Romero Alcalde, E.; Román Moltó, JE. (2011). Computing subdominant unstable modes of turbulent plasma with a parallel Jacobi-Davidson eigensolver. Concurrency and Computation: Practice and Experience. 23:2179-2191. doi:10.1002/cpe.1740

Computing Subdominant Unstable Modes of Turbulent Plasma with a Parallel Jacobi-Davidson Eigensolver

Eloy Romero* Jose E. Roman†

September 19, 2002

Abstract

In the numerical solution of large-scale eigenvalue problems, Davidson-type methods are an increasingly popular alternative to Krylov eigensolvers. The main motivation is to avoid the expensive factorizations that are often needed by Krylov solvers when the problem is generalized or interior eigenvalues are desired. In Davidson-type methods, the factorization is replaced by iterative linear solvers that can be accelerated by a smart preconditioner. Jacobi-Davidson is one of the most effective variants. However, parallel implementations of this method are not widely available, particularly for non-symmetric problems. We present a parallel implementation that has been included in SLEPc, the Scalable Library for Eigenvalue Problem Computations, and test it in the context of a highly scalable plasma turbulence simulation code. We analyze its parallel efficiency and compare it with a Krylov-Schur eigensolver.[1]

Key words Message-passing parallelization; eigenvalue computations; Jacobi-Davidson; plasma physics simulation

INTRODUCTION

We are concerned with the standard eigenvalue problem defined by a large, sparse matrix A of order n, $Ax = \lambda x$, where the scalar λ is called the eigenvalue, and the n-vector x is called the eigenvector. Many iterative methods are available for the partial solution of the above problem, that is, for computing a subset of the eigenvalues. The most popular ones are 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 [2]. Krylov methods achieve good performance when computing extreme eigenvalues, but usually fail to compute interior eigenvalues. In that case, the convergence can be improved by combining the method with a spectral transformation technique, i.e., to solve $(A - \sigma I)^{-1}x = \theta x$ instead of $Ax = \lambda x$. The drawback of this approach is the added high computational cost of solving large linear systems at each iteration of the eigensolver. Moreover, for stability reasons these systems must be solved very accurately (normally with direct methods). Davidson-type methods aim at reducing the cost by solving linear systems approximately, without compromising the robustness, usually with iterative methods. This topic is treat by some authors [11].

*Universidad Politécnica de Valencia, Instituto I3M, Camino de Vera s/n, 46022 Valencia, Spain (elroal@upvnet.upv.es)

†Universidad Politécnica de Valencia, Instituto I3M, Camino de Vera s/n, 46022 Valencia, Spain (jroman@dsic.upv.es)

[1]Sponsored by Ministerio de Ciencia e Innovación, grant number TIN2009-075.

Davidson methods are becoming an excellent alternative due to the possibility of striking a balance between numerical behaviour and computational performance. A powerful preconditioner (close to the matrix inverse), if available, can usually reduce the number of iterations significantly. However, in practice its use is normally too expensive computationally and may be difficult to parallelize, thus dominating the cost of the eigensolver. Otherwise, depending on the performance of the matrix-vector product, the preconditioner and the orthogonalization, there exist Davidson-type variants that can be competitive with respect to Krylov-type eigensolvers. This paper illustrates an example of this.

Despite their potential benefit, it is still difficult to find freely available parallel implementations of Davidson-type eigensolvers, especially for the non-symmetric case, although there are some publications dealing with parallel implementations of these methods employed for certain applications, see for instance [10, 1, 7]. Parallel Davidson-type methods can be found in PRIMME [26] and Anasazi [3]. PRIMME implements many Davidson-type variants including Jacobi-Davidson, whereas Anasazi only implements a basic block Generalized Davidson method. However, none of them support non-Hermitian problems. Implementation of non-Hermitian eigensolvers gets complicated because of the need to work with invariant subspaces rather than eigenvectors, as well as to consider both right and left eigenspaces. Our aim is to provide a robust and efficient parallel implementation of the Jacobi-Davidson method in the context of SLEPc, the Scalable Library for Eigenvalue Problem Computations [9], that can address standard and generalized problems, both Hermitian and non-Hermitian, with either real or complex arithmetic. Our implementation is already included in version 3.1 of SLEPc, which contains also different Davidson variants other than Jacobi-Davidson. Regarding our Jacobi-Davidson solver, some preliminary results were presented in [18], where a simple non-restarted variant with real arithmetic is discussed. In this work, we focus on the restarted Jacobi-Davidson method for complex non-Hermitian problems. This paper builds upon our previous work [20] presented at Euro-Par 2010 in Ischia (Italy), and extends it with additional results.

The eigenvalue problem is the main algebraic problem in many areas such as structural dynamics, quantum chemistry and control theory. In this work, we show results for the eigenvalue calculation that takes place in the plasma physics application GENE, that solves a set of non-linear partial integro-differential equations in five-dimensional phase space by means of the method of lines. Because of the shape of the spectrum (see Fig. 1), computing the largest magnitude eigenvalues of the linearized operator is not particularly difficult, despite the unfavorable characteristics of the problem (complex non-Hermitian with matrix in implicit form). However, the case of computing the rightmost eigenvalues is much more difficult from the numerical point of view, since these eigenvalues are much smaller in magnitude compared to the dominant ones. This makes the computational problem challenging and suitable as a testbed for our new parallel eigensolver running on distributed memory architectures.

The rest of the paper is organized as follows. First we describe the Jacobi-Davidson method and several relevant variants such as harmonic extraction. Then the implementation details, including how the method is parallelized, are discussed. Next we provide a brief description of the application. The performance of the parallel eigensolver in this application is presented in the results section. Finally, we wrap up with some conclusions.

