Parallel Computing 40 (2014) 195–212

Contents lists available at ScienceDirect

Parallel Computing

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

Parallel eigenvalue calculation based on multiple shift–invert Lanczos and contour integral based spectral projection method

Hasan Metin Aktulga [a,*], Lin Lin [a], Christopher Haine [b], Esmond G. Ng [a], Chao Yang [a]

[a] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, United States
[b] Versailles Saint-Quentin-en-Yvelines University, 55 Avenue de Paris, 78000 Versailles, France

ARTICLE INFO

Article history:
Available online 21 March 2014

Keywords:
Parallel eigenvalue computations
Spectral transformation
Multiple shift–invert Lanczos
Contour integral based spectral projection method
Strong scaling
Weak scaling

ABSTRACT

We discuss the possibility of using multiple shift–invert Lanczos and contour integral based spectral projection method to compute a relatively large number of eigenvalues of a large sparse and symmetric matrix on distributed memory parallel computers. The key to achieving high parallel efficiency in this type of computation is to divide the spectrum into several intervals in a way that leads to optimal use of computational resources. We discuss strategies for dividing the spectrum. Our strategies make use of an eigenvalue distribution profile that can be estimated through inertial counts and cubic spline fitting. Parallel sparse direct methods are used in both approaches. We use a simple cost model that describes the cost of computing k eigenvalues within a single interval in terms of the asymptotic cost of sparse matrix factorization and triangular substitutions. Several computational experiments are performed to demonstrate the effect of different spectrum division strategies on the overall performance of both multiple shift–invert Lanczos and the contour integral based method. We also show the parallel scalability of both approaches in the strong and weak scaling sense. In addition, we compare the performance of multiple shift–invert Lanczos and the contour integral based spectral projection method on a set of problems from density functional theory (DFT).

1. Introduction

In a number of applications, we are interested in computing a small percentage of the eigenvalues of a large sparse matrix A or matrix pencil (A, B) [2,25]. Often these are eigenvalues at the low end of the spectrum.

When the dimension of the matrix n is above 10^6, for example, even one percent of n amounts to more than 10,000 eigenvalues. When that many eigenpairs are needed, many of the existing linear algebra libraries such as PARPACK [20], Anasazi [3], BLOPEX [12], PRIMME [31], for solving large-scale eigenvalue problems, which are mostly based on extracting approximate eigenpairs from a single subspace (e.g, a k-dimensional Krylov subspace $\mathcal{K}(A, v_0) = \{v_0, Av_0, \ldots, A^{k-1}v_0\}$ associated with some starting vector v_0) often do not provide an efficient way to solve the problem. This is true even when a vast amount of computational resource is available. Three major sources of the problems with these solvers are:

1. The amount of parallelism that can be exploited in a standard Krylov subspace method such as the Lanczos algorithm is limited. The main computational tasks involved in such an algorithm include matrix vector multiplications, basis

* Corresponding author. Tel.: +1 510 486 4228.
E-mail address: HMAktulga@lbl.gov (H.M. Aktulga).

http://dx.doi.org/10.1016/j.parco.2014.03.002
0167-8191/© 2014 Elsevier B.V. All rights reserved.

orthogonalization, and the Rayleigh–Ritz procedure for extracting desired eigenvalues from a subspace. For a general sparse matrix, it may be difficult to speed up the sparse matrix operations (such as sparse matrix vector multiplication or sparse matrix factorization) on more than hundreds of processors, although this may be possible for some applications [5,6].

2. As the number of required eigenpairs increases, the dimension of the subspace required to extract approximate spectral information can become quite large. Hence the cost for constructing an orthonormal basis for such a subspace may grow even faster than that associated with performing sparse matrix vector multiplications.
3. The projected eigenvalue problem is often replicated on each processor and solved sequentially in existing solvers. Solving the projected problem can become a bottleneck when the number of desired eigenvalues is over a few thousands. Although this part of the computational can be parallelized through the use of ScaLAPACK[4], it often requires a nontrivial change to the data structure that becomes incompatible with user defined sparse matrix operations. When the computation is carried out on a large number of processors, a large amount of communication overhead is expected. This overhead may degrade the parallel scalability of the computation.

