could be complex even if the matrices are real. This problem arises in many scientific and engineering areas such as structural dynamics, electrical networks, quantum chemistry, and control theory.

Many different methods have been proposed for solving the above problem, including Subspace Iteration, Krylov projection methods such as Lanczos, Arnoldi or Krylov-Schur, and Davidson-type methods such as Generalized Davidson or Jacobi-Davidson. Details of these methods can be found in [1]. Subspace Iteration and Krylov methods achieve good performance when computing extreme eigenvalues, but usually fail to compute interior eigenvalues. In that case, the convergence is accelerated by combining the method with a spectral transformation technique, i.e., to solve $(A - \sigma B)^{-1} Bx = \theta x$ instead of Eq. 1. However this approach adds the high computational cost of solving

¹Corresponding Author: Instituto ITACA, Universidad Politécica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain; E-mail: eromero@itaca.upv.es.

large linear systems at each iteration of the eigensolver and very accurately (usually with direct methods). Davidson-type methods try to reduce the cost, by solving systems only approximately (usually with iterative methods) without compromising the robustness. This kind of methods are reaching increasing popularity due to their good numerical behaviour. Among the benefits of these methods, we can highlight (1) their good convergence rate in difficult problems, e.g. when internal eigenvalues are to be computed, (2) the possibility of using a preconditioner as a cheap alternative to spectral transformations, and (3) the possibility to start the iteration with an arbitrary subspace, so that good approximations from previous computations can be exploited.

Parallel Davidson-type eigensolvers are currently available in the PRIMME [11] and Anasazi [2] software packages. However, currently PRIMME can only cope with standard eigenproblems, and Anasazi only implements a basic block Davidson Method. Our aim is to provide robust and efficient parallel implementations of the Generalized Davidson Method in the context of SLEPc, the Scalable Library for Eigenvalue Problem Computations [8], that can address both standard and generalized problems. In this work, we focus on the case of symmetric-definite generalized eigenproblems, although some attention will be devoted to the case of semi-definite or indefinite B .

Section 2 describes the different variants of the method that we have considered. In sections 3 and 4 we provide details related to the implementation. Section 5 presents some performance results comparing the developed implementation with other solvers.

2. Generalized Davidson Method

The Generalized Davidson (GD) method is an iterative subspace projection method for eigenproblems. It is a generalization of the original Davidson's method [4] that allows using an arbitrary preconditioner. In its variant for symmetric-definite generalized eigenproblems, in each iteration the GD method performs a Rayleigh-Ritz procedure for selecting the most relevant approximate eigenpairs (θ, x) in the search subspace represented by a matrix with B -orthonormal columns, V . Then it computes a correction to x and expands the search subspace by adding this correction.

Many optimizations have been proposed with respect to the basic algorithm. The following are the most important ones considered in this work:

- The block version (originally called Liu-Davidson method [9,5]) updates several eigenpairs simultaneously. Besides accelerating the convergence, this technique is useful for finding more than one eigenpair. However, the block size s should be kept small to avoid a cost blow-up. In this section, X denotes a matrix with s columns, the approximate eigenvectors x_i . A similar notation is used for the corrections, D , and the residuals, R .
- Restarting combined with locking strategies allows the computation of more eigenpairs without enlarging the search subspace's maximum dimension.
- Olsen's variant [10] consists in computing the correction vector d_i as

$$d_i \leftarrow K^{-1}(I - x_i(x_i^* K^{-1} x_i)^{-1} x_i^* K^{-1})r_i, \quad (2)$$

being K^{-1} a preconditioner.

- Thick restart [12], in particular the $\text{GD}(m_{min}, m_{max})$ variant, restarts with the best m_{min} eigenvectors when the size of V reaches m_{max} .