THE JACOBI-DAVIDSON METHOD

Davidson-type methods belong to the class of subspace projection methods, where approximate eigenvectors are taken from a search subspace $\mathcal{V}$. Each iteration of these methods

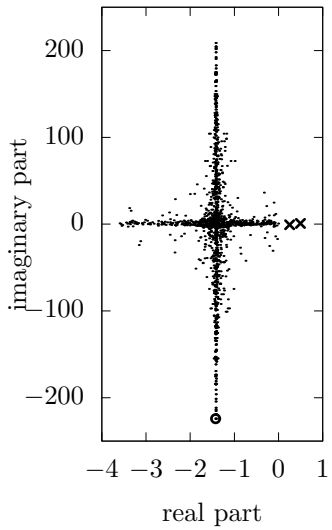

Figure 1: Spectrum of the linearized operator of a GENE problem similar to the test case I (Table 1). The largest magnitude (circle marks) and the rightmost (cross marks) eigenvalues are desired.

has two phases: subspace extraction and expansion. In the extraction phase, the solver selects the best (in terms of closeness to the desired region of the spectrum) from all available eigenpair approximations contained in $\mathcal{V}$. In the subspace expansion, a correction for the selected eigenpair is added to $\mathcal{V}$.

The subspace expansion distinguishes a Davidson-type variant from others. Jacobi-Davidson computes a correction t orthogonal to the selected approximate eigenvector u as an approximate solution of the so-called Jacobi orthogonal component correction (JOCC) [12] equation

$$A(u+t) = \lambda(u+t) \ , \quad u \perp t \ . \tag{1}$$

From (1) it is possible to formulate different linear systems, which are generically referred to as the correction equation. For our purpose, we implement the correction equation

$$\left(I - \frac{uz^*}{z^*u}\right)(A - \theta I)\left(I - \frac{uz^*}{z^*u}\right)t = -r \ , \tag{2}$$

where $r = Au - \theta u$ is the residual associated to the selected approximate eigenpair (θ, u), and $z \in \text{span}\{Au, u\}$. The above correction equation is more general than the one proposed in the original Jacobi-Davidson paper [24] and was introduced in [6].

If (2) is solved exactly, one step of the algorithm turns out to be one step of the Rayleigh Quotient Iteration, which converges almost quadratically [6]. Otherwise, if it is solved approximately, this high convergence rate may get lost. There is a trade-off between speed of convergence and the amount of work one is willing to spend for solving the equation, that is easily tuned if an iterative method is used. In practice, the performance of the eigensolver depends dramatically on a suitable stopping criterion for the iterative method.

In the subspace extraction phase, Davidson-type methods classically impose the Ritz-Galerkin condition to the eigenpair (θ, u) that will be selected,

$$r = Au - \theta u \perp \mathcal{V} \ . \tag{3}$$

Since $u \in \mathcal{V}$, it is possible to express $u = V\tilde{u}$, V being an orthogonal basis of $\mathcal{V}$. This leads to the low-dimensional projected eigenproblem $V^*AV\tilde{u} = \theta\tilde{u}$.

In practice this extraction technique, called Rayleigh-Ritz projection, obtains good convergence rates when the eigenvalues of interest are those located at the periphery of the spectrum. However, it gives poor approximate eigenvectors for interior eigenvalues. The harmonic Rayleigh-Ritz method was proposed in [14, 16] as an alternative extraction technique for this case.

Assuming that interior eigenvalues close to a given target τ are desired, harmonic Rayleigh-Ritz imposes the Petrov-Galerkin condition

$$(A - \tau I)u - \xi u \perp \mathcal{W} \tag{4}$$

to the selected eigenpair (θ, u) with $u = V\tilde{u}$, where the test subspace $\mathcal{W} \equiv (A-\tau I)\mathcal{V}$, and $\xi = \theta - \tau$. For numerical stability reasons, both V and W (a basis of $\mathcal{W}$) are constructed to be orthonormal. The relation between the two bases is given by $(A - \tau I)V = WS$, where S is upper triangular. Similarly to the previous case, this leads to the projected eigenproblem

$$S\tilde{u} = \xi W^*V\tilde{u} \ , \tag{5}$$

considering that $W^*(A - \tau I)V = S$. Then the smallest magnitude pairs $(\xi, \tilde{u})$ of (5) correspond to the pairs $(\xi + \tau, V\tilde{u})$ in $\mathcal{V}$ closest to the target τ.

Algorithm 1 *Block Jacobi-Davidson with harmonic Rayleigh-Ritz extraction*

Input: matrix A of size n, target τ, number of desired eigenpairs p,
block size s, maximum column size of V m_{max}, restart with m_{min} vectors
Output: resulting eigenpairs $(\widetilde{\Theta}, \widetilde{X})$

Choose an $n \times s$ full rank matrix V such that $V^*V = I$
While size$(\widetilde{\Theta}) < p$

1. Compute W, S such that $(A - \tau I)V = WS$, with $W^*W = I$ and S upper triangular
2. Compute $H \leftarrow W^*V$
3. Compute the eigenpairs (Ξ, U) of the matrix pencil (S, H) and sort them ascendantly
4. Compute the first s harmonic Ritz pairs, $X \leftarrow VU_{1:s}$ and $\theta_i \leftarrow \xi_i + \tau$
5. Compute the residual vectors R, $r_i \leftarrow AVu_i - \theta_i Vu_i$
6. Test for convergence
7. Compute the corrections T,
$$\left(I - \frac{x_i z_i^*}{z_i^* x_i}\right)(A - \theta_i I)\left(I - \frac{x_i z_i^*}{z_i^* x_i}\right) t_i = -r_i, \text{ with } z_i = (\bar{\tau}A + I)x_i$$
8. If size$(V) \geq m_{max}$, $V \leftarrow VU_{1:m_{min}}$
   Else if k pairs are converged
   Add eigenvalues $\theta_1, \ldots, \theta_k$ to $\widetilde{\Theta}$
   $\widetilde{X} \leftarrow [\widetilde{X} \quad VU_{1:k}]$
   $V \leftarrow VU_{k+1:k'}$, where $k' = \text{size}(V)$
   Else, $V \leftarrow [V \quad \text{orthonormalize}([\widetilde{X} \quad V], T)]$

