Parallel Computing 36 (2010) 339–358

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Fast eigenvalue calculations in a massively parallel plasma turbulence code

Jose E. Roman [a,*], Matthias Kammerer [b], Florian Merz [b], Frank Jenko [b]

[a] Instituto ITACA, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain
[b] Max-Planck-Institut für Plasmaphysik, Boltzmannstr. 2, D-85748 Garching, Germany

A R T I C L E I N F O

Article history:
Received 14 October 2008
Received in revised form 31 July 2009
Accepted 2 December 2009
Available online 31 December 2009

Keywords:
Plasma physics
Complex non-Hermitian eigenproblem
Rightmost eigenvalues

A B S T R A C T

Magnetic fusion aims at providing CO_2 free energy for the 21st century and well beyond. However, the success of the international fusion experiment ITER (currently under construction) will depend to a large degree on the value of the so-called energy confinement time. One of the most advanced tools describing the underlying physical processes is the highly scalable (up to at least 32,768 cores) plasma turbulence code GENE.

GENE solves a set of nonlinear partial integro-differential equations in five-dimensional phase space by means of the method of lines, with a 4th order explicit Runge–Kutta scheme for time integration. To maximize its efficiency, the code computes the eigenspectrum of the linearized equation to determine the largest possible timestep which maintains the stability of the method. This requires the computation of the largest (in terms of its magnitude) eigenvalue of a complex, non-Hermitian matrix whose size may range from a few millions to even a billion. SLEPc, the Scalable Library for Eigenvalue Problem Computations, is used to effectively compute this part of the spectrum.

Additionally, eigenvalue computations can provide new insight into the properties of plasma turbulence. The latter is driven by a number of different unstable modes, including dominant and subdominant ones, that can be determined employing SLEPc. This computation is more challenging from the numerical point of view, since these eigenvalues can be considered interior, and also because the linearized operator is available only in implicit form. We analyze the feasibility of different strategies for computing these modes, including matrix-free spectral transformation as well as harmonic projection methods.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

It is well known that solving large systems of linear equations on parallel computers efficiently is not trivial. The main difficulty comes from the fact that most algorithms are not scalable to a large number of processors, particularly direct solvers and preconditioners. In most applications, choosing a good preconditioner is of paramount importance when using an iterative linear solver, so that convergence is as fast as possible, but typically preconditioners lose effectiveness when the number of processors grow. Only in recent years, methods such as parallel multi-grid have been developed that scale well to hundreds or even thousands of processors.

In this work, we focus on an eigenvalue calculation that takes place in a plasma physics application. One of the goals of this paper is to illustrate that difficulties also arise in the context of eigencomputations, similarly to linear systems. The point is that straightforward algorithms cannot deal with difficult problems that arise in many applications, in which some kind of preconditioning is required for a successful resolution. Preconditioning eigenvalues is a topic that is currently attracting

* Corresponding author. Tel.: +34 963877356; fax: +34 963877359.
E-mail addresses: jroman@itaca.upv.es, jroman@dsic.upv.es (J.E. Roman), kammerem@ipp.mpg.de (M. Kammerer), flm@ipp.mpg.de (F. Merz), fsj@ipp.mpg.de (F. Jenko).

0167-8191/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2009.12.001

much interest in the numerical algorithms community. Multi-level eigensolvers, which have a potential for good scalability, are recently being investigated. Apart from these, preconditioned eigensolvers can use a standard preconditioner, taken from the linear system practice, which will likely suffer from scalability problems. If one chooses to use a simple preconditioner that scales well instead, then there is the risk that convergence is not sufficiently enhanced.

Most preconditioners need to have knowledge of the associated operator to some extent, because the main idea of preconditioning is to cheaply approximate the inverse of such operator, or, more precisely, its matrix representation. In some applications, such as the one described in this paper, the matrix form of the operator is not explicitly available and the problem has to be solved exclusively by exploiting the information provided by matrix–vector products. This severely restricts the number of alternatives available for preconditioning. One such alternative is to use a few iterations of a Krylov linear solver as a preconditioner.

In this paper, we will not use a genuinely preconditioned eigensolver. Instead, a Krylov eigensolver with inexact spectral transformation will be applied. Spectral transformations can be viewed as a form of preconditioning for eigenproblems. When this technique is used, a linear system has to be solved in each iteration of the eigensolver. In our case, due to the unavailability of the explicit operator, one can either use an unpreconditioned iterative solver, or an iterative solver preconditioned by another iterative solver. The latter case is a nested inner–outer iteration such as FGMRES [1], where a few iterations of a Krylov linear solver is carried out in each iteration of a flexible variant of GMRES. See [2] for additional information on flexible inner–outer iterations. In any case, the effectiveness of the spectral transformation will be rather poor in our particular application, as the numerical experiments will show.

An alternative is to completely avoid any kind of preconditioning, and try to exploit the subspaces generated by iterative eigensolvers in a smart way. This can be accomplished by a combination of restarting techniques and improved extraction techniques (e.g., harmonic projection), in such a way that we are able to drive convergence toward the desired part of the spectrum. These techniques may not work for all possible scenarios, but they have proved very useful in our application.

The application addressed in this paper is the simulation of plasma turbulence phenomena, as realized in the GENE code [3,4]. This is a good example of how modern science endeavors increasingly rely on numerical simulation tools. The complexity of the underlying model equations makes it a very challenging application. In particular, GENE solves a set of nonlinear partial integro-differential equations in five-dimensional phase space by means of the method of lines, with a 4th order explicit Runge–Kutta scheme for time integration. 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 (which have a large imaginary part). In this paper, we focus on strategies for approximating this part of the spectrum.

All eigenvalue computations in GENE are carried out by means of the SLEPc library [5]. SLEPc is a parallel library that provides a number of tools for solving large-scale eigenvalue problems, including modern eigensolvers and spectral transformations. With SLEPc, it is possible to address different types of eigenproblems, including singular value computations, with either real or complex arithmetic. SLEPc tries to keep up with state-of-the-art developments in the field of iterative eigensolvers, offering a variety of solvers from which the user can choose. The philosophy of encapsulating the know-how in the form of a software library has proved very effective in many fields. In this sense, with this work we also want to place emphasis on how helpful software libraries are in bridging the gap between algorithmic research and applications, thus constituting an effective technology transfer mechanism for novel developments that would otherwise be limited to theoretical speculation.

The rest of the paper is organized as follows. Section 2 gives an overview of the application, showing the structure of the linear gyrokinetic equations, and discussing the eigenproblems that arise in this context. Section 3 describes the functionality of SLEPc, and focuses on the Krylov–Schur method that is used for the solution of the eigenproblem. Considerable emphasis is placed on the techniques employed for getting the rightmost eigenvalues: spectral transformation and harmonic projection. In Section 4, we provide results of some numerical experiments carried out with the code in the context of several realistic simulation scenarios. The analysis is carried out in terms of convergence and also in terms of parallel efficiency and scalability. We end up with some conclusions in Section 5.

2. Linear gyrokinetics

The magnetized plasmas that occur in the context of magnetic confinement fusion exhibit a large number of "microscopic" (i.e. gyroradius-scale) instabilities driven by the background density and temperature gradients. These microinstabilities constitute the drive for the turbulence that leads to the anomalous transport determining the energy confinement time. The starting point for nonlinear investigations, which address the physically and technically very important plasma turbulence problem, therefore has to be a thorough study of the underlying linear phenomena.

2.1. Linear gyrokinetic equations

In strongly magnetized, dilute plasmas, as they occur in fusion experiments, the motion of ions and electrons perpendicular to the magnetic field is dominated by a rapid gyration around the field lines. This periodic motion is not relevant for most investigations, which usually focus on observables that are related to much slower time scales like, e.g., the net particle transport. As collisions are weak due to low plasma densities and high temperatures, kinetic effects have to be taken into account. However, the fast gyration can be removed analytically from the Maxwell–Boltzmann system of equations, reducing the dimensionality of the problem from six to five (three spatial and two velocity space dimensions). The linearized version of this so-called gyrokinetic equation in the δf flux tube limit can be written schematically in an appropriate normalization [6] as

$$\frac{\partial g}{\partial t} = \mathcal{L}[g]. \tag{1}$$

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. Note that g does not describe the full plasma, it is only the perturbation about a Maxwellian background distribution function F_0. The time independent, complex, non-Hermitian integro-differential operator $\mathcal{L}$ is given by

$$\mathcal{L}[g] = -\left(\omega_n + \left(v_\parallel^2 + \mu B_0 - \frac{3}{2}\right)\omega_{Tj}\right)F_{0j} i k_y \chi - \frac{v_{Tj}}{JB_0} v_\parallel \Gamma_{jz} - \frac{T_{0j}(2v_\parallel^2 + \mu B_0)}{q_j B_0}(K_y \Gamma_{jy} + K_x \Gamma_{jx}) + \frac{v_{Tj}}{2JB_0}\mu \partial_z B_0 \frac{\partial f_j}{\partial v_\parallel} + \langle C_j(f) \rangle,$$

where $\chi,\ \Gamma$ and f are connected to g via linear (integro-differential) operators. The first term, which contains the density and temperature gradients $\omega_n,\ \omega_T$ of F_0, constitutes the drive of the system, the second term describes the dynamics parallel to the magnetic field (v_{Tj} is the normalized thermal velocity of species j, J the Jacobian of the metric), the third one is the curvature term with the curvature factors $K_x,\ K_y$, which arises due to the non-Euclidean geometry that is necessary to describe toroidal fusion devices (q_j is the electric charge), the fourth term describes magnetic mirror effects along the field line, and the last term is a linearized Landau–Boltzmann collision operator. For a more detailed description of the gyrokinetic equations as implemented in the GENE code, see [6].

2.2. GENE code

The GENE code [3,4] is a massively parallel plasma simulation code written in Fortran 90/95, which numerically solves the (linear and nonlinear) gyrokinetic equations. It follows an Eulerian approach, meaning that the phase space is discretized using a fixed grid and the time evolution of the distribution function g is then computed on this grid. The x and y directions are treated spectrally, and the finite extent of the simulation domain already leads to a discretization in these directions. In the z and $v_\parallel$ directions, finite differences are used to discretize the parallel derivatives. The integrals in the μ and $v_\parallel$ coordinates that appear in the computation of χ are performed using a Gauss–Legendre and modified trapezoid rule, respectively.

Physically, g is a scalar field of the three spatial, two velocity space, and the species coordinates; mathematically, it can be viewed as the complex state vector of the system. Discretization makes this state vector finite dimensional and turns the operator $\mathcal{L}$ into a matrix, so that the right hand side of the linear gyrokinetic equation is formally a matrix–vector product. However, the operator matrix is never computed explicitly but implemented in a highly parallelized and efficient matrix-free form. This is necessary to perform computations for reasonably resolved state vectors, which typically consist of several hundred thousand entries for linear simulations up to a few billion entries for nonlinear problems.

For nonlinear and traditionally also for linear simulations, the time derivative is discretized using an explicit 4th order Runge–Kutta scheme and the gyrokinetic equation is solved as an initial value problem. This means that the distribution function is initialized and then evolved in time, and after a transient phase, the key features of the system like the fluctuation level or the heat and particle transport show a constant exponential growth for linear simulations or reach a quasistationary level for nonlinear simulations, which are then independent of the initialization. While this approach is the only possibility for the nonlinear problem, the linear problem can also be solved as an eigenvalue problem, as is described in this article.