New strategies and software tools must be developed to handle this type of calculations on the emerging distributed multi/many-core computing platforms. These strategies may depend on a number of factors such as the sparsity of the matrix, the distribution of the eigenvalues, and also the type of computational resources available to the user.

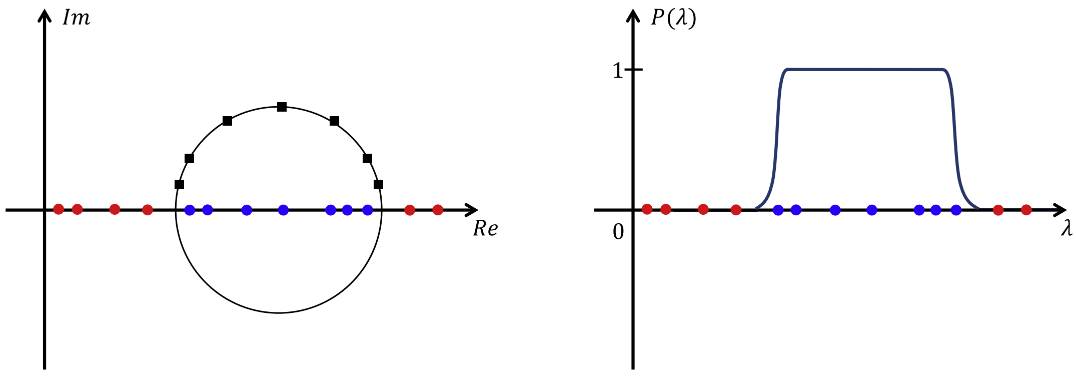
One way to reduce the cost of the Rayleigh–Ritz calculation based on the trace-penalty minimization method is described in [34]. In this paper, we examine a different strategy and provide some computational examples that illustrate the type of issues that we must consider and how they may be resolved. The general idea we pursue is to divide the desired part of the spectrum into several subintervals and to compute eigenvalues within each interval in parallel. By adding a coarse grained level of parallelism at the top, this approach obviously increases the total amount of concurrency in computation. By limiting the size of each interval, it also mitigates the second and the third issues that currently prevent some existing eigensolvers from achieving high efficiency. We would like to point out that this is an important aspect of the approach we discuss in this paper. Without putting a constraint on the size of each interval, the cost of solving the projected problem and performing orthogonalization may not be ignored, and the optimization of resource allocation and spectrum division then becomes more difficult.

We will consider two methods for computing eigenvalues within a prescribed interval:

1. Shift–invert Lanczos (SIL) with implicit restarts [15]
2. Subspace iteration applied to a projection operator constructed from an approximation to the following contour integral

$$P = \frac{1}{2\pi i}\oint_{\Gamma}(A - zI)^{-1}dz, \tag{1}$$

where Γ is a contour on the complex plane that encloses only the eigenvalues of interest [7]. We will refer to this approach as a contour integral based spectral projection method. Fig. 1(a) shows a typical contour Γ that encloses a number of eigenvalues marked by solid circles. The contour is numerically discretized into several quadrature points. Fig. 1(b) shows the



(a) A typical contour (big black circle) encircling a number of eigenvalues of interest (blue solid circles), and the contour is numerically discretized into several quadrature points (black squares). Due to symmetry only those quadrature points on the upper half plane are used.

(b) The value of the discretized projection operator along the axis of the spectrum.

Fig. 1. Sketch of the contour and the associated projection operator used by the spectral projection method. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

value of the discretized projection operator associated with such a contour, which maps eigenvalues within the contour to 1 and eigenvalues outside of the contour to 0, with certain amount of smearing near the boundary of the circle due to finite discretization. This approach has been discussed in [26,27,22,35,17]. In particular, both the FEAST method [22] and the Sakurai–Sugiura (SS) projection [26] method fall under this category.