End while

Algorithm 1 summarizes the scheme of a Jacobi-Davidson method with harmonic Rayleigh-Ritz extraction and the correction equation (2) with the particular choice of $z_i = (\bar{\tau}A + I)x_i$ as suggested in [6]. Note that the algorithm illustrates a block version, i.e., s eigenpair approximations are improved simultaneously in each iteration. As in all iterative algorithms based on expanding subspaces, due to memory limitations and in order to improve efficiency, the maximum size of the search and test subspaces have to be bounded. Thus, it is necessary to restart the computation whenever the available space for new basis vectors is exhausted. The thick restart technique [27] resets the subspace with the best m_{min} approximate eigenvectors when its size reaches m_{max}. A related issue is locking of already converged eigenvectors ($\tilde{X}$ in Algorithm 1), a deflation technique that amounts to extracting them from the active basis in order to avoid unnecessary computation for further improvement. When an eigenpair converges, it is removed from the subspace bases V and W, forcing a restart without it and in subsequent iterations orthogonalizing the new t_i vectors also against all locked vectors.

An important detail not shown in Algorithm 1 is that eigenvectors of the projected eigenproblem, U, are not orthogonal in general, so updates such as $VU_{1:k}$ would require reorthogonalizing the resulting basis. For improved stability, instead of directly computing the eigenpairs of the projected problem, a variant of the method called JDQZ [6] builds the (generalized) Schur decomposition and works with Schur vectors along the computation. At the end, the solver has obtained a partial Schur decomposition from which it is possible to compute the corresponding approximate eigenpairs. In the restart, JDQZ replaces the eigenvectors by an orthogonal basis of the corresponding eigenspace, which means updating V and W with the first m_{min} right and left Schur vectors of the projected problem, respectively.

For a more detailed description, the reader is referred to [24, 6, 23, 25].

IMPLEMENTATION DESCRIPTION

In this section, we describe the details of our particular implementation, with special attention to the parallelization and important aspects such as the solution of the correction equation.

Overview of SLEPc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [9], is a software library for the parallel solution of large-scale, sparse eigenvalue problems. It was designed to solve problems formulated in either standard or generalized form, both Hermitian and non-Hermitian, with either real or complex arithmetic. It can also be used for singular value and quadratic eigenvalue problems.

SLEPc provides a collection of eigensolvers on top of PETSc (Portable, Extensible Toolkit for Scientific Computation, [4]), including Krylov-Schur, Arnoldi, Lanczos, Subspace Iteration and Power/RQI. Davidson-type solvers were missing, and this motivated the development of our implementation, which was finally included in SLEPc 3.1 (released in August 2010).

PETSc is 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. PETSc is object-oriented in the sense that all the code is built around a set of data structures and algorithmic objects. The application pro-

grammer 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 GMRES, BiCGstab and BiCGstab(ℓ) [22], which can be combined with different preconditioners.

SLEPc eigensolvers rely on the parallel implementation of vector operations, the matrix-vector product and linear equation solvers. Basic implementations of these operations are supplied by PETSc objects. However, certain time-consuming, critical operations have custom implementation in SLEPc in order to improve the overall performance, as explained below.

Parallelization details

The problem matrix A and the vectors of size n, such as those stored in V, W, X, $\tilde{X}$ and R are distributed by blocks of rows in the corresponding matrices. The rest of vectors and matrices of size bounded by $m_{max} \ll n$, such as S, H, U, Ξ, $\widetilde{\Theta}$, are replicated in all nodes.

Operations involving distributed operands are parallelized. These include updating W, computing the coefficient matrices of the projected eigensystem (S, H), the selected Ritz vectors X and their residuals R, solving the correction equation (2) and orthogonalizing V and W.

The search subspace V is initialized with a basis of randomly generated vectors in parallel, taking care that each processor generates different random sequences.

The orthogonalization is based on a variant of classical Gram-Schmidt with selective reorthogonalization, providing both numerical robustness and good parallel efficiency [8]. The Schur decomposition of the projected problem and other minor computations are replicated in all nodes.

PETSc only provides basic support for multivectors (a multivector can be seen as a thin tall matrix, or a set of vectors that should be stored contiguously for memory efficiency). In order to develop an optimized version of Jacobi-Davidson it is necessary to implement basic multivector operations using BLAS to perform the local calculations. We provide an implementation in which individual vectors in a multivector can be used in common PETSc functions.

The most time-consuming operations are the multivector inner product W^*V and the update VU. However, PETSc only implements the level 2 BLAS operations W^*v_i and Vu_i. For W^*V, our implementation (i) performs the level 3 BLAS matrix-matrix product of the locally stored parts of V and W on each process, and then (ii) sums up all of them with a single call to an MPI reduction operation. For VU, it performs the BLAS matrix-matrix product of the locally stored part of V and the whole U.

Solution of the correction equation

The correction equation (2) is solved using PETSc's Krylov linear solvers, which need to compute matrix-vector products with the coefficient matrix. In this case, performing the shifting and the projections implicitly is more efficient than explicitly building the coefficient matrix. Also, applying only the left projector is sufficient to guarantee the condition $t \perp u$, provided that a Krylov solver is used with a zero starting vector, as shown in [25].

The trade-off between performance and global convergence is controlled in two ways. First, the maximum number of iterations and the relative residual tolerance can be tuned

for the linear system solver. Generally, increasing the number of linear solver iterations (inner iterations) causes a decrease of the global method iterations (outer iterations). In the next section we will compare the performance of two stopping criteria.