2.3. Eigenvalue problems in GENE

Since the eigenspectrum of the operator matrix fully defines the linear system, it is very beneficial to have an eigenvalue solver that can access all parts of this spectrum. Several facts call for an iterative eigenvalue solver in this context. The first argument is that a direct solver would take too long for the matrix sizes considered here, and typically only a few eigenvalues (largest absolute value or rightmost eigenvalues) are really of interest. In addition, the linear operator is an integro-differential operator and therefore not sparse, explicit representations of the operator matrix are therefore not suitable for computations. Furthermore, since GENE is parallelized, it is advantageous if the eigenvalue solver is parallelized as well. The iterative eigenvalue solver SLEPc meets all these requirements. As an iterative, parallelized eigenvalue solver which only requires a routine that computes the right hand side of the linear gyrokinetic equation given a test vector g, it can directly use the routines written for the initial value solver and take advantage of their efficient parallel implementation.

The eigenvalue solver has two applications in GENE. The first one is the optimization of the initial value mode, where GENE uses an explicit Runge–Kutta scheme for the discretization of the time derivative, which is only conditionally stable. For the linear gyrokinetic equation described above, this sets a maximum on the time step width Δt, which depends on the spectrum of the linear operator and the stability region of the specific time stepping scheme. The spectrum of the operator $\mathcal{L}$ in the complex plane is mostly aligned with the imaginary axis, so that the eigenvalue that yields the most restrictive time step limit is the one with the largest absolute value. With the interface to SLEPc, it is possible to easily compute this part of the spectrum and to determine the optimal time step prior to an initial value computation and thus to minimize the computation time.

The second application is the computation of subdominant, unstable modes (i.e. the modes with positive real part of the eigenvalue). In typical fusion plasmas, the background temperature and density gradients lead to the destabilisation of one or a few eigenmodes, which then grow exponentially in time. With an initial value solver, only the most unstable mode is accessible, since it dominates the distribution function after a short amount of time. For these investigations, an eigenvalue solver is clearly superior, provided that it is comparable to the initial value solver in speed. Unfortunately, the extent of the eigenvalue spectrum along the imaginary axis is typically three orders of magnitude larger than along the positive real axis, and the real and imaginary part of the eigenvalues of interest are comparable, so that efficient algorithms to find internal eigenvalues have to be used. Several methods to compute these eigenvalues are implemented in SLEPc, they will be presented and evaluated in the context of linear gyrokinetics in the next sections.

The GENE–SLEPc interface has already been used to study some interesting features of mode transitions in linear gyrokinetics [7].

3. Eigenvalue computation with SLEPc

In the current scene of computational science software, we often encounter complex application codes developed by teams of experts in the particular research field, as is the case of GENE. In this context, it is very important to make use of specialized software libraries in order to cope with the increasing complexity of applications. Software reuse allows the application programmer to shorten the development cycle, as well as to exploit the expertise that is encapsulated in that software. This latter feature is especially appealing in the case of numerical software addressing mathematical problems, since numerical algorithms can be very difficult and tricky to implement for non-experts. Modern numerical libraries are designed to provide enough flexibility to make software reuse feasible in many different contexts, but also with other goals in mind such as numerical robustness, computational efficiency, portability and scalability in a wide variety of parallel computer systems, extensibility and interoperability with other software.

GENE makes use of software libraries mainly for two tasks: fast Fourier transform and eigenvalue computation. Here, we focus on the latter.

In the last years, research on numerical methods for the solution of large sparse eigenvalue problems has materialized in the form of publicly available software, see [8] for an updated survey including download information. The software ranges from simple Fortran or Matlab subroutines to sophisticated parallel libraries providing some of the desirable features mentioned above.

GENE requires a parallel library, based on the message-passing programming paradigm with MPI, for the solution of a complex non-Hermitian eigenvalue problem, as described in Section 2.3. These very specific requirements considerably reduce the list of suitable software. Only ARPACK, Anasazi and SLEPc are currently able to cope with such kind of problem.

ARPACK [9] is a collection of Fortran 77 subroutines designed to solve large-scale standard and generalized eigenvalue problems ($Ax = \lambda x$ and $Ax = \lambda Bx$), both symmetric and non-symmetric, in either real or complex arithmetic. It is based on an algorithmic variant of the Arnoldi process called the implicitly restarted Arnoldi method. Although this library represented an unprecedented leap forward in the solution of non-symmetric eigenproblems, and it can still be useful, today we can consider it superseded by implementations of the Krylov–Schur method in Anasazi and SLEPc.

Anasazi [10] is a parallel object-oriented software written in C++, that provides a generic interface to a collection of algorithms for solving large-scale standard eigenvalue problems. The three algorithms currently offered are block Krylov–Schur, block Davidson and LOBPCG. From these, only Krylov–Schur can be employed for non-Hermitian eigenproblems.

SLEPc, the Scalable Library for Eigenvalue Problem Computations [5], is a software library for the solution of large, sparse eigenvalue problems on parallel computers. It can be used for eigenproblems formulated in either standard or generalized form, both Hermitian and non-Hermitian, with either real or complex arithmetic. SLEPc is built on top of PETSc, the Portable Extensible Toolkit for Scientific Computation [11], a parallel framework for the numerical solution of problems arising in applications modeled by partial differential equations. In PETSc all the code is built around a set of objects that encapsulate data structures and solution algorithms. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The data objects include management of index sets, vectors and sparse matrices in different formats, as well as basic support for structured and unstructured meshes. Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers. For solving linear systems of equations, PETSc provides a variety of iterative methods such as Conjugate Gradient and GMRES, that can be combined with different preconditioners such as the incomplete LU factorization. SLEPc extends PETSc with all the functionality necessary for the solution of eigenvalue problems. It provides uniform and efficient access to a growing number of iterative eigensolvers. In particular, SLEPc implements the Krylov–Schur method, which is described in Section 3.1. The user is able to easily switch among different eigensolvers by simply specifying the method at run time. Apart from the solver, many other options can be specified such as the number of eigenvalues to compute, the requested tolerance, or the portion of the spectrum of interest. The latter is a central aspect in this paper and will receive considerable attention throughout this section.

SLEPc also provides built-in support for different spectral transformations such as the shift-and-invert technique. When such a transformation is applied, the matrix inverses such as $(A-\sigma B)^{-1}$ are not computed explicitly but handled implicitly via a linear system solver provided by PETSc. When this type of computation is required, SLEPc is much more easy to use compared to ARPACK, in which the user has to provide all the coding for solving the linear systems or calling an external library for that task. Spectral transformations are considered in detail in Section 3.2.

GENE has opted for SLEPc for solving the eigenvalue problems. From the advantages of SLEPc with respect to Anasazi, we could point out an easier programming paradigm and a simple integration with Fortran, the programming language used in GENE. A further benefit of SLEPc is the harmonic projection feature, described in Section 3.3 below, that is currently unique in the eigenvalue software scenario.

A convenient characteristic is SLEPc's data-structure neutral implementation, meaning that the application code uses an abstract interface that is independent of how data are represented internally. For instance, a matrix object is manipulated only through a set of standardized operations, that are implemented differently for different matrix formats. This paradigm gives the application programmer the possibility to create new matrix types. These so-called shell matrices provide only the operations that have been defined by the user, and allow the development of matrix-free applications, that is, applications that do not compute the matrix explicitly but rather use it in the form of, e.g., matrix–vector products. GENE is implemented in this way, so the matrix that represents the operator is never built explicitly.

3.1. Krylov eigensolvers

In applications where only a localized part of the spectrum of a large-scale eigenvalue problem is required, traditional eigensolvers (usually called dense or direct) are not appropriate. Instead, another class of methods (usually called sparse or iterative) has to be employed. The key feature of such methods is that they do not modify the problem matrix, thus preserving the sparsity pattern. These methods use the matrix only in matrix–vector product operations, and are appropriate only if these products can be computed with linear complexity, in other words if the matrix is sparse or structured.

SLEPc provides a general framework for the implementation of iterative eigensolvers. For background material on iterative eigensolvers, the reader is referred to [12,13]. Most of these solvers can be considered projection methods, since they are based on projecting the problem onto a certain subspace. From the many methods available, here we focus our attention on the so-called Krylov methods, and particularly the Krylov–Schur method, which is currently the most advanced solver available in SLEPc.

To be more specific, consider the equation

$$Ax_i = \lambda_i x_i, \tag{2}$$

where A is a square matrix of order n. In the case of GENE, A is complex and non-Hermitian. The objective is to compute a small number of eigenpairs, $(\lambda_i, x_i),\ i = 1,\dots,k$, with $k \ll n$, where λ_i is a complex scalar called the eigenvalue, and x_i is an n-vector called the eigenvector.

The basic principle of projection methods is to find the best approximations to the eigenvectors in a given subspace of small dimension. This is achieved with the so-called Rayleigh–Ritz procedure, described next.

Given an $n\times m$ matrix V, with $k \leqslant m \ll n$, whose columns v_i constitute an orthonormal basis of a given subspace $\mathcal{V}$, i.e. $V^*V = I_m$ and $\operatorname{span}\{v_1, v_2,\dots,v_m\} = \mathcal{V}$, the Rayleigh–Ritz procedure consists in computing $H = V^*AV$, which is a square matrix of order m, and then solving the eigenvalue problem associated with it, $Hy_i = \theta_i y_i$. The approximate eigenpairs $(\tilde{\lambda}_i, \tilde{x}_i)$ of the original eigenvalue problem (2) are then $\tilde{\lambda}_i = \theta_i$ and $\tilde{x}_i = Vy_i$, which are called Ritz values and Ritz vectors, respectively. Note that Ritz vectors belong to subspace $\mathcal{V}$. It can be shown that the Ritz values and vectors are the best possible approximations in that subspace. Matrix H is the projection of A onto $\mathcal{V}$, hence the name of this class of methods.

The quality of the Ritz approximations depends on how well the subspace $\mathcal{V}$ approximates an invariant subspace of A. In principle, a good choice for $\mathcal{V}$ is the Krylov subspace associated with matrix A and a given initial vector x_1,

$$\mathcal{K}_m(A, x_1) = \operatorname{span}\{x_1, Ax_1, A^2x_1, \dots, A^{m-1}x_1\}. \tag{3}$$

The method of Arnoldi computes an orthonormal basis of the Krylov subspace and at the same time computes the projected matrix H, all this in an efficient and numerically stable way. In brief, the Arnoldi algorithm computes the m columns of V sequentially, where column v_{j+1} is the result of orthogonalizing Av_j with respect to previous columns, and normalizing. The orthogonalization is carried out by means of a Gram–Schmidt procedure, that removes all the components in the directions of $v_1,\dots,v_j$. The computed quantities satisfy a relation of the form

$$AV_m = V_mH_m + \beta v_{m+1}e_m^*, \tag{4}$$

where H_m is an upper Hessenberg matrix, i.e. $h_{ij} = 0$ for $i > j+1$, and e_m is the mth vector of the canonical basis. The last term of the Arnoldi relation is the residual and gives an indication of how close is $\mathcal{K}_m(A, x_1)$ to an invariant subspace. In particular, β is used to assess the accuracy of the computed Ritz pairs. See [12] for additional details.

The Arnoldi algorithm is very simple and easy to implement, and it is very efficient computationally provided that the value of m is not too large. Unfortunately, in practical applications it is often the case that convergence is not achieved with a moderate m. Therefore, a restarted variant of the Arnoldi process must be used in practice. Restarting means running the algorithm again trying to exploit the spectral information available in the quantities computed so far. Several restarting schemes exist. The simplest one is called explicit restart, and consists in explicitly computing a new initial vector x_1 that is rich in the direction of the wanted eigenvectors, for instance as a linear combination of a subset of the computed Ritz vectors. In many cases, this simple mechanism is not sufficient and a good convergence also requires filtering out the components of x_1 in the direction of unwanted eigenvectors. There is a clever way of doing this called implicit restart. Instead of building the new x_1 explicitly, this method is based on building an Arnoldi decomposition of order $p < m$ in such a way that it retains the relevant information from the decomposition of order m, and then this small decomposition is extended again to order m. This method works remarkably well but is difficult to implement due to potential numerical instabilities. The Krylov–Schur algorithm [14], that we describe next, provides a simpler form of implicit restart without numerical difficulties. An implementation of this method is currently the default eigensolver in SLEPc, and has been used for solving the eigenproblems presented in this paper.