Algorithm 1 (Block symmetric-definite generalized GD with B -orthogonalization)

Input: matrices A and B of size n , preconditioner K , number of wanted eigenpairs p ,
block size s , maximum size of V m_{max} , restart with m_{min} vectors

Output: resulting eigenpairs $(\tilde{\Theta}, \tilde{X})$

Choose an $n \times s$ full rank matrix V such that $V^*BV = I$

While $\text{size}(\tilde{\Theta}) < p$

1. Compute $H \leftarrow V^*AV$
2. Compute the eigenpairs (Θ, U) of H and sort them
3. Compute the first s Ritz vectors, $X \leftarrow VU_{1:s}$
4. Compute the residual vectors, $r_i \leftarrow AVu_i - BVu_i\theta_i$
5. Test for convergence
6. Compute the corrections, $d_i \leftarrow K^{-1}[I - x_i(x_i^*K^{-1}x_i)^{-1}x_i^*K^{-1}]r_i$
7. If $\text{size}(V) \geq m_{max}$, $V \leftarrow VU_{1:m_{min}}$
Else if k pairs are converged
Add eigenvalues $\theta_1, \dots, \theta_k$ to $\tilde{\Theta}$
 $\tilde{X} \leftarrow [\tilde{X} \quad VU_{1:k}]$
 $V \leftarrow VU_{k+1:k'}$, where $k' = \text{size}(V)$
Else, $V \leftarrow [V \quad B\text{-orthonormalize}([\tilde{X} \quad V], D)]$

End while

Algorithm 1 details the GD method for symmetric-definite eigenproblems, including the optimizations mentioned above. The subspace basis V grows in each iteration with s new vectors, except in the case of a restart or when some eigenpairs have converged in the last iteration. Note that most operations involving V , such as the computation of H , can be optimized to reuse values obtained in previous iterations.

From the numerical point of view, maintaining a B -orthonormal basis of subspace V is beneficial because exact eigenvectors satisfy a B -orthogonality relation. However, enforcing B -orthogonality may be too expensive, and if B is numerically singular the method may suffer from instability. As an alternative, we propose another variant that works with an orthonormal basis V , as shown in Algorithm 2. This algorithm is an adaptation of GD for non-Hermitian generalized problems [6], where we maintain the searching subspace basis V orthogonal with respect to the left invariant subspace corresponding to the converged vectors, spanned by $Y = B\tilde{X}$. With these changes, the method becomes valid for problems with B being indefinite and nonsingular, and $B^{-1}A$ being non-defective.

As a result, the new method does not work with eigenvectors, but with Schur vectors that are always orthonormal.

There are differences in the cost of both methods. On the one hand, Algorithm 2 has to compute two projected matrices H and G (step 1), whereas Algorithm 1 needs only one, and it also needs to orthogonalize the residual vector against the columns of Y (step 4). On the other hand, if the B -orthonormalization of step 7 is implemented with Iterative Gram-Schmidt, then Algorithm 1 may perform more than one B -inner product per vector.

Algorithm 2 (Block symmetric-indefinite generalized GD with orthonormalization)

Input: matrices A and B of size n , preconditioner K , number of wanted eigenpairs p , block size s , maximum size of V m_{max} , restart with m_{min} vectors

Output: resulting eigenvalues $\tilde{\Theta}$ and Schur vectors \tilde{X}

Choose an $n \times s$ full rank matrix V such that $V^*V = I$

While $\text{size}(\tilde{\Theta}) < p$

1. Compute $H \leftarrow V^*AV$ and $G \leftarrow V^*BV$
2. Compute the eigenvalues Θ and Schur vectors U associated to the pencil (H, G)
3. Compute the first s approximate Schur vectors, $X \leftarrow VU_{1:s}$
4. Compute the residual vectors, $r_i \leftarrow \text{orthonormalize}(Y, AVu_i - BVu_i\theta_i)$
5. Test for convergence
6. Compute the correction, $d_i \leftarrow K^{-1}[I - x_i(x_i^*K^{-1}x_i)^{-1}x_i^*K^{-1}]r_i$
7. If $\text{size}(V) \geq m_{max}$, $V \leftarrow VU_{1:m_{min}}$
Else if k pairs are converged
Add eigenvalues $\theta_1, \dots, \theta_k$ to $\tilde{\Theta}$
 $\tilde{X} \leftarrow [\tilde{X} \quad VU_{1:k}]$
 $Y \leftarrow [Y \quad BVU_{1:k}]$
 $V \leftarrow \text{orthonormalize}(Y, VU_{k+1:k'})$, where $k' = \text{size}(V)$
Else, $V \leftarrow [V \quad \text{orthonormalize}([Y \quad V], D)]$

End while

3. Implementation Details

In this section, we describe the context in which the algorithms have been implemented, and we compare it with existing related software efforts for solving large-scale eigenvalue problems with Davidson methods [7], in particular Anasazi and PRIMME.