Secondly, the convergence behaviour of different Krylov solvers depends on the properties of the problem. We have tested two well-known solvers of this family: GMRES and BiCGstab(ℓ). They have different parallel behaviour because GMRES generally requires less matrix-vector products than BiCGstab(ℓ), but in contrast it has to explicitly maintain an orthonormalized basis whereas BiCGstab does not.

Finally, it is noteworthy that in the first outer steps the pairs resulting from the (harmonic) Rayleigh-Ritz procedure are usually poor approximations of the desired eigenpairs, and the target τ may be a relatively better approximation. Therefore, when the selected eigenpair's associated residual norm is greater than a threshold value *fix*, the correction equation (2) is solved with $\theta = \tau$ instead [6, Section 4.0.1].

In principle, the preconditioning of the correction equation is desirable. However, this issue is not addressed in this work because, as we will see below, the application matrix is defined in implicit form, thus preventing from computing conventional preconditioners. The issue of building an ad-hoc preconditioner is outside the scope of this paper.

GENE: A GYROKINETIC PLASMA SIMULATION CODE

One of the main goals of plasma simulation is to study the micro-instabilities that drive turbulence which in turn produces anomalous transport. This analysis must be done to determine the energy confinement time, a crucial parameter for the design of a fusion reactor.

GENE [5] is a massively parallel plasma simulation code written in Fortran 90/95, which is based on the numerical solution of the gyrokinetic equations. These equations stem from a simplification of the Maxwell-Boltzmann equations by eliminating the fast gyration of ions and electrons in strongly magnetized, dilute plasmas. This periodic motion is not relevant for most investigations, usually focusing on observables related to much slower time scales such as the net particle transport.

The linearized gyrokinetic equation can be written schematically as

$$\frac{\partial g}{\partial t} = \mathcal{L}[g], \tag{6}$$

where $\mathcal{L}$ is a time independent, complex, non-Hermitian integro-differential operator. It describes the time evolution of the modified distribution function of the gyrocentres g, which is a (scalar) function of the perpendicular spatial wave vector $(k_x,\ k_y)$, the coordinate z parallel to the magnetic field, the velocity parallel to the magnetic field $v_\parallel$, the magnetic moment μ, and the species label j. The GENE code follows an Eulerian approach. In particular, an explicit Runge-Kutta scheme is used for time integration, while the semi-discretization of phase space variables is done with a fixed grid and a combination of spectral and finite difference techniques. In GENE, the operator matrix is never computed explicitly, but implemented in a highly parallelized and efficient matrix-free form. For reasonably accurate models, the size of the problem ranges from several hundred thousand for linear simulations up to a few billion for nonlinear problems.

In GENE, some selected eigenvalues of the linearized operator need to be computed. In [17], the Krylov eigensolvers available in SLEPc are used for this. One scenario is the computation of the largest magnitude eigenvalue (circled in the spectrum of Fig. 1) in order to estimate the optimal timestep of the initial value solver. In this case, Krylov

Table 1: Test case I: GENE configuration for a very unstable kinetic ballooning mode with growth rate of 0.2055 and frequency of 0.2872 and another unstable mode ($0.1227 - 0.4494i$).

Dir.	Resol.	Boxsize		*Geom. & other params.*		Param.	Ions	Electrons
s	2			geom.	circular	R/L_n	2.0	2.0
x	12	l_x	125.628	$\hat{s}$	0.8	R/L_T	3.125	3.375
y	1	$k_{y,min}$	0.25	q_0	1.4	$mass$	1.0	0.00027
z	24			trpeps	0.18	$charge$	1.0	-1.0
v	48	l_v	3.0	β	0.001	T	1.0	1.5
μ	12	l_μ	9.0	(hyp_z, hyp_v)	(2, 0.5)	$dens$	1.0	1.0

Table 2: Test case II: GENE configuration similar to test case I but with a more realistic species configuration: deuterium, tritium, helium and electrons.

Dir.	Resol.	Boxsize		*Geom. & other params.*		Param.	^{2}H	^{3}H	He	e$^-$
s	4			geom.	circular	R/L_n	2.5	2.5	2.5	2.5
x	12	l_x	auto	$\hat{s}$	0.8	R/L_T	3.5	3.5	3.5	4.0
y	1	$k_{y,min}$	0.25	q_0	1.4	$mass$	2.014	3.016	4.002	5.4e-4
z	24			trpeps	0.18	$charge$	1	1	2	-1
v	48	l_v	3.0	β	0.001	T	1.0	1.0	1.0	1.5
μ	8	l_μ	9.0	(hyp_z, hyp_v)	(2, 0.5)	$dens$	0.45	0.45	0.05	1.0

solvers converge very fast. In a different context, SLEPc is also used for computing the subdominant unstable modes, i.e., the rightmost eigenvalues (crosses in Fig. 1). Due to the shape of the spectrum, these eigenvalues are much more difficult to compute. In [17], it is shown that the Krylov-Schur method with harmonic extraction has a reasonably good performance, compared to plain Krylov-Schur with spectral transformation. In the next section, we will show that our Jacobi-Davidson implementation performs even better. Computing these rightmost eigenpairs very fast is critical for some kind of analyses, e.g., when tracking the subdominant modes for varying values of several parameters [13], in which case a sequence of eigenproblems has to be solved.

COMPUTATIONAL RESULTS

This section summarizes the experiments carried out in order to evaluate the performance of our implementation, particularly in terms of scalability to a large number of processes.

Table 3: Test case III: GENE configuration corresponding to a stellarator device.