The Krylov–Schur method is defined by generalizing the Arnoldi decomposition of order m, (4), to a Krylov decomposition of order m,

$$AV_m = V_mB_m + v_{m+1}b_m^*, \tag{5}$$

in which matrix B_m is not restricted to be upper Hessenberg and b_m is an arbitrary m-vector. The space of the decomposition is that spanned by the columns of V_m, which need not be orthogonal in general.

The Krylov–Schur method is based on two observations. First, Krylov decompositions are invariant under similarity transformations, that is, the space of the decomposition remains the same if B_m is replaced by $W^{-1}B_mW$ for any non-singular W. In particular, a unitary similarity transformation results in

$$AV_mQ = V_mQ(Q^*B_mQ) + v_{m+1}b_m^*Q, \tag{6}$$

where Q is a unitary matrix, and maintains orthogonality of the space basis if V_m had orthogonal columns. Similarity transformations can be used to transform any Krylov decomposition to a Krylov–Schur decomposition,

$$A\widetilde{V}_m = \widetilde{V}_mT_m + v_{m+1}t_m^*, \tag{7}$$

where T_m is upper triangular (or quasi-triangular in the case of real arithmetic).

The second observation is that a Krylov–Schur decomposition can be truncated at any point. If we write it as

$$A\begin{bmatrix}\widetilde{V}_1 & \widetilde{V}_2\end{bmatrix} = \begin{bmatrix}\widetilde{V}_1 & \widetilde{V}_2\end{bmatrix}\begin{bmatrix}T_{11} & T_{12}\\ 0 & T_{22}\end{bmatrix} + v_{m+1}[t_1^* \quad t_2^*], \tag{8}$$

then

$$A\widetilde{V}_1 = \widetilde{V}_1T_{11} + v_{m+1}t_1^* \tag{9}$$

is also a Krylov–Schur decomposition, that can be extended to order m again. The key of the Krylov–Schur method is to keep in T_1 the relevant spectral information contained in T_m, and this is achieved by appropriately sorting the eigenvalues in the diagonal of T_m. See [14] for details.

In SLEPc's parallel implementation of the Krylov–Schur method, the problem matrix A is distributed row-wise among all processors, in PETSc style, and so are the vectors v_j. All operations involving these vectors are carried out in parallel with PETSc primitives. The matrix vector product implemented in PETSc has good parallel efficiency in most applications. In contrast, the projected matrix B_m is usually so small that storing it distributedly would be disadvantageous, thus it is replicated in every processor, and so are computations involving it.

An important requirement for an implementation of a Krylov subspace eigensolver is the stability and efficiency of the orthogonalization procedure. The orthogonalization used by default in SLEPc is a Classical Gram-Schmidt procedure with conditional reorthogonalization and norm estimation. This procedure has an excellent balance between stability and parallel performance, see [15] for details.

3.2. Spectral transformation

The Krylov–Schur method described in the previous subsection is conceptually very simple, but also very effective compared to older algorithms. However, its practical use requires to address two questions, (a) which part of the spectrum is computed and (b) how does it behave in terms of convergence. It turns out that both questions are somewhat related. A Krylov subspace, (3), usually contains good approximations of the eigenspaces associated to eigenvalues with largest magnitude. That is why the simple Arnoldi method typically will return only the dominant eigenvalues (those with largest magnitude). The theory of convergence of these eigensolvers is very complicated, especially in the case of non-Hermitian problems. Roughly speaking, we could state that convergence depends on magnitude as well as on relative separation of the eigenvalues. In summary, Krylov methods are good at computing eigenvalues located in the periphery of the spectrum, especially if they are well separated. The filtering effect offered by the Krylov–Schur restart scheme can be used to modify the natural trend of Krylov subspaces, and this is done by appropriately ordering the Ritz values in the projected matrix, as explained before. In this way, it is possible to seek for smallest magnitude eigenvalues, or rightmost or leftmost eigenvalues, or even eigenvalues located in the interior of the spectrum. Nevertheless, in many circumstances the filtering is not good enough to eliminate the components in the direction of dominant eigenvectors. This is the case in GENE, where the rightmost eigenpairs are those of interest, but these are very small compared to those with largest magnitude. A workaround is to try to enhance convergence of these very eigenvalues, by making them larger and improving their separation. This effect can be achieved by spectral transformations, described below, and also by a cheaper alternative described in Section 3.3.

The basic idea of a spectral transformation is to replace the original eigenproblem (2) by a new one in which eigenvalues are mapped to a new position while eigenvectors remain unchanged. More precisely, we now want to solve the equation

$$Tx_i = \theta_i x_i, \tag{10}$$

where T is a matrix related to A in some way, and the eigenvector x_i is the same as in (2). Typically, the transformation is chosen in such a way that there is a simple mapping between the computed eigenvalues, θ_i, and the eigenvalues of the original problem, λ_i, and also that the eigenvalue distribution of the transformed spectrum is more favorable in terms of convergence. In this paper, we consider only rational spectral transformations, namely shift-and-invert and Cayley transform. Both of them can be used to enhance convergence of eigenvalues in the neighborhood of a given value in the complex plane.

The shift-and-invert spectral transformation is defined by the matrix

$$T_{SI} = (A - \sigma I)^{-1}. \tag{11}$$

This transformation is effective for finding eigenvalues near σ since the transformed eigenvalues θ_i that are largest in magnitude correspond to the original eigenvalues λ_i that are closest to the shift σ in absolute value. Additionally, if σ is sufficiently close to the sought-after eigenvalues, then these will become more separated, so Krylov eigensolvers such as Arnoldi will be able to extract them much easier. In this case, the relation between the eigenvalues of both problems is $\theta_i = (\lambda_i - \sigma)^{-1}$.

The shift-and-invert spectral transformation implies the solution of a system of linear equations whenever a matrix–vector product with T_{SI} is required. In SLEPc, this system is solved via linear solvers provided by PETSc, either direct or iterative. Direct methods are generally recommended for solving the systems accurately. We will refer to exact shift-and-invert when a direct linear solver is used. In contrast, inexact shift-and-invert solves the related linear systems only approximately by employing an iterative linear solver that approximates the solution up to a given tolerance. As we will discuss later on, in the case of GENE the choice of linear solver will be highly restricted since the coefficient matrix is not available explicitly.

Shift-and-invert has been successfully employed in many different application areas, particularly in structural dynamics where a symmetric generalized eigenproblem, $Ax = \lambda Bx$, has to be solved. The systematic use of shift-and-invert in the context of Krylov eigensolvers started in the early 1980s, with the work by Ericsson and Ruhe [16], and Scott [17]. The method was later improved in [18], particularly for the case of singular B. These works used a direct linear solver via a triangular factorization. The factorization needs to be computed only once and is reused throughout all the eigencomputation. Also, in symmetric problems the factorization provides inertia information, which is essential when one is interested in computing all eigenvalues in a given interval. This technique called spectrum slicing was fully realized in [19], and has been recently implemented by making use of SLEPc [20]. Since the availability of robust non-Hermitian eigensolvers, shift-and-invert has also been applied to such problems, see for instance [21] and [22].

A practical consideration when using shift-and-invert is that the convergence criterion that is used by the eigensolver refers to the transformed problem. This means that the computed eigenpairs (θ_i, x_i) are accurate in the backward error sense regarding the eigenproblem (10), but what we want is to get accurate eigenpairs (λ_i, x_i) of the original problem. In [16], a correction to the eigenvector is proposed in order to attain smaller residual error in the original eigenproblem. This is further analyzed in [23], and a similar argument can be applied to non-symmetric problems [12, §7.6.8]. However, these criteria are difficult to implement robustly in a general-purpose library covering many different types of eigenproblems. The approach in SLEPc is simply to test convergence on the transformed problem, and let the user decide if this is appropriate or not. Practical experience shows that exact shift-and-invert will result in accurate eigenpairs of the original problem, provided that the shift σ is not too far from the wanted eigenpairs.

The above considerations apply to exact shift-and-invert. In the case of inexact shift-and-invert, we have to add the error introduced by the inexact solution of linear systems. Thus there are more chances to get an inaccurate result. The behaviour of inexact eigensolvers is being studied theoretically in simple algorithms such as inverse iteration, see [24] and references therein. The case of Krylov methods will be discussed below in the context of the Cayley transform. A related thing is that iterative linear solvers may have slow convergence, depending on different aspects such as conditioning. It turns out that the closer σ is to an eigenvalue, the worse the conditioning of matrix T_{SI} will be. So it is often convenient to use a farther shift, but this requirement is contrary to the one mentioned above.

The generalized Cayley transform is defined by the matrix

$$T_C = (A - \sigma I)^{-1}(A - \tau I), \tag{12}$$

where τ is a scalar sometimes called the anti-shift. This transformation maps the eigenvalues in a slightly different way, see [25]. However, in the context of Krylov methods, it is mathematically equivalent to shift-and-invert, because $T_C = I + (\sigma - \tau)T_{SI}$ and Krylov subspaces are invariant under matrix shifts. Therefore, the Cayley transform is effective for finding eigenvalues near σ as well. In this case, the relation between the eigenvalues of the original and transformed problems is $\theta_i = (\lambda_i - \sigma)^{-1}(\lambda_i - \tau)$. The interest of the Cayley transform lies in its better numerical properties with respect to shift-and-invert, especially in the context of inexact solution of the associated linear systems [12, §11.2].

The Cayley transformation has been used for the particular case that the rightmost eigenvalues of a non-Hermitian matrix have to be computed. That is exactly what is required in GENE, as well as in many other application areas concerned with the stability analysis of different physical phenomena. In [26], Meerbergen and Roose survey different alternatives for computing rightmost eigenvalues, including the use of shift-and-invert and Cayley with Arnoldi, and also discuss the case of inexact solution of the linear systems with iterative solvers. In [27], the analysis of inexact Cayley is developed further. Based on the analysis, the authors confirm the impression that σ plays a conflicting role in the sense that when moved to the right the linear systems are easier to solve, but then the transformed eigenvalues are less well separated. An iterative scheme is proposed in which the shift is updated at each Arnoldi restart, but it is considered only for computing one eigenpair.

The analysis in [28] is more closely related to our needs in this work, since several rightmost eigenvalues are computed by an implicitly restarted Arnoldi method with an inexact Cayley transform, solving linear systems with a Krylov linear solver such as GMRES. The rationale of using Cayley instead of shift-and-invert is that shift-and-invert maps eigenvalues close to σ to those of largest magnitude, but also maps the eigenvalues far from the shift to zero. This results in potential convergence difficulties for the linear solver due to a large condition number of the coefficient matrix. In contrast, conditioning is improved in the case of Cayley transform, since eigenvalues far from the shift are mapped close to one instead of zero. The heuristic advocated in [28] is to choose σ and τ so that the wanted eigenvalues satisfy $2\sigma - \tau < \Re(\lambda_i) < \sigma < \tau$.

As a summary of spectral transformations, one may draw the conclusion that their use is not completely satisfactory, particularly when computing rightmost eigenvalues of large non-Hermitian matrices. If the shift is too far from the wanted eigenvalues, then accuracy may be bad, but if we use a shift that is closer to the spectrum then difficulties arise in the iterative linear solvers. A solution would be a method that dynamically adjusts the shift, moving it closer to the spectrum, but robust enough so that linear systems need not be solved very accurately. This is the spirit of preconditioned eigensolvers such as Davidson and Jacobi-Davidson, see [29,30]. These eigensolvers are not available in SLEPc yet, though there are plans to implement them in the near future.