Anasazi [2] is part of Trilinos, a parallel object-oriented software framework for large-scale multi-physics scientific applications. It was designed for being independent of the choice of the underlying linear algebra primitives, in order to facilitate its incorporation into larger libraries and application codes. The Anasazi framework contains a collection of independent eigensolvers that includes Block Davidson as described in Algorithm 1, among others.

PRIMME implements a parametrized Davidson method general enough to include algorithms ranging from Subspace Iteration to Jacobi-Davidson with several options about the correction equation. It uses its own distributed vectors with the sole support of BLAS, LAPACK and a sum reduction operation. Currently, it only supports standard eigenproblems, but the interface is prepared for a non-trivial matrix B .

The implementation of the algorithms that we are presenting here is being integrated as a solver in SLEPc, the Scalable Library for Eigenvalue Problem Computations. SLEPc's design goals lie between Anasazi and PRIMME. SLEPc eigensolvers use vectors, matrices and linear algebra primitives of PETSc (Portable, Extensible Toolkit for Scientific Computation, [3]), 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. However, the PETSc-dependence does not reduce the software interoperability because PETSc objects allow user implementation.

The stopping criteria and the computation of the correction vectors D (step 6) are encapsulated and decoupled from the eigensolvers, as in Anasazi, because they are the most application dependent components. The orthonormalization is also encapsulated in order to be possible to select any routine that SLEPc offers.

4. Parallel Implementation

The parallelization of the Davidson method can be done easily by parallelizing the operations with multivectors (such as V and X) and matrices (such as A and B), i.e., the matrix-vector products (such as $W \leftarrow AV$), the inner vector products (such as $H \leftarrow W^*V$) and the vector updates (such as $V \leftarrow VU$). The multivectors are dense rectangular matrices stored as a collection of vectors. One advantage of using a multivector data structure, instead of performing the operations column by column, is to improve the ratio of floating-point operations to memory references thus better exploiting the memory hierarchy.

Anasazi provides an interface to these operations, and has wrappers to employ the Epetra and Thyra (also Trilinos packages) multivectors and matrices. In PRIMME the user must provide the matrix-vector product, and the sum reduction operation (typically implemented by calling `MPI_Allreduce`) used by the hard-coded multivector routines.

PETSc supports MPI parallel sparse matrix computations by providing parallel matrix storage formats (distributed by blocks of contiguous rows), along with parallel vectors (distributed in a compatible way) and vector scattering operations for efficient communication among processes. If the matrix nonzero pattern is reasonably arranged (e.g. coming from an unstructured mesh discretization partitioned in compact subdomains), then the matrix-vector product operations scale well.

PETSc does not provide support for multivectors. In order to develop an optimized version of a block Generalized Davidson it is necessary to implement AXPY and dot product multivector operations, using BLAS to perform the local calculations.

In addition, some optimizations are applied in our implementations to improve the parallel performance. It is possible to mitigate the latency of communication primitives by joining several reductions together. For instance, the residual norm is commonly needed by the convergence test routine. Instead of performing the summation of the residual norms vector by vector, all the local results are considered in a unique call. This technique is also used in our eigensolver for updating the matrices H and G with a single communication operation. It is also possible to reduce the communication time by avoiding transferring unnecessary data. For instance, since V^*AV is symmetric only the upper triangular part is taken into account in the parallel operation.

5. Testing and validation

This section summarizes the experiments carried out in order to evaluate the convergence behaviour and the parallel performance of our implementations versus Anasazi and PRIMME. The preconditioner and the matrix-vector product may have a great im-

Table 1. Sequential results for computing on Odin the 10 largest magnitude eigenpairs over the symmetric-definite generalized eigenproblems testbed of 37 problems with Anasazi, SLEPc Davidson Algorithm 1, and SLEPc Krylov-Schur and ARPACK with a MUMPS direct solver and the GMRES solver. The iterations and accumulated time only include the problems solved by all eigensolvers.