Dir.	Resol.	Boxsize		*Geom. & other params.*		Param.	Ions	Electrons
s	2			geom.	tracer	R/L_n	0.0	0.0
x	3	l_x	auto	file	hm128.dat	R/L_T	4.0	0.0
y	1	$k_{y,min}$	0.3	$\hat{s}$	-0.1088	$mass$	1.0	0.0025
z	128			q_0	1.11	$charge$	1.0	-1.0
v	48	l_v	3.0	β	0.001	T	1.0	1.0
μ	12	l_μ	9.0	(hyp_z, hyp_v)	(5, 0.5)	$dens$	1.0	1.0

The experiments are executed on Tirant, a machine consisting of 256 JS20 blade computing nodes, each of them with two 64-bit PowerPC 970+ processors running at 2.2 GHz, and interconnected with a low latency Myrinet network. Only 256 processors are used due to account limitations.

The following software is employed: GENE 1.4, PETSc 3.1, SLEPc 3.1 and LAPACK 3.2.1. All of them are built with the IBM compilers XL for C and Fortran, and linked with the BLAS routines in ESSL and MPICH 1.2.7.

Three different GENE parameter settings are used, detailed in the Tables 1, 2 and 3. These can be considered real use scenarios. In our previous work [18], only a lower resolution version of the test case 1 was employed. The GENE parameters shown in the tables determine the size and the spectrum of the associated eigenproblem (which influence the convergence of the solvers), and change the performance of matrix-vector products (which is an important part of the overall performance).

All experiments in this section are run using the default domain distribution (how many groups of processes there are in each direction) computed by GENE. For instance, when running test case II with 64 processes the default is to split the s, x, y, z, v, and μ directions in 4, 2, 8, 1, 1, 1, respectively. The impact of different domain distributions on parallel performance is studied in [17] and [19].

The Jacobi-Davidson solver will be compared with the fastest alternative found in [17], that is the Krylov-Schur method with harmonic extraction, which is available in SLEPc. Both solvers are configured for computing the two eigenvalues with largest real part (that correspond to the two instabilities with largest growth rates), with a tolerance of 10^{-5} for the residual norm relative to the magnitude of the eigenvalue. The search subspace is limited to 64 vectors, and when it is full, Krylov-Schur restarts with 32 vectors whereas Jacobi-Davidson keeps only 8. We have used block size 1, since larger values do not improve the performance in this case.

The harmonic procedure in both eigensolvers needs a target (called τ in Algorithm 1), that is, a point in the complex plane whose nearest eigenvalues are the desired ones. In the experiments the target is set to 1, which is a reasonable assumption for the upper bound for the real part of the eigenvalues for the problem at hand.

Stopping criterion for the correction equation solver

As already remarked, the way in which the correction equation is solved has a significant impact on the overall performance of the method. Our study in [20] compares the performance of Jacobi-Davidson when solving the correction equation with 110 iterations of GMRES and BiCGstab(2), showing that the latter obtained better results (this can be attributed to the high overhead of basis orthogonalization in GMRES). In this work, we extend the analysis by testing more flexible criteria with BiCGstab(2).

Figure 2 shows the time spent by Jacobi-Davidson using four processes when solving test case I with the BiCGstab(2) solver configured to perform 25, 50 and 80 (inner) iterations. We also consider modified versions of these configurations using a variable tolerance, that is, the iterative solution of the correction equation is stopped earlier if

$$\|r^{(j)}\|_2 \leq 2^{-i}\|r^{(0)}\|_2,$$

where $r^{(j)}$ is the residual of the correction equation at the linear solver iteration j, and i is the current outer iteration. This criterion comes from Newton methods and its use in the context of Jacobi-Davidson is suggested in [6], and is also used in [7].

From the figure, we observe that the best times are obtained limiting the number of iterations to 80, and in general the more iterations, the better time. In all cases, the

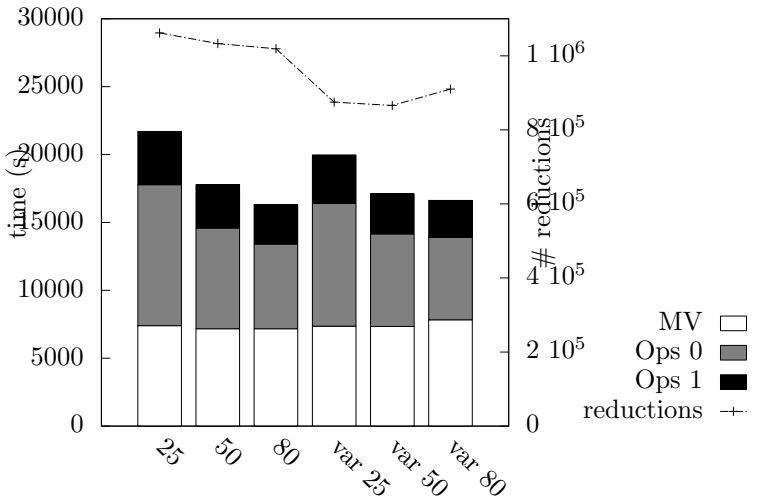

Figure 2: Time (in seconds, left axis) and number of reductions (right axis) spent by Jacobi-Davidson solving the test case I with four processes. The correction equation is solved with BiCGstab(2) and a maximum number of iterations of 25, 50 and 80. The cases labeled as "var" use a variable tolerance for the stopping criterion. The total time is split into matrix-vector products (MV) and vector operations without (Ops 0) and with (Ops 1) communication.

Table 4: Total number of matrix-vector products (#MV), total time spent by them (T. MV) in seconds, average time spent by one product (T./#MV), and number of reductions (#red.) performed by both methods when solving the test cases with four processes.

	Krylov-Schur				Jacobi-Davidson			
Case	#MV	T. MV (s)	T./#MV (s)	#red.	#MV	T. MV (s)	T./#MV (s)	#red.
I	119776	5226.2	0.0436	177000	297147	7185.15	0.0241	865900
II	67136	5299.4	0.0789	100700	169685	5754.05	0.0339	494000
III	45216	2439.6	0.0539	66960	110499	2527.2	0.0228	321100

variable tolerance criterion slightly reduces the time and, more importantly, significantly reduces the number of parallel reductions. Parallel reductions require the synchronization of all processes and constitute a great penalty for the scalability of the method. If we repeat the analysis for a larger number of processes, we will see that using more iterations may be counterproductive because overhead associated to collective communication becomes more important. Therefore, in subsequent experiments we use the variable tolerance criterion with a maximum number of inner iterations of 50.