3.3. Harmonic projection methods

In Section 3.1, we defined the Ritz values of a matrix A with respect to a Krylov subspace $\mathcal{K}_m(A, x_1)$. Harmonic Ritz values of a matrix A are the reciprocals of the Ritz values of A^{-1} with respect to the subspace $A\mathcal{K}_m(A, x_1)$ [31,32]. The definition suggests the idea that an eigensolver could build a basis of the Krylov subspace and then change the last step of the Rayleigh–Ritz projection procedure in order to extract harmonic Ritz values instead of standard Ritz values. In that way, it would be possible to compute the smallest eigenvalues of A without explicitly computing its inverse.

More precisely, since harmonic Ritz values with respect to Krylov subspaces are not shift-invariant (contrarily to Ritz values), we will be interested in approximating the Ritz values of $(A - \kappa I)^{-1}$, where the scalar κ will be called the target and play an analogous role as the shift σ in spectral transformations. In this case, harmonic Ritz values will approximate eigenvalues of A closest to κ.

The concept of harmonic Ritz values has been used for computing interior eigenvalues of non-Hermitian matrices [33,34], in the context of the Arnoldi method. The idea can be applied also to the Krylov–Schur method, as hinted next. The key point is to avoid building $(A - \kappa I)^{-1}$ explicitly, and also to avoid building two bases, one for $\mathcal{K}_m$ and another for $(A - \kappa I)\mathcal{K}_m$.

Suppose a Krylov decomposition, (5), has been built. After shifting, we have

$$(A - \kappa I)V_m = V_m(B_m - \kappa I) + v_{m+1}b_m^*, \tag{13}$$

where V_m is the orthonormal basis of the Krylov subspace. Now, define a second basis $U_m := (A - \kappa I)V_m$, then we have $U_m = V_m(B_m - \kappa I) + v_{m+1}b_m^*$, or

$$V_m = U_m(B_m - \kappa I)^{-1} - v_{m+1}g^*, \tag{14}$$

with $g := (B_m - \kappa I)^{-*}b_m$. Eq. (14) can be seen as a Krylov decomposition of $(A - \kappa I)^{-1}$, so if θ_i is an eigenvalue of $(B_m - \kappa I)^{-1}$ then $\theta_i^{-1} + \kappa$ is a harmonic Ritz value of A, that approximates an eigenvalue close to the target. Note that this scheme is suggestive of doing shift-and-invert on the projected matrix, B_m, which is computationally more benign than doing the inversion on the large matrix.

The above approach is very naive and has no practical interest, because it would require to build and store the basis U_m, and moreover to orthogonalize it explicitly. However, it is possible to overcome these difficulties by rearranging the equations. Suppose we apply the Rayleigh–Ritz procedure to matrix $(A - \kappa I)^{-1}$, but with respect to the subspace defined by the basis U_m. In that case, if the Ritz pair is denoted as $(\tilde{\theta}, U_m z)$, the projection can be expressed with the Galerkin condition

$$U_m^*((A - \kappa I)^{-1}U_m z - \tilde{\theta}U_m z) = 0, \tag{15}$$

or

$$U_m^*(A - \kappa I)^{-1}U_m z = \tilde{\theta}U_m^*U_m z. \tag{16}$$

Then, writing this equation in terms of the orthogonal Krylov basis V_m by expanding the definition of U_m,

$$V_m^*(A-\kappa I)^*V_m z = \tilde{\theta}V_m^*(A-\kappa I)^*(A-\kappa I)V_m z. \tag{17}$$

From (13), this is equivalent to

$$(B_m-\kappa I)^*z = \tilde{\theta}\big((B_m-\kappa I)^*(B_m-\kappa I)+b_m b_m^*\big)z, \tag{18}$$

that is, the projected problem is a symmetric-definite generalized eigenvalue problem. The values $\tilde{\theta}$ are Ritz values of $(A-\kappa I)^{-1}$ with respect to U_m, or alternatively, the values $\tilde{\theta}^{-1}+\kappa$ are harmonic Ritz values of A. Eq. (18) can be formulated as a standard eigenvalue problem, by pre-multiplying the equation by the inverse of $(B_m-\kappa I)^*$, to get

$$z = \tilde{\theta}((B_m-\kappa I)+(B_m-\kappa I)^{-*}b_m b_m^*)z, \tag{19}$$

or, equivalently, using the definition of g of (14),

$$z = \tilde{\theta}((B_m-\kappa I)+g b_m^*)z. \tag{20}$$

Rearranging terms, the projected problem can be written as

$$(B_m+g b_m^*)z = (\tilde{\theta}^{-1}+\kappa)z, \tag{21}$$

that is, the eigenvalues of matrix $B_m+g b_m^*$ are the harmonic Ritz values of A. The corresponding harmonic Ritz vectors are equal to $U_m z$, or $(A-\kappa I)V_m z$. For practical purposes, the vector $V_m z$ can be taken instead.

It turns out that, apart from similarities (6), Krylov decompositions are invariant under the so-called translations, that is, the decomposition (5) has the same space as

$$AV_m = V_m(B_m+g b_m^*)+(v_{m+1}-V_m g)b_m^*. \tag{22}$$

This fact can be exploited to implement a harmonic Krylov–Schur method where the truncation of the decomposition keeps the relevant harmonic Ritz values. For additional details, see [35].

In terms of parallelization, this method is equivalent to the standard Krylov–Schur method, since the bulk of the computation is done with the Arnoldi method, and the changes affect only the operations concerning the projected matrix B_m (recall that this matrix is replicated in all processors). Since there is no need to solve a large linear system in parallel, this method is expected to have better efficiency than the spectral transformation variants.

4. Performance evaluation

In this section, we report on the performance of the different solution algorithms described in Section 3 when computing the rightmost eigenvalues of the GENE operator in different situations. The objective is to assess how the different parameters affect the computation time and conclude which method is the most appropriate for the particular application.

The analysis has been carried out with several test cases that can be considered real use scenarios. The test cases are defined as a set of parameters specified in the GENE input files. In this work, two test cases were used, that will be referred to as PAR1 and PAR2, see Tables 1 and 2. In the tables, the word SCAN indicates that the parameter takes values in a given range. In all tests involving spectral transformations, the tolerance requested to the linear solver is $0.1\cdot$ `tol`, that is, a slightly more stringent tolerance than for the eigensolver iteration.

For the tests, the following computing platform has been used: a Linux cluster based on Intel quad core technology, with 94 nodes, each of them with 8 cores and 16 GB of memory, and interconnected with a high-speed InfiniBand network. The

Table 1
Test case PAR1: very unstable kinetic ballooning mode with growth rate of 3.461 and frequency of 0.818 and another unstable mode (2.97 + 1.01*i*). Refer to Section 2.1 for a description of the parameters.

Direction	Parallel.	Resol.	Boxsize		SLEPc	
s	SCAN	2			`which`	sinvert
x	1	12			`tol`	10^{-5}
y	1	1	$k_{y,min}$	0.2	`nev`	1
z	SCAN	16			`shift`	SCAN
v	SCAN	48	l_v	3.0		
μ	SCAN	8	l_μ	9.0		

Other		Geometry		Param.	Ions	Electrons
β	0.1	$\hat{s}$	0.786	R/L_n	6	6
hyp_x	0.0	q_0	1.4	R/L_T	6	7.5
hyp_z	6.0	trpeps	0.53	m	1.0	0.000545
hyp_v	0.2			charge	1.0	-1.0
				T	1.0	1.0
				n	1.0	1.0

Table 2
Test case PAR2: for $\beta = 0$, the rightmost eigenvalues are $0.385 + 0.534i$ and $0.158 - 0.369i$; for $\beta = 0.016, 0.900 + 2.07i$ and $0.164 - 0.345i$. The largest magnitude eigenvalue is: $-3.73 - 957i$.

Direction	Parallel.	Resol.	Boxsize		SLEPc	
s	2	2			`which`	SCAN
x	1	7			`tol`	SCAN
y	1	1	$k_{y,min}$	0.2	`nev`	2
z	1	24			`shift`	SCAN
v	2	48	l_v	3.0	`ncv`	SCAN
μ	8	8	l_μ	9.0		

Other		Geometry		Param.	Ions	Electrons
β	0.0, 0.016	$\hat{s}$	0.8	R/L_n	2.22	2.22
hyp_x	0.2	q_0	1.4	R/L_T	6.89	6.89
hyp_z	8.0	trpeps	0.18	m	1.0	0.0005669
hyp_v	0.2			charge	1.0	-1.0
				T	1.0	1.0
				n	1.0	1.0

software installation includes MVAPICH2 1.0.1, GENE 1.2, and the development versions of PETSc and SLEPc. The latter is an evolution of SLEPc 2.3.3 that includes the functionality described in Section 3.3.

4.1. Evaluation of methods in terms of convergence and robustness

In this subsection, we carry out performance measurements with a fixed number of processors, in order to assess how the different methods behave relative to each other, and which is the influence of some adjustable parameters such as the shift. See Section 4.2 for an analysis of scalability for varying number of processors.

We start by analyzing the influence of SLEPc's `ncv` parameter in the context of spectral transformations. This parameter specifies the number of column vectors used by the eigensolver, that is, the dimension of the subspace that is built internally. It must be chosen in such a way that it is sufficiently large to have good convergence but it does not make it too expensive to build the subspace. Typically, with spectral transformations the dimension of the subspace need not be too large. Fig. 1 illustrates the variation of computing time when the value of `ncv` is changed. The plots seem to indicate that shift-and-invert is quite insensitive to changes in the dimension of the subspace, but in the case of Cayley a value larger than 30 may be preferable.

The parameter that impacts performance most in the case of spectral transformations is the shift, σ. As indicated in Section 3.2, the shift must be chosen to be relatively closer to the wanted eigenvalues than to the rest of the spectrum. Fig. 2 illustrates how eigenvalues are remapped for different values of the shift in the case of shift-and-invert. This figure corresponds to a test case of order 1024 so that computing the whole spectrum was feasible (in particular, 1 species with resolutions 2, 1, 8, 8, 8 for x, y, z, v, and μ, respectively). As can be seen from the plots, if the shift is very close to the rightmost eigenvalue, then the eigenvalues in that region of the spectrum become very well separated, whereas largest magnitude

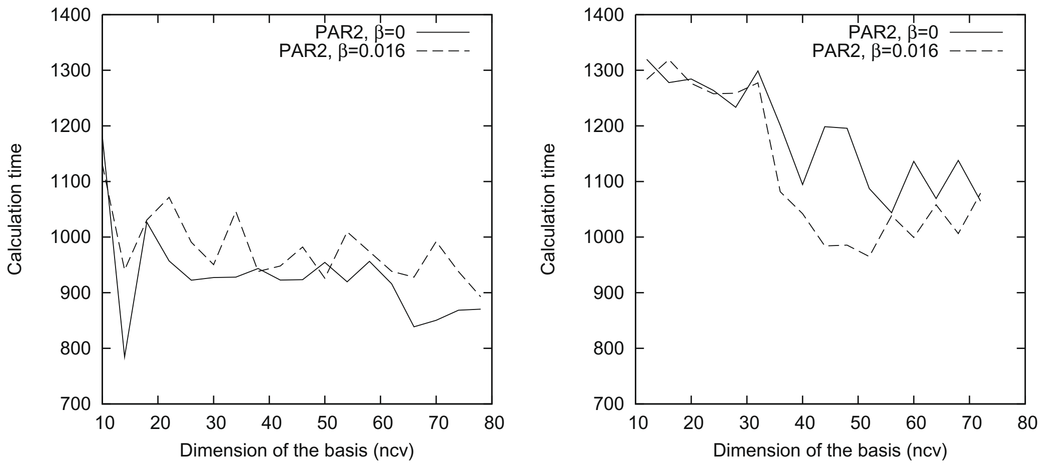

Fig. 1. Calculation time (in seconds) for PAR2 test case with different values of parameter `ncv` in the case of shift-and-invert (left) and Cayley transform (right).