Solver	Solved problems	Accumulated Iterations	Accumulated Time (s)	Accumulated Time/Iterations
SLEPc Davidson Algorithm 1	22	5923	13.54	0.0353
Anasazi	20	6251	17.40	0.0398
SLEPc Krylov-Schur MUMPS	37	–	5.23	–
SLEPc Krylov-Schur GMRES	35	–	5.81	–
ARPACK MUMPS	36	–	7.18	–

Table 2. Sequential results for computing on Odin the 10 largest magnitude eigenpairs over the symmetric-indefinite generalized eigenproblems testbed of 12 problems with SLEPc Davidson Algorithm 2 and SLEPc Krylov-Schur with a MUMPS direct solver. The iterations and accumulated time only include the problems solved by both eigensolvers.

Solver	Solved problems	Accumulated Iterations	Accumulated Time (s)	Accumulated Time/Iterations
SLEPc Davidson Algorithm 2	8	209	11.63	0.406
SLEPc Krylov-Schur MUMPS	12	–	55.90	–

pact on the global performance, especially on parallel performance. For that reason, the parallel test problems were selected to have an efficient parallel matrix-vector product. All tests use the Jacobi preconditioner (matrix K) because it is appropriate for $A - \sigma B$ operators with variable σ and its setup time is negligible.

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 CaesarAugusta, consisting of 256 JS20 blade computing nodes, each of them with two 64-bit PowerPC 970FX processors running at 2.2 GHz, interconnected with a low latency Myrinet network. In this machine, only 64 processors were employed due to account limitations.

The eigensolvers were configured for computing the 10 largest magnitude eigenvalues with a relative tolerance of 10^{-7} , with an m_{max} of 40, and up to 5000 iterations. The Davidson eigensolvers start with 5 vectors and restart with 8 eigenvectors. The default convergence test labels an eigenpair as converged if $\|Ax - Bx\theta\|_2/\theta \leq \text{tol}$, with $\|x\|_B = 1$ for Algorithm 1, and $\|x\|_2 = 1$ for Algorithm 2. The Krylov eigensolvers, which work on the operator $B^{-1}A$, were tested using a direct (MUMPS) and an iterative (GMRES) linear system solvers (both available in PETSc) with the default options.

The matrices used in the tests are from the University of Florida Sparse Matrix Collection. The symmetric-definite testbed used in Table 1 includes 37 eigenproblems with dimension up to 15,439. All matrices are real symmetric, and B positive definite. The tested Davidson eigensolvers solve around 21 problems of the testbed and in a comparable time (although Anasazi is slightly slower), with respect to Krylov eigensolvers. That may be due to the already commented limitation of the Jacobi preconditioner.

Table 2 summarizes the performance solving 12 indefinite B problems of size as the ones above. In that case, the Davidson eigensolver is faster than Krylov-Schur.

Figure 1 shows the speedup in both clusters solving the Boeing/pwtk matrix (standard problem), a 217,918 size matrix with 11,524,432 nonzero elements. The PRIMME performance is significantly better compared with SLEPc and Anasazi implementations.

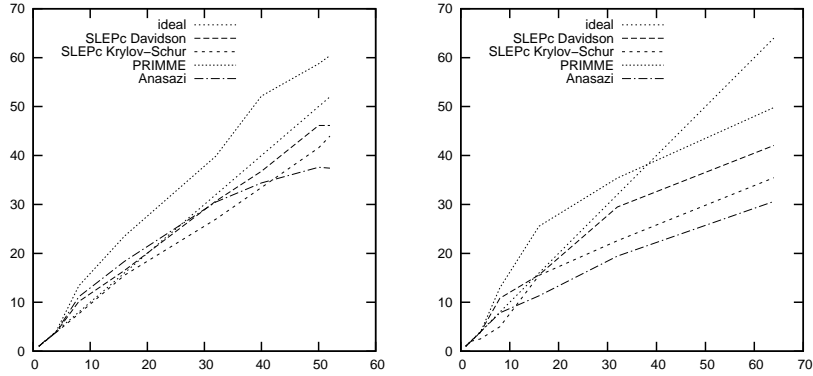


Figure 1. Speedup for the solution of the problem pwtk on Odin (left) and CaesarAugusta (right).