Jacobi-Davidson versus Krylov-Schur

Figure 3 compares the results of Jacobi-Davidson and Krylov-Schur for test cases I, II and III. With four processes (left plot) the advantage of Jacobi-Davidson is clear. However, the difference between both methods is reduced significantly when the number of processes increases, as can be appreciated in the results with 256 processes (right plot) and in the speedups in Figure 4 commented below.

One of the possible causes is that Jacobi-Davidson needs more than twice as many

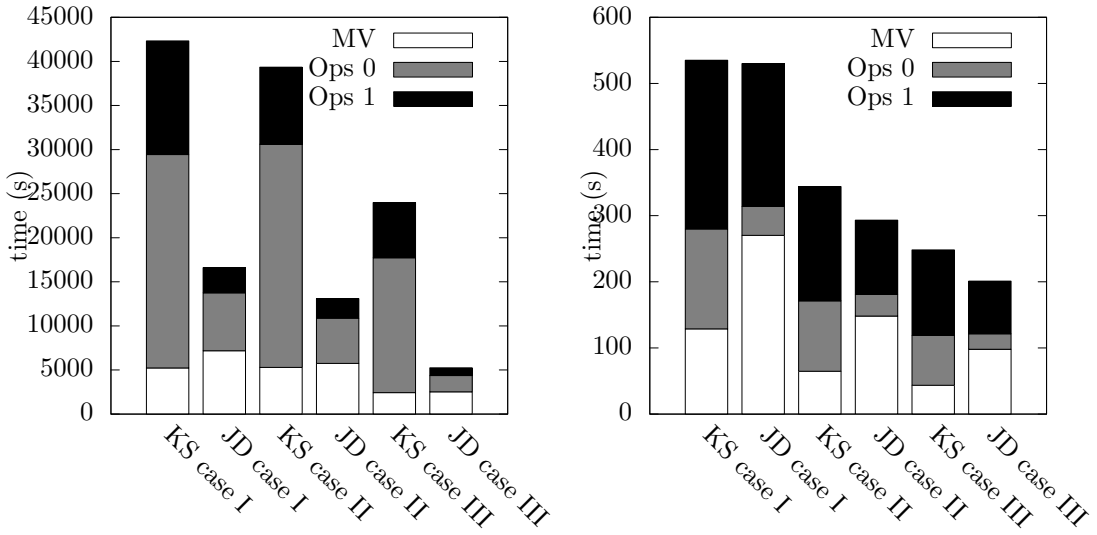

Figure 3: Time (in seconds) using Krylov-Schur (KS) and Jacobi-Davidson (JD) with BiCGstab(2) solving test cases I, II and III with 4 (left) and 256 (right) processes. The total time is split into matrix-vector products (MV) and vector operations without (Ops 0) and with (Ops 1) communication.

matrix-vector products as Krylov-Schur (see Table 4), but the order in which they are performed (quite consecutive compared to Krylov-Schur) allows the cache to reduce this penalty. When the local problem size is small enough, Krylov-Schur performs the matrix-vector product as fast as Jacobi-Davidson and the excess of matrix-vector products in Jacobi-Davidson is reflected in the time (see matrix-vector product times in the bottom right plot of Figure 4).

Another explanation is the fact that Jacobi-Davidson performs approximately five times more parallel reductions than Krylov-Schur (see Table 4). Some parallel reductions are performed in the orthogonalization procedure and others in the iterations of the solution of the correction equation. The latter is the main source of parallel reductions in Jacobi-Davidson, and keeping their number small is necessary for a competitive implementation.

Speedup and Scalability

Figure 4 illustrates the speedups, in the strong scaling sense, of Jacobi-Davidson and Krylov-Schur solving the test cases I, II and III. The time spent by Jacobi-Davidson in four processes is selected as the reference time for the speedup in each test case. The bottom right plot shows the speedup of the matrix-vector product taking as basis the performance in Jacobi-Davidson.

We can observe that there is a clear connection between the speedup of the matrix-vector product and the total speedup. The reason is that this operation approximately accounts for 50% of the total time in Jacobi-Davidson and just 33% in Krylov-Schur, with 256 processes. This can explain the poor speedup results for test case I. In each test case, the trend of the global speedup inherits the trend of the matrix-vector product as the number of processes grows.

Finally, Figure 5 illustrates the weak scaling scenario, plotting the time spent by the eigensolvers in the solution of the cases I (left plot) and II (right plot) with the resolution of the v direction increased. It is observed that both solvers are comparably good, with