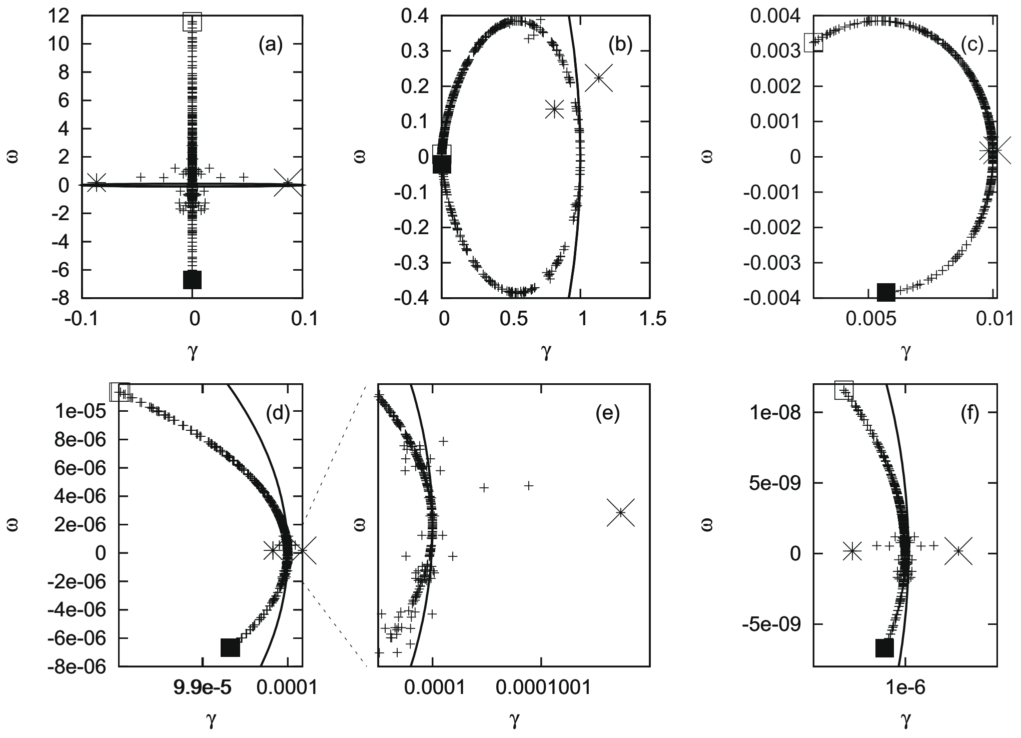

Fig. 2. Transformation of the spectrum with shift-and-invert: (a) is the original spectrum; (b), (c), (d) and (f) are transformed with $\sigma = 1, \ \sigma = 10, \ \sigma = 100$, and $\sigma = 1000$, respectively; (e) is a detail of the case $\sigma = 100$. The solid line represents equimodular sets. The four extreme eigenvalues (rightmost, leftmost, topmost, bottommost) are marked with a cross, an asterisk, an empty square, and a filled square, respectively.

eigenvalues tend to cluster around the origin. In contrast, if the shift gets farther away, then eigenvalues are kept relatively away from the origin but separation is increasingly worse. In all cases, in principle, rightmost eigenvalues become the ones with largest magnitude. However, if the unstable eigenvalues have relatively large frequencies (i.e. imaginary parts), then it may happen that a shift located in the real axis is closer to the stable eigenvalues than to the unstable ones. In such cases, a large enough shift should be chosen for the rightmost eigenvalues to be the largest magnitude ones in the transformed spectrum.

If we consider the strong influence that separation has on convergence, we can anticipate that the eigensolver will require much more iterations when the shift gets away from the wanted eigenvalues. This is confirmed by experiments, in which we have measured computing time (in parallel with 2, 2, and 8 processors for the s, v and μ directions, respectively) and number of eigensolver iterations for test case PAR1 with increasing values of σ, see Fig. 3. As expected, iterations grow linearly with the shift. In contrast, the computation time seems to decrease steadily for farther away shifts. The explanation is that, in the case that σ is very close to the wanted eigenvalues, the iterative linear solver takes many iterations to reach convergence (because some eigenvalues of the transformed operator are close to zero), whereas for larger σ's linear systems are solved quite easily. This may lead us to the conclusion that a large shift is computationally advantageous. However, we will soon show that this is misleading because the errors in the computed solutions must be taken into account.

In the case of Cayley, the eigenvalues are remapped in a rather different way, see Fig. 4. Note that no eigenvalues lie close to the origin, and this is in principle favorable for iterative linear solvers. An analogue of the study of Fig. 3 comparing the execution time and the number of iterations of the eigensolver is shown in Fig. 5 for the Cayley transform. In order to analyze the effect of both σ and τ, the plots show the results with respect to $s := \tau = n\sigma$ for $n = 1, \ n = 2$ and $n = 4$. According to the plots, the cases $n = 2$ and $n = 4$ seem to perform quite similarly, and with no significant difference with respect to the behaviour of shift-and-invert observed in Fig. 3. In the case of $n = 1$, the situation is rather equivalent, with the difference that it seems less robust for values of s larger than 200.

The first conclusion to be drawn is, that in this particular application, there seems to be no advantage in using Cayley transform instead of shift-and-invert, at least for the values of τ that were analyzed. The situation could be different for other choices of τ, or for cases in which the spectrum has a different structure or the shift σ lies closer to the wanted eigenvalues. In any case, the good news is that with SLEPc choosing one transformation or the other is extremely easy, just a matter of specifying it at the command line.

As stated before, choosing a value of σ that is far from the spectrum makes the computation faster. However, the computed solution is less accurate as illustrated in Fig. 6. If the user specifies a tolerance of 10^{-5}, then one should expect the

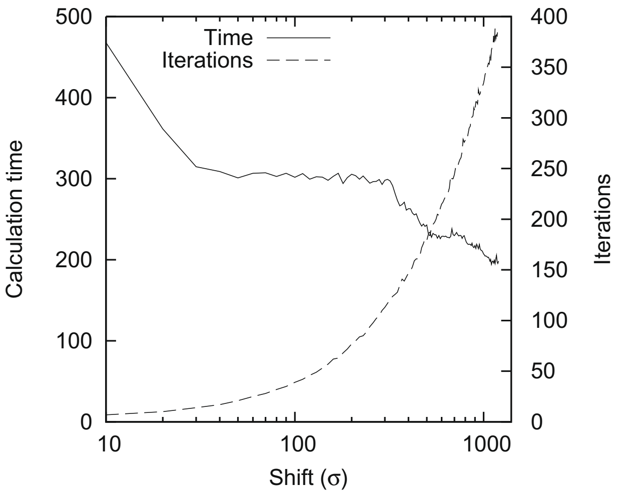

Fig. 3. Calculation time (in seconds) and number of iterations required by the eigensolver for different values of the shift σ in the case of shift-and-invert spectral transformation (with test case PAR1).

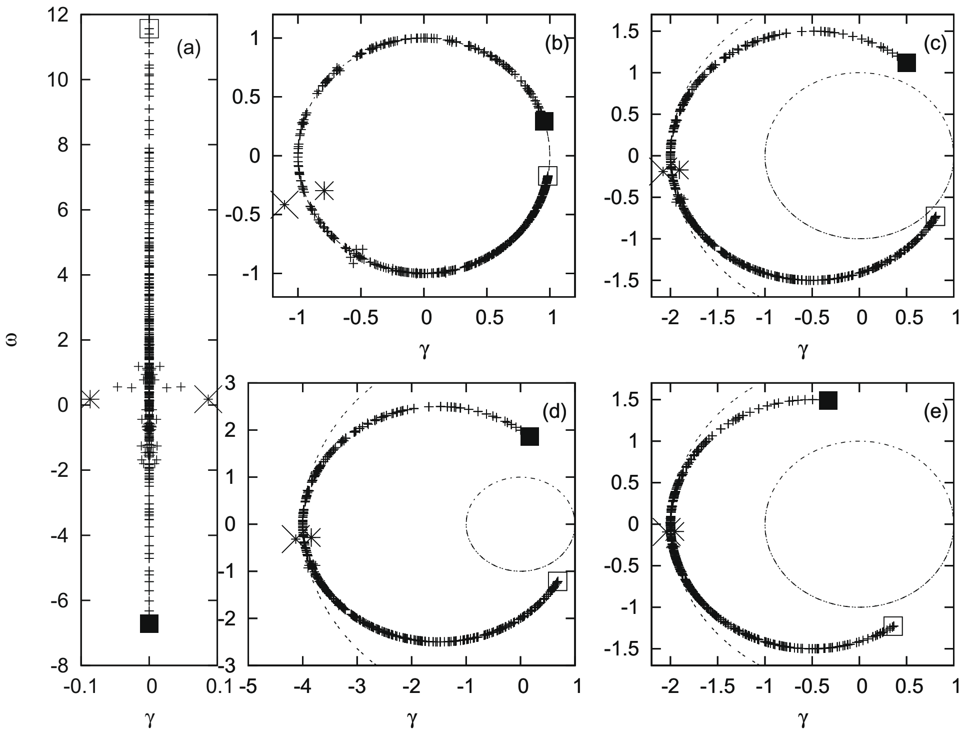

Fig. 4. Transformation of the spectrum with Cayley transform: (a) original spectrum; (b) with $\sigma = 1$ and $\tau = 1$; (c) with $\sigma = 3$ and $\tau = 6$; (d) with $\sigma = 3$ and $\tau = 12$; (e) with $\sigma = 6$ and $\tau = 12$. Dotted lines represents equimodular sets. The four extreme eigenvalues (rightmost, leftmost, topmost, bottommost) are marked with a cross, an asterisk, an empty square, and a filled square, respectively.

computed eigenpairs to have an associated (relative) residual norm of the same order of magnitude. However, Fig. 6 shows a growing residual norm for growing shift, for both shift-and-invert and Cayley. The growth is asymptotically linear, so roughly the accuracy of the computed solution loses as many decimal digits as the order of magnitude of σ. As already explained in Section 3.2, the reason for this loss of accuracy is that the implementation of the spectral transformation is done in such a way that the stopping criterion of the iterative eigensolver is checked in terms of the transformed eigenproblem, and not the original one. In other words, when a given eigenpair satisfies the criterion, it means that its associated residual norm is smaller than the tolerance, but with respect to T_{SI} or T_C, not with respect to A. A workaround for this would be to explicitly

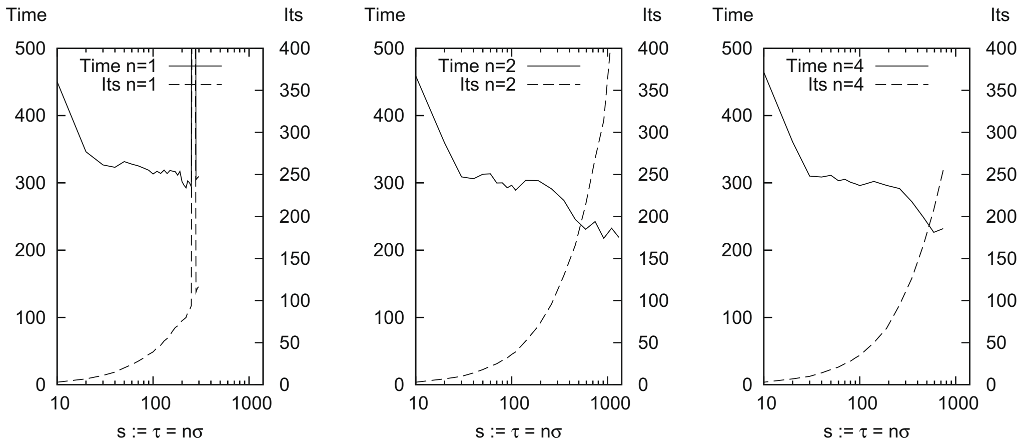

Fig. 5. Calculation time (in seconds) and number of iterations required by the eigensolver for different values of the parameter $s := \tau = n\sigma$ in the case of the Cayley transform (with test case PAR1).