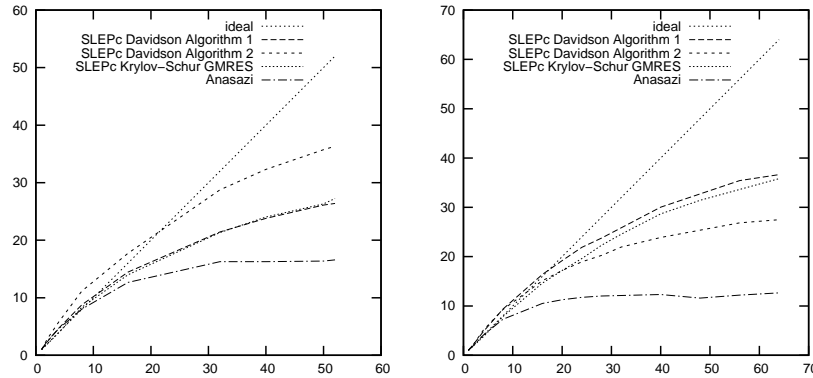


Figure 2. Speedup for the solution of the problem qa8f on Odin (left) and CaesarAugusta (right).

The generalized problem used in Figure 2 is formed by the Cunningham group matrices qa8fk and qa8fm. Its size is 66,127 and each matrix has 1,660,579 nonzero elements. SLEPc Algorithm 1 shows similar speedup as Krylov-Schur with GMRES, maybe dominated by the B -orthonormalization, and has the best speedup in CaesarAugusta. But on Odin, SLEPc Algorithm 2 has the best speedup. That may be due to the different performance of the B matrix-vector product and the multivector inner product (the core operation of orthonormalization) in both clusters, as well as the different use of these operations by the Algorithms 1 and 2 (see the end of section 2). In both platforms, the performance of Anasazi in generalized problems is not good.

6. Conclusions

In this work, we have presented a parallel implementation of the Generalized Davidson method for the solution of generalized eigenproblems. The methods have been implemented in SLEPc and constitute the basis for a future solver covering a number of Davidson-type methods, including Jacobi-Davidson. For the moment, only symmetric problems have been addressed, both definite and indefinite.

We have evaluated two algorithms, one with B -orthonormal basis and another one with orthonormal basis, the latter being able to solve indefinite problems. Both variants

are based on a block Davidson subspace expansion, with the Olsen method for computing the correction, and the thick restart technique with locking. In terms of convergence, our implementation of the B -orthogonal version is comparable to those available in Anasazi and PRIMME. The orthonormal version is unique to SLEPc since other packages only address the symmetric-definite case, and has proven to be faster than Krylov-Schur in our tests. From the perspective of parallel performance, our solver is less efficient than PRIMME, but among the available Davidson solvers for generalized eigenproblems it is the one that achieves better speedup.

Future works will include the development of different options for the computation of the correction, including the Jacobi-Davidson one, as well as the extension to standard and generalized non-Hermitian eigenvalue problems. It also remains to analyze the impact of different preconditioners on the overall performance.

Acknowledgements

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

References

- [1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.
- [2] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Trans. Math. Software*, 36(3):13:1–13:23, July 2009.
- [3] Satish Balay, Kris Buschelman, Victor Eijkhout, William Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, December 2008.
- [4] Ernest R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comput. Phys.*, 17(1):87–94, 1975.
- [5] Ernest R. Davidson. Monster matrices: Their eigenvalues and eigenvectors. *Comput. Phys.*, 7(5):519–522, 1993.
- [6] Diederik R. Fokkema, Gerard L. G. Sleijpen, and Henk A. van der Vorst. Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20(1):94–125, January 1999.
- [7] V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal. A survey of software for sparse eigenvalue problems. Technical report, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2007. Available at <http://www.grycap.upv.es/slepc>.
- [8] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, September 2005.
- [9] B. Liu. The simultaneous expansion method for the iterative solution of several of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. Technical Report LBL-8158, Lawrence Berkeley National Laboratory, 1978.
- [10] Jeppe Olsen, Poul Jørgensen, and Jack Simons. Passing the one-billion limit in full configuration-interaction (FCI) calculations. *Chem. Phys. Lett.*, 169(6):463–472, 1990.
- [11] Andreas Stathopoulos and James R. McCombs. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. Technical Report WM-CS-2006-08, College of William & Mary, November 2006.
- [12] Andreas Stathopoulos and Yousef Saad. Restarting techniques for the (Jacobi-)Davidson symmetric eigenvalue methods. *Electron. Trans. Numer. Anal.*, 7:163–181, 1998.