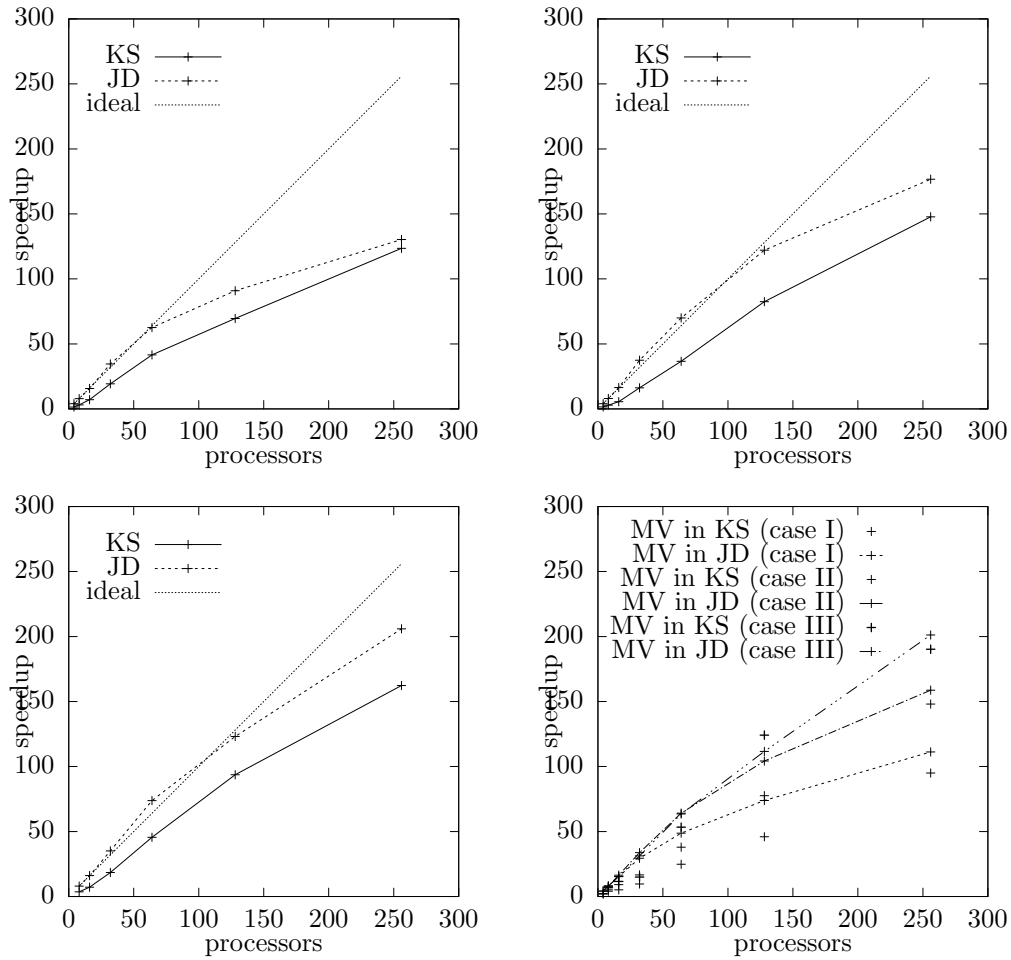

Figure 4: Speedups solving the test cases I (top left), II (top right) and III (bottom left) with Jacobi-Davidson and Krylov-Schur. Speedup of the matrix-vector product in these problems (bottom right).

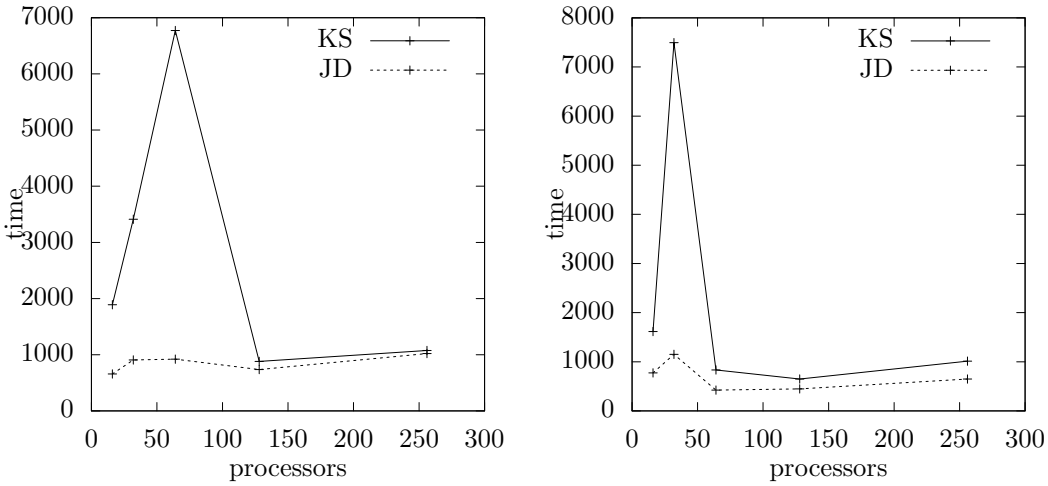

Figure 5: Time (in seconds) solving versions of the test cases I (left) and II (right) with increasing resolution in the v direction, with Jacobi-Davidson and Krylov-Schur.

the exception of the anomalous and punctual malfunction of Krylov-Schur with 16 and 32 processes due to extremely slow convergence (this may not happen with a different set of parameters).

CONCLUSIONS

We have presented a parallel implementation of the Jacobi-Davidson eigensolver for complex non-Hermitian matrices. The proposed solver incorporates all the ingredients necessary to be competitive with other solvers, such as restart and locking with Schur vectors. It is also equipped with harmonic extraction for finding interior eigenvalues, and several options for the efficient solution of the correction equation (projectors, preconditioning, and the fix parameter). The implementation has been carried out in the context of SLEPc, where the user is able to easily adjust the different parameters for the best performance.

In order to analyze the performance of the new solver, we have addressed a relevant scientific computing application, namely the computation of micro-instabilities in fusion plasmas as implemented in the GENE code. This application requires computing the rightmost eigenvalues (unstable modes) of a discretized advection dominated partial integro-differential equation, where the matrix has almost pure imaginary eigenvalues. This problem is a challenge for iterative eigensolvers.

The comparison, in terms of time, with the harmonic Krylov-Schur method is very favorable, being Jacobi-Davidson up to four times faster in some cases. However, this gain is diminished when the local problem size is small, that is, both methods become nearly equivalent when increasing the number of processes, although in absolute terms Jacobi-Davidson is still faster, at least up to 256 processes with the problems tested. The reason for this behaviour is that the Jacobi-Davidson eigensolver, in order to be competitive with respect to Krylov-Schur, needs to perform many inner iterations. In this way, the cost is dominated by the matrix-vector product operation. In the considered application, the parallel efficiency of the matrix-vector product is rather variable, depending on the configuration of the GENE parameters.