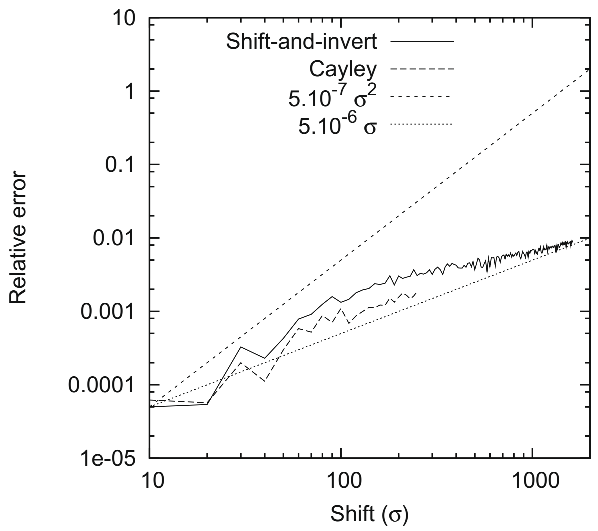

Fig. 6. Relative residual error for both shift-and-invert and Cayley transform for different values of the shift in PAR1 test case. In Cayley, the value $\tau = \sigma$ is used.

compute the residual norm for those potentially converged eigenpairs. Though this solution might seem too computationally expensive, in practice this computation would not take place too often. However, the problem is that this scheme would require to solve the linear systems associated to the spectral transformation much more accurately, otherwise the tolerance of the outer eigenvalue process would not be attained in most cases. All in all, for this particular application, spectral transformations are clearly outperformed by the harmonic projection strategy discussed below.

We now turn our attention to the performance of the linear solver that is used within the eigensolver for the spectral transformation. Recall that the problem matrix is not available explicitly, and this implies that these systems must be solved with an iterative method. Furthermore, it is not possible to use a conventional preconditioner for convergence enhancement. We consider either unpreconditioned Krylov solvers, or FGMRES preconditioned by an inner Krylov iteration, as discussed in the introduction.

Krylov-based iterative linear solvers without preconditioner perform reasonably well in this particular application. PETSc provides a number of iterative linear solvers, including GMRES, BiCGStab, CGS, and TFQMR (see [1] for an overview of iterative linear solvers). Since GMRES is the only solver that depends on a parameter (the maximum subspace dimension, also known as the restart), we first study the influence of this parameter in different situations. Fig. 7 shows how the total computing time varies when we allow GMRES to work with a bigger subspace. The conclusion is that the behaviour is good for small values of the restart parameter, and there is no point in using a larger value. The optimal value seems to be 12, for test case PAR2 and similarly for the other test case.

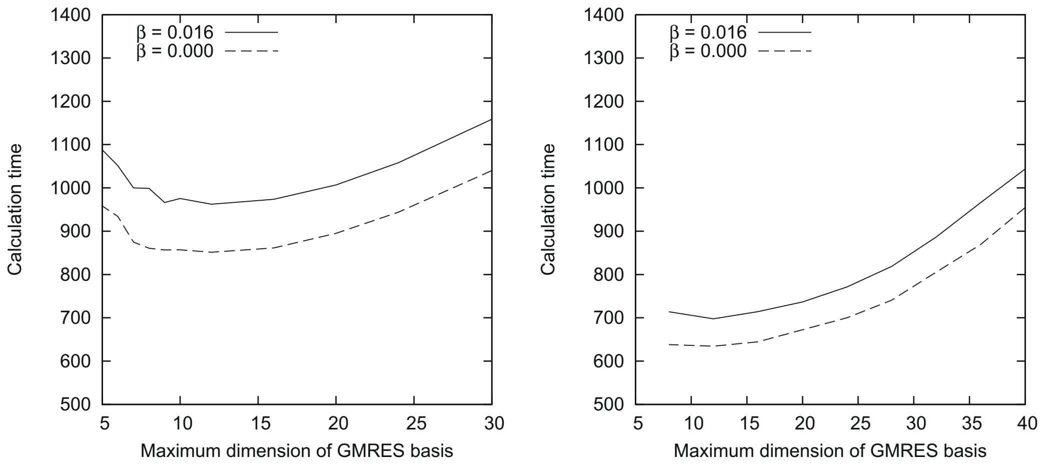

Fig. 7. Calculation time (in seconds) for PAR2 test case with different values of the restart parameter in GMRES, in the case of shift-and-invert (left) and Cayley transform (right).

Table 3 compares the different linear solvers in terms of how they affect the total computation time of the eigensolver, in the context of shift-and-invert. The differences are quite significant, sometimes of 40%. Roughly, we could conclude from the data that TFQMR is the best performing solver, which is more or less equivalent to GMRES with a restart parameter of 12. Table 4 yields a similar conclusion for the case of Cayley transform.

Also, we carried out some experiments with a flexible inner-outer iteration, particularly FGMRES where the preconditioner is a GMRES iteration with a variable tolerance for the stopping criterion. We found that the most favorable strategy for the tolerance in our application is to start with a tolerance of 0.9 at the beginning of the FGMRES cycle and then reduce it by a factor of 0.9 in each iteration. With these settings, we were able to reduce the total execution time in about 12% in the case of a restart value of 30 (the first entry in Table 3). Other cases reported even less benefit, so we can conclude that flexible inner–outer schemes are not especially useful in this particular setting.

We now focus on the analysis of the harmonic projection method described in Section 3.3. As in the case of the spectral transformation, we start by looking at the influence of SLEPc's `ncv` parameter (dimension of the subspace) in the overall computing time. The results are shown in Fig. 8, and seem to indicate that a very small basis is preferable, with about 10 or 12 vectors only. Therefore, for this particular application the harmonic projection technique does a very good job in retrieving the wanted eigenvalues, without the necessity of building a large Krylov subspace. Another conclusion that is readily recognizable when comparing Fig. 8 with Fig. 1 is that the harmonic projection option is much faster than the spectral transformation, with a gain of one order of magnitude at least. This remarkable benefit has been observed in other test cases as well, and harmonic projection is always at least five times faster. Also, the computed solution is much more reliable as discussed below.

Table 3
Calculation times (in seconds) for different linear solvers in the case of PAR2 with shift-and-invert for shifts of 10 and 100, and cases $\beta = 0$ and $\beta = 0.016$.

Method	$t_{10,0.0}$	$t_{10,0.016}$	$t_{100,0.0}$	$t_{100,0.016}$
GMRES30	1039	1158	1062	846
GMRES12	850	964	881	706
BiCGStab	1181	1280	1070	825
CGS	904	966	968	770
TFQMR	888	945	949	667

Table 4
Calculation times (in seconds) for different linear solvers in the case of PAR2 with Cayley transform for shifts of 10 and 100, and cases $\beta = 0$ and $\beta = 0.016$.

Method	$t_{10,0.0}$	$t_{10,0.016}$	$t_{100,0.0}$	$t_{100,0.016}$
GMRES30	780	774	769	857
GMRES12	637	627	634	698
BiCGStab	892	830	875	936
CGS	667	677	745	798
TFQMR	638	653	715	782

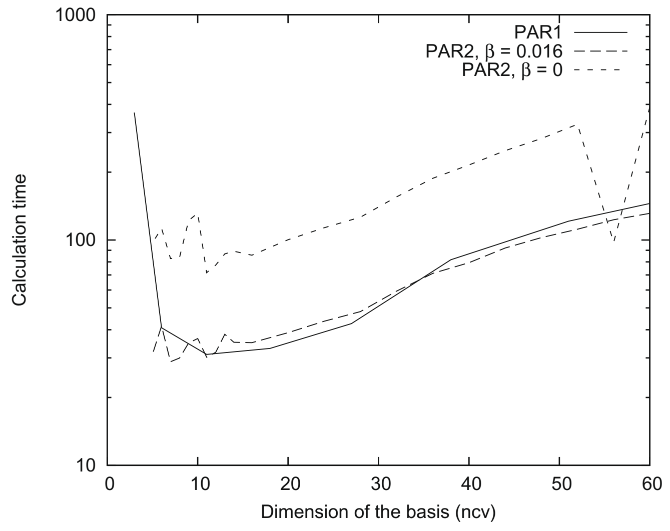

Fig. 8. Calculation time (in seconds) for different values of parameter `ncv` in the case of harmonic projection.

As in the previous discussion of errors, we are going to consider the relative residual norm of the computed eigenpairs. As we will shortly see, in the case of harmonic projection, errors will not depend on the value of the target, as opposed to the case of spectral transformation. With spectral transformations, we have seen that one can get more accurate results by moving the shift σ closer to the wanted eigenvalues, or alternatively by reducing the tolerance value. In both cases, this has an accompanying increase in computing time. This relation between accuracy and computing time is illustrated in Fig. 9. The solid line in this figure represents the behaviour of plain Krylov–Schur, that is, when we apply the method without spectral transformation or harmonic projection, only with the method's own filtering mechanism instructed to compute "largest real" (i.e. rightmost) eigenvalues. The computed relative error for this option is roughly of the same order of magnitude of the tolerance (the figure shows tolerances ranging from 10^{-1} to 10^{-6}), where the computational effort is of course larger for smaller tolerance. This method takes a lot of iterations to converge, but the cost of each iteration is relatively cheap. In contrast, spectral transformations converge with much fewer iterations, but the cost per iteration is much higher, and on top of that there is the problem of obtaining less accuracy than demanded. If we analyze the two spectral transformations for different values of the shift and different tolerances (see Fig. 9) then we get the surprising result that only a few cases are able to beat the performance of the "largest-real" option, that is, get more accuracy with less computing time. In contrast, when running the Krylov–Schur method with harmonic projection, the calculation time and the relative error are remarkably similar

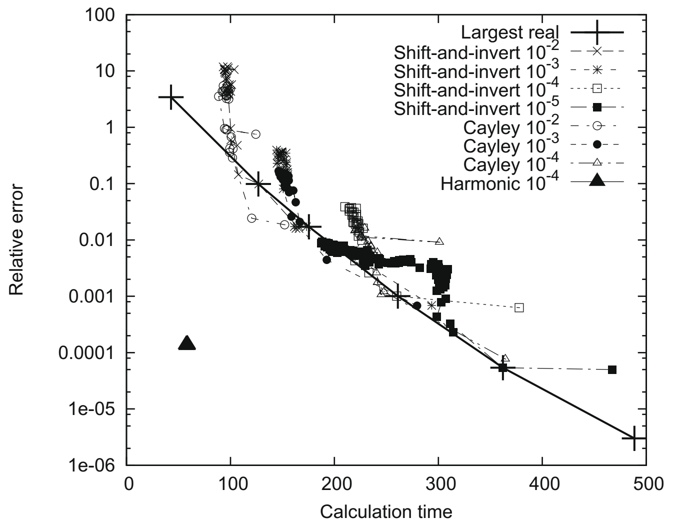

Fig. 9. Comparison of the relative error of the computed eigenpairs with respect to the corresponding calculation time (in seconds) for different solution alternatives and different tolerances, in test case PAR1.

for different values of the target, and in all cases the eigensolver attains the requested precision with much less computational effort compared to the other alternatives (black triangle in the figure). Although not shown, similar results have been obtained for the other test case, with the exception that plain Krylov–Schur with the largest-real option does not always converge to the rightmost eigenvalues, thus being less reliable in general than the rest of the methods.

4.2. Parallel performance

The goal of this subsection is to show that the computation scales well to hundreds of processors. The analysis is not straightforward because in GENE the parallelization is carried out with respect to four of the six independent variables, namely $s,\ z,\ v$, and μ (parallelization is not possible in the x-direction, whereas the y modes are linearly decoupled so this parallelization direction is not necessary). For the scalability analysis, we have conducted several timing experiments with test case PAR1. Note that the number of processors used for parallelizing s is limited to 2 in this test case, since only two species (ions and electrons) are defined.

We start by analyzing the strong scaling of the computation, that is, when the problem size is fixed as the number of processors varies. The fixed problem size is that given by the resolution values in Table 1. We will denote by $p_s,\ p_z,\ p_v$, and p_μ the number of processors used for parallelization in the respective direction. Also, we will represent the number of processors as a tuple, $p = (p_s, p_z, p_v, p_\mu)$. For strong scaling, we measure the speed-up in each direction, defined as

$$S_v(p) = \frac{T(p_s, p_z, 1, p_\mu)}{T(p_s, p_z, p_v, p_\mu)} \tag{23}$$

for the v direction, and analogously for the other directions, where $T(\cdot,\cdot,\cdot,\cdot)$ represents the computing time for the given number of processors. The ideal speed-up is considered to be $S_v(p) = p_v$.

Table 5 shows the computing times obtained in the case of shift-and-invert spectral transformation when we vary the number of processors in each direction. The tests were done with $\sigma = 120$, and tolerances equal to 10^{-5} for the eigensolver and 10^{-6} for the linear solver. Note that the times differ significantly from the best to the worst case. Fig. 10 shows graphically the speed-up obtained in each of the $z,\ v$, and μ directions, with a fixed number of processors for the other directions. In all cases, the worst case performs rather poorly, but in contrast the speed-up of the best case is quite good, with super-linear speed-up in some cases.

We present an analogous comparison for the harmonic projection method in Table 6 and Fig. 11. These data show that the harmonic solver is not only much faster, but also its scalability is much better, at least for the best cases. The super-linear speed-up could be attributed to a more favorable use of the memory hierarchy.

We finish this section with a discussion on weak scaling, that is, when the problem size is increased in accordance with the number of processors. As in the case of strong scaling, weak scaling is studied for the different variables independently, in particular for the $z,\ v$ and μ directions in PAR1. For this analysis, we assume that the cost of applying the operator grows linearly with respect to each of these variables, and consider the scaled speed-up, defined as the speed-up obtained when the problem size is increased linearly with the number of processors. For example, consider the v-direction, and suppose we are solving a problem of size n_v (resolution in the v-direction) with one processor, and the computation takes $T(p_s, p_z, 1, p_\mu)$ seconds. Then for the scaled speed-up we solve a bigger problem, with resolution equal to $n_v \cdot p_v$ with p_v processors, in $T'(p_s, p_z, p_v, p_\mu)$ seconds. Note that ideally both times should be equal, by the assumption that the cost is linear with respect to the problem size (except for the varying number of iterations discussed below). The scaled speed-up is

Table 5
Calculation times (in seconds) employed by the eigensolver in computing two eigenpairs of test case PAR1 with shift-and-invert, for different combinations of processor numbers in the different directions. For each group, the combinations that lead to smallest times are listed on the left (best) and the ones that lead to largest times are listed on the right (worst).

p				T	p				T	p				T	p				T
s	z	v	μ	best	s	z	v	μ	worst	s	z	v	μ	best	s	z	v	μ	worst
Four processors										Thirty-two processors									
1	1	1	4	2027	1	2	2	1	2875	2	1	2	8	302	1	8	4	1	671
1	2	1	2	2197	1	1	2	2	2723	2	1	4	4	307	1	8	2	2	534
2	2	1	1	2226	2	1	1	2	2455	1	1	4	8	307	2	8	2	1	531
Eight processors										Sixty-four processors									
1	1	1	8	1667	1	8	1	1	2352	2	2	4	4	216	1	8	8	1	466
2	1	1	4	1673	1	4	2	1	2124	2	2	2	8	217	2	8	4	1	360
1	1	2	4	1713	2	4	1	1	2037	1	2	4	8	220	1	8	4	2	359
Sixteen processors										One hundred and twenty eight processors									
2	1	1	8	708	1	8	2	1	1219	2	2	4	8	207	1	8	16	1	469
1	1	2	8	730	2	8	1	1	1053	1	2	8	8	209	2	8	8	1	356
2	1	2	4	746	1	8	1	2	1044	2	4	2	8	210	1	8	8	2	348

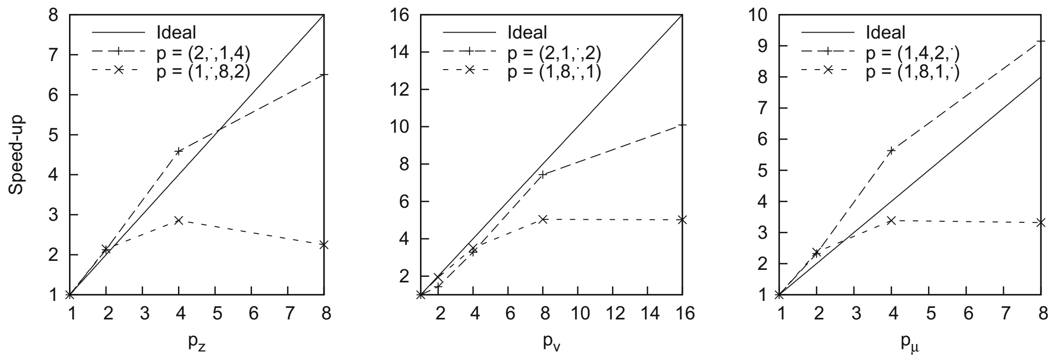

Fig. 10. Strong scaling results (with fixed problem size) obtained when solving PAR1 in shift-and-invert mode with varying number of processors in the z, v and μ directions. The plots show the best and the worst speed-up in each case. Ideal speed-up is shown as a solid line.

Table 6
Calculation times (in seconds) employed by the eigensolver in computing two eigenpairs of test case PAR1 with harmonic projection, for different combinations of processor numbers in the different directions. For each group, the combinations that lead to smallest times are listed on the left (best) and the ones that lead to largest times are listed on the right (worst).

p				*T*	*p*				*T*	*p*				*T*	*p*				*T*
s	*z*	*v*	*μ*	best	*s*	*z*	*v*	*μ*	worst	*s*	*z*	*v*	*μ*	best	*s*	*z*	*v*	*μ*	worst
One processor										Sixteen processors									
1	1	1	1	1250	1	1	1	1	1250	2	1	1	8	118	1	8	2	1	190
										1	1	2	8	122	2	8	1	1	170
										2	1	2	4	125	1	8	1	2	169
Two processors										Thirty-two processors									
2	1	1	1	610	1	2	1	1	820	2	1	2	8	40	1	8	4	1	87
1	1	1	2	615	1	2	1	1	650	1	1	4	8	43	2	8	2	1	71
1	2	1	1	650	1	1	1	2	616	2	1	4	4	44	1	8	2	2	70
Four processors										Sixty-four processors									
2	1	1	2	398	2	2	1	1	493	2	2	2	8	30	1	8	8	1	62
1	1	4	1	422	1	2	1	2	487	1	2	4	8	31	2	8	4	1	49
1	2	2	1	445	2	1	2	1	478	2	2	4	4	31	1	8	4	2	49
Eight processors										One hundred and twenty-eight processors									
1	1	1	8	322	1	8	1	1	403	2	2	4	8	27	1	8	8	2	45
2	1	1	4	322	1	4	2	1	370	1	2	8	8	27	2	8	8	1	45
1	1	2	4	326	2	4	1	1	363	2	4	2	8	27	1	8	4	4	38

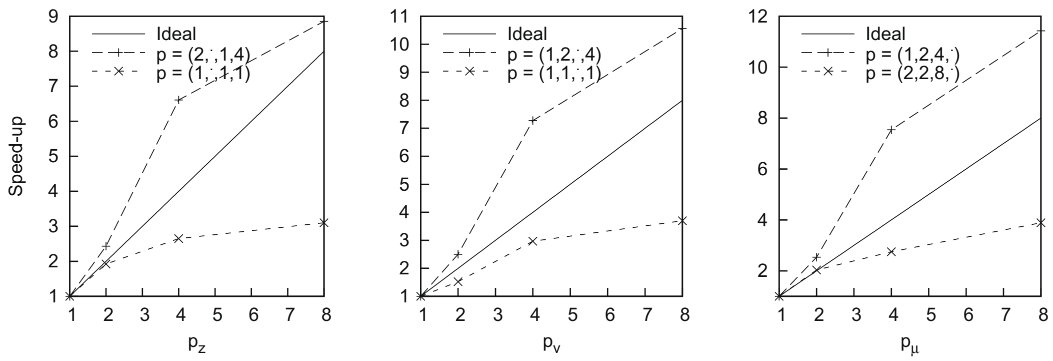

Fig. 11. Strong scaling results (with fixed problem size) obtained when solving PAR1 in harmonic projection mode with varying number of processors in the z, v and μ directions. The plots show the best and the worst speed-up in each case. Ideal speed-up is shown as a solid line.

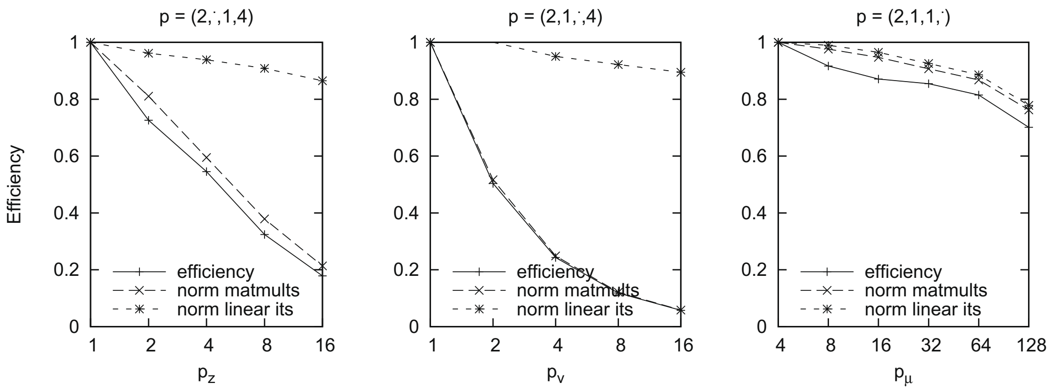

Fig. 12. Weak scaling results (with variable problem size) obtained when solving PAR1 in shift-and-invert mode with varying number of processors in the z, v and μ directions.

$$S'_v(p) = \frac{T'(p_s, p_z, 1, p_\mu)}{T'(p_s, p_z, p_v, p_\mu)} = \frac{p_v \cdot T(p_s, p_z, 1, p_\mu)}{T'(p_s, p_z, p_v, p_\mu)}. \quad (24)$$

Ideally, $S'_v(p) = p_v$ since the computing time remains constant. For convenience, we will use the scaled efficiency,

$$E'_v(p) = \frac{S'_v(p)}{p_v} = \frac{T(p_s, p_z, 1, p_\mu)}{T'(p_s, p_z, p_v, p_\mu)}, \quad (25)$$

that is, the ratio of execution times, as in the case of standard speed-up. In summary, an efficiency close to 1 indicates good weak scalability.