The developed solver is included in the latest version of SLEPc with all the features already described, except for the variable tolerance stopping criterion for the correction equation solver that will be included in future versions. The latter will be especially important for GENE and similar applications.

In terms of practical use, we have shown that Jacobi-Davidson is a competitive method for GENE, even without using a preconditioner. It remains as a topic for further research the addressing of the correction equation preconditioning by, for instance, the use of flexible inner-outer Krylov methods [21] (preconditioning with a Krylov method is possible in the case of implicit operators), or designing a specific preconditioner for the GENE linearized operator. Furthermore, it may be interesting to consider other iterative methods for solving the correction equation, such as deflated restarting GMRES [15], and advanced stopping criteria, such as those proposed in [11].

Acknowledgements We are indebted to Florian Merz for providing us with the test cases and for his useful suggestions. We acknowledge the computer resources provided by the Barcelona Supercomputing Center (BSC). This work was supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2009-07519.

References

[1] Peter Arbenz, Martin Becka, Roman Geus, Ulrich Hetmaniuk, and Tiziano Mengotti. On a parallel multilevel preconditioned Maxwell eigensolver. *Parallel Comput.*, 32(2):157–165, 2006.

[2] 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.

[3] 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, 2009.

[4] 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 3.1, Argonne National Laboratory, 2010.

[5] T. Dannert and F. Jenko. Gyrokinetic simulation of collisionless trapped-electron mode turbulence. *Phys. Plasmas*, 12(7):072309, 2005.

[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, 1998.

[7] Menno Genseberger. Improving the parallel performance of a domain decomposition preconditioning technique in the Jacobi–Davidson method for large scale eigenvalue problems. *App. Numer. Math.*, 60(11):1083–1099, 2010.

[8] V. Hernandez, J. E. Roman, and A. Tomas. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. *Parallel Comput.*, 33(7–8):521–540, 2007.

[9] 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, 2005.

[10] V. Heuveline, B. Philippe, and M. Sadkane. Parallel computation of spectral portrait of large matrices by Davidson type methods. *Numer. Algorithms*, 16(1):55–75, 1997.

[11] Michiel E. Hochstenbach and Yvan Notay. Controlling inner iterations in the Jacobi–Davidson method. *SIAM J. Matrix Anal. Appl.*, 31(2):460–477, 2009.

[12] C. G. J. Jacobi. Über ein leichtes Verfahren die in der Theorie der Säculärstörungen vorkommenden Gleichungen numerisch aufzulösen. *Crelle's J.*, 30:51–94, 1846.

[13] F. Merz and F. Jenko. Nonlinear interplay of TEM and ITG turbulence and its effect on transport. *Nuclear Fusion*, 50:054005, 2010.

[14] Ronald B. Morgan. Computing interior eigenvalues of large matrices. *Linear Algebra Appl.*, 154–156:289–309, 1991.

[15] Ronald B. Morgan. GMRES with deflated restarting. *SIAM J. Sci. Comput.*, 24(1):20–37, 2002.

[16] Chris C. Paige, Beresford N. Parlett, and Henk A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numer. Linear Algebra Appl.*, 2(2):115–133, 1995.

[17] J. E. Roman, M. Kammerer, F. Merz, and F. Jenko. Fast eigenvalue calculations in a massively parallel plasma turbulence code. *Parallel Comput.*, 36(5-6):339–358, 2010.

[18] E. Romero, M. B. Cruz, J. E. Roman, and P. B. Vasconcelos. A parallel implementation of the Jacobi–Davidson eigensolver for unsymmetric matrices. In J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, editors, *High Performance Computing for Computational Science – VECPAR 2010*, volume 6449 of *Lect. Notes Comp. Sci.*, pages 380–393. Springer, 2011.

[19] E. Romero and J. E. Román. A parallel implementation of the Davidson method for generalized eigenproblems. In B. Chapman, F. Desprez, G. R. Joubert, A. Lichnewsky, F. Peters, and T. Priol, editors, *Parallel Computing: From Multicores and GPU's to Petascale*, volume 19 of *Advances in Parallel Computing*, pages 133–140. IOS Press, 2010.

[20] E. Romero and J. E. Roman. A parallel implementation of the Jacobi–Davidson eigensolver and its application in a plasma turbulence code. In P. D'Ambra, M. Guarracino, and D. Talia, editors, *Euro-Par 2010, Part II*, volume 6272 of *Lect. Notes Comp. Sci.*, pages 101–112. Springer, 2010.

[21] Valeria Simoncini and Daniel B. Szyld. Flexible inner-outer Krylov subspace methods. *SIAM J. Numer. Anal.*, 40(6):2219–2239, 2003.

[22] G. L. G. Sleijpen and D. R. Fokkema. BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *Electron. Trans. Numer. Anal.*, 1:11–32, 1993.

[23] Gerard L. G. Sleijpen, Albert G. L. Booten, Diederik R. Fokkema, and Henk A. van der Vorst. Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT*, 36(3):595–633, 1996.

[24] Gerard L. G. Sleijpen and Henk A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996.

[25] Gerard L. G. Sleijpen, Henk A. van der Vorst, and Ellen Meijerink. Efficient expansion of subspaces in the Jacobi–Davidson method for standard and generalized eigenproblems. *Electron. Trans. Numer. Anal.*, 7:75–89, 1998.

[26] Andreas Stathopoulos and James R. McCombs. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. *ACM Trans. Math. Software*, 37(2):21:1–21:30, 2010.

[27] Andreas Stathopoulos, Yousef Saad, and Kesheng Wu. Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods. *SIAM J. Sci. Comput.*, 19(1):227–245, 1998.