Fig. 12 shows the scaled efficiency for the z, v and μ directions. In the case of the μ-direction, the efficiency is quite good, always above 70% up to 128 processors (note that in this plot the reference case is 4 processors instead of 1). However, in the z and v directions the efficiency decays very quickly, even for a moderate number of processors. The explanation for this is that the characteristics of the problem change with the problem size, in particular the conditioning and spectral properties, and consequently the solution method will require more iterations, typically. This is true for the eigensolver, but also for the iterative linear solver associated to the spectral transformation. In order to eliminate this effect from the analysis, in Fig. 12 we also show the efficiency normalized with respect to the number of matrix–vector products required by the eigensolver, and with respect to the iterations required by the linear solver (that is, we consider execution times divided by these numbers). We can see from the plots that normalizing with respect to matrix–vector products does not change the picture significantly, but in contrast the normalization with respect to linear iterations recovers a good efficiency, as good as in the case of the μ-direction. The interpretation for this must be that the linear systems associated to the spectral transformation become increasingly difficult to solve, requiring more iterations when the problem size grows. So we can conclude that the spectral transformation strategy does not scale with respect to the z and v directions, due to the linear solver.

In the harmonic projection technique there is no resolution of linear systems, so one could expect good scalability in all cases. However, Fig. 13 shows a similar decay for the z and v directions. In this case, the interpretation is that the spectrum

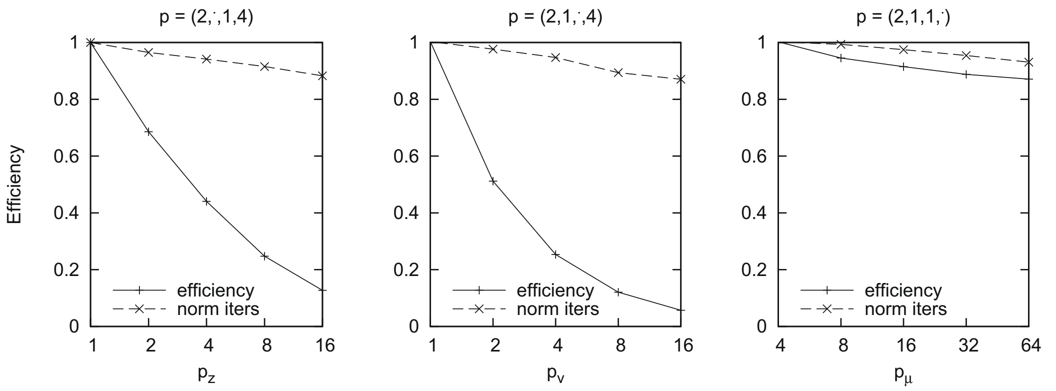

Fig. 13. Weak scaling results (with variable problem size) obtained when solving PAR1 in harmonic projection mode with varying number of processors in the z, v and μ directions.

becomes more anisotropic (the dominant eigenvalue grows in magnitude) with a larger problem size, and the harmonic extraction technique needs more iterations to approximate the wanted eigenvalues. The plot also shows the efficiency normalized with respect to the number of iterations of the eigensolver, recovering an efficiency of about 90%.

5. Conclusions

We have addressed the problem of computing the rightmost eigenvalues and their associated eigenvectors of the linearized gyrokinetic equations. These computations are very important for the study of turbulence phenomena in confined plasmas. The determination of these unstable eigenmodes can provide valuable insight into the properties of plasma turbulence.

Rightmost eigenvalues can be computed with a Krylov eigensolver combined with a spectral transformation technique such as shift-and-invert or Cayley transform. In our particular application, the unavailability of the operator matrix in explicit form prevents from using a direct linear solver, so the only possibility is to do an inexact spectral transformation with an unpreconditioned iterative linear solver or with a flexible inner–outer Krylov iteration such as FGMRES-GMRES. We have analyzed the behaviour of such methods in terms of convergence and accuracy of computed results, showing that in this application the Cayley transform does not seem to provide any advantage over shift-and-invert. Both methods have a very high cost due to slow convergence of the iterative linear solver, and provide large residual errors for some choices of the parameters.

The alternative to inexact spectral transformation is harmonic projection, which combined with the effective restarting mechanism of the Krylov–Schur eigensolver is able to drive convergence toward the wanted part of the spectrum. Apart from being much faster (a factor of 5 at least), this approach has the advantage of requiring less parameters to be set by the user (only the target value, compared to the shift, anti-shift, and tolerance of the inner iteration required by the Cayley approach). Also, the harmonic Krylov–Schur method is much more robust, because the requested accuracy is always attained irrespective of the parameters specified by the user.

Regarding parallel efficiency, all solution strategies have shown reasonably good speedups for different test cases, and with respect to different independent variables. Parallel performance of the overall computation is mainly inherited from the parallelization of the matrix–vector product, since no preconditioning is being used and the eigensolver does not introduce much parallel overhead apart from several synchronization points. In this sense, the spectral transformation approach is expected to have a slightly worse parallel efficiency because the number of synchronization points is larger, due to the inner–outer iteration scheme. This conclusion is supported by numerical experiments. In terms of scalability to large number of processors, both approaches behave well, although they are quite sensitive to changes in the spectrum that typically occur when the problem size is changed.

Since in SLEPc it is trivial to switch from one eigensolver to another one, without changing the application code, it would be interesting to analyze the behaviour of preconditioned eigensolvers such as Jacobi–Davidson, when it is available in SLEPc. However, this would require to address the question of how to build a good preconditioner for the linear gyrokinetic operator. Another topic for future research could be the use of eigensolvers with refined Ritz extraction [36], although preliminary experiences seem to indicate that poor improvements are obtained in this particular application.

Acknowledgement

The first author was supported in part by the Spanish Ministry of Science and Innovation under Grant TIN2009-07519.

References

[1] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.
[2] V. Simoncini, D.B. Szyld, Flexible inner–outer Krylov subspace methods, SIAM Journal on Numerical Analysis 40 (6) (2003) 2219–2239.
[3] F. Jenko, B. Dorland, M. Kotschenreuther, B. Rogers, Electron temperature gradient driven turbulence, Physics of Plasmas 7 (5) (2000) 1904–1910.
[4] T. Dannert, F. Jenko, Gyrokinetic simulation of collisionless trapped-electron mode turbulence, Physics of Plasmas 12 (7) (2005) 072309.
[5] V. Hernandez, J.E. Roman, V. Vidal, SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software 31 (3) (2005) 351–362.
[6] F. Merz, Gyrokinetic simulation of multimode plasma turbulence, Ph.D. Thesis, Universität Münster, 2008.
[7] M. Kammerer, F. Merz, F. Jenko, Exceptional points in linear gyrokinetics, Physics of Plasmas 15 (5) (2008) 052102.
[8] V. Hernandez, J.E. Roman, A. Tomas, V. Vidal, A survey of software for sparse eigenvalue problems, Technical Report STR-6, Universidad Politécnica de Valencia, available at <http://www.grycap.upv.es/slepc>, 2007.
[9] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK Users', Guide Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1998.
[10] C.G. Baker, U.L. Hetmaniuk, R.B. Lehoucq, H.K. Thornquist, Anasazi software for the numerical solution of large-scale eigenvalue problems, ACM Transactions on Mathematical Software 36 (3) (2009) 13:1–13:23.
[11] S. Balay, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M. Knepley, L.C. McInnes, B.F. Smith, H. Zhang, PETSc users manual, Technical Report ANL-95/11 – Revision 2.3.3, Argonne National Laboratory, 2007.
[12] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.
[13] G.W. Stewart, Matrix Algorithms, Eigensystems, vol. 2, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.
[14] G.W. Stewart, A Krylov–Schur algorithm for large eigenproblems, SIAM Journal on Matrix Analysis and Applications 23 (3) (2001) 601–614.
[15] V. Hernandez, J.E. Roman, A. Tomas, Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement, Parallel Computing 33 (7–8) (2007) 521–540.

[16] T. Ericsson, A. Ruhe, The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems, Mathematics of Computation 35 (152) (1980) 1251–1268.
[17] D.S. Scott, The advantages of inverted operators in Rayleigh–Ritz approximations, SIAM Journal on Scientific and Statistical Computing 3 (1) (1982) 68–75.
[18] B. Nour-Omid, B.N. Parlett, T. Ericsson, P.S. Jensen, How to implement the spectral transformation, Mathematics of Computation 48 (178) (1987) 663–673.
[19] R.G. Grimes, J.G. Lewis, H.D. Simon, A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems, SIAM Journal on Matrix Analysis and Applications 15 (1) (1994) 228–272.
[20] H. Zhang, B. Smith, M. Sternberg, P. Zapol, SIPs: shift-and-invert parallel spectral transformations, ACM Transactions on Mathematical Software 33 (2) (2007) 1–19.
[21] B.N. Parlett, Y. Saad, Complex shift and invert strategies for real matrices, Linear Algebra and its Applications 88/89 (1987) 575–595.
[22] M.N. Kooper, H.A. van der Vorst, S. Poedts, J.P. Goedbloed, Application of the implicitly updated Arnoldi method with a complex shift-and-invert strategy in MHD, Journal of Computational Physics 118 (2) (1995) 320–328.
[23] U.L. Hetmaniuk, R.B. Lehoucq, Uniform accuracy of eigenpairs from a shift-invert Lanczos method, SIAM Journal on Matrix Analysis and Applications 28 (4) (2006) 927–948.
[24] M.A. Freitag, A. Spence, Convergence theory for inexact inverse iteration applied to the generalised nonsymmetric eigenproblem, Electronic Transactions on Numerical Analysis 28 (2007) 40–64.
[25] K. Meerbergen, A. Spence, D. Roose, Shift–invert and Cayley transforms for detection of rightmost eigenvalues of nonsymmetric matrices, BIT Numerical Mathematics 34 (3) (1994) 409–423.
[26] K. Meerbergen, D. Roose, Matrix transformations for computing rightmost eigenvalues of large sparse non-symmetric eigenvalue problems, IMA Journal of Numerical Analysis 16 (3) (1996) 297–346.
[27] K. Meerbergen, D. Roose, The restarted Arnoldi method applied to iterative linear system solvers for the computation of rightmost eigenvalues, SIAM Journal on Matrix Analysis and Applications 18 (1) (1997) 1–20.
[28] R.B. Lehoucq, A.G. Salinger, Large-scale eigenvalue calculations for stability analysis of steady flows on massively parallel computers, International Journal for Numerical Methods in Fluids 36 (2001) 309–327.
[29] G.L.G. Sleijpen, H.A.V. der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems, SIAM Review 42 (2) (2000) 267–293.
[30] R.B. Morgan, Preconditioning eigenvalues and some comparison of solvers, Journal of Computational and Applied Mathematics 123 (1–2) (2000) 101–115 (Numerical analysis 2000, Vol. III. Linear algebra).
[31] R.B. Morgan, Computing interior eigenvalues of large matrices, Linear Algebra and its Applications 154–156 (1991) 289–309.
[32] C.C. Paige, B.N. Parlett, H.A. van der Vorst, Approximate solutions and eigenvalue bounds from Krylov subspaces, Numerical Linear Algebra with Applications 2 (2) (1995) 115–133.
[33] R.B. Morgan, M. Zeng, Harmonic projection methods for large non-symmetric eigenvalue problems, Numerical Linear Algebra with Applications 5 (1) (1998) 33–55.
[34] R.B. Morgan, M. Zeng, A harmonic restarted Arnoldi algorithm for calculating eigenvalues and determining multiplicity, Linear Algebra and its Applications 415 (1) (2006) 96–113.
[35] J.E. Roman, Practical implementation of harmonic Krylov–Schur, Technical Report STR-9, Universidad Politécnica de Valencia, available at <http://www.grycap.upv.es/slepc>, 2009.
[36] Z. Jia, Refined iterative algorithms based on Arnoldi's process for large unsymmetric eigenproblems, Linear Algebra and its Applications 259 (1–3) (1997) 1–23.