We will refer to the combination of using spectrum division and SIL as *multiple shift–invert Lanczos* (MSIL) in subsequent discussions. The use of spectrum division and subspace iterations on spectral projectors constructed from the contour integral formulation will be referred to as *multiple contour integral based spectral projection method* (MCISPM). Both algorithms require solving a number of linear systems of equations of the form $(A - zI)x = b$, where z is either a complex or a real shift. Although these equations can also be solved by preconditioned iterative methods such as MINRES[21], GMRES[24], BiCGSTAB [33] and QMR [9], the operator $A - zI$ is in general not positive definite, even when z is chosen to be on the real axis as in MSIL. As a result it can be difficult to keep the number of iterations under control using iterative methods. In this paper, we will focus on direct methods that perform sparse LDL^T factorizations of $A - zI$ and sparse back substitutions using L and D factors. Alternative ways to implement spectrum slicing include using a polynomial filtered subspace iteration [28] and the Jacobi–Davidson algorithm [31] to compute eigenvalues within each slice.

Although spectrum division is conceptually simple, implementing this strategy efficiently on a large-scale parallel computer is not straightforward. The key to achieving high efficiency is to develop a way to divide the spectrum so that computational resources can be utilized in an optimal way (i.e., to maximize the overall throughput of the computation.) An optimal spectrum division scheme should place an appropriate number of eigenvalues in each interval to keep the cost per interval relatively low, while making sure the overall computation is load balanced and can be completed quickly. Developing such a scheme requires a careful analysis of the computational cost associated with each interval, which depends on the cost for performing an LDL^T factorization and a number of triangular substitutions as well as the number of eigenvalues and the convergence rate of these eigenvalues by using either SIL or subspace iteration.

When a large amount of computational resource is available, the optimal strategy may be different from one that is designed for a computational environment with a small limited amount of computing power. In both cases, one has to decide how much computational resource one should allocate to compute eigenvalues within one interval. Using all computational resources on one interval at a time is often not the most efficient strategy because neither a sparse LDL^T factorization nor a triangular substitution can scale beyond a few hundred processors. It is usually much more efficient to allocate fewer processors to one interval and compute eigenvalues in several intervals in parallel.

Although both MISL and the MCISPM require solving linear systems, there are a number of notable differences between these two approaches for computing eigenvalues within each interval. The most important difference is perhaps the fact that linear equations must be solved in sequence in SIL, whereas more parallelism may be exploited in a subspace iteration when multiple equations are solved simultaneously. Thus, it may appear that better scalability can be expected from a subspace iteration method. However, when the lower triangular factor of A must be distributed among different processors, it is actually not easy to take advantage of multiple right hand sides to achieve full concurrency in triangular substitutions. Because only one substitution can be performed at a time, and the amount of parallelism is limited in each triangular substitution. Therefore, the actual parallel efficiency may be far more limited, as we will show in our experiments.

One drawback of MCISPM is that the construction of a projection operator requires multiple factorizations to be performed in complex arithmetic. Although these factorizations can be performed in an embarrassingly parallel way, each factorization is more expensive than the real arithmetic factorizations required in the SIL approach, and as a whole these factorizations, can consume more computational resource than what is required in MSIL for computing a comparable number of eigenvalues.

In terms of memory usage, MCISPM is also more costly than MSIL. This is partly because the factor L is a complex matrix in the contour integral based approach. Even though the matrix A is sparse, the space required for storing the factor L can be significant due to fill-ins during factorization. In addition, several (8 to 16) factorizations are required to compute eigenvalues within each interval. To perform them in parallel, additional (distributed) memory is required. The increased memory requirements of MCISPM can be a limiting factor in terms of performance as well.

The use of MSIL was considered in the work of [36]. The focus of their work is to use shift–invert with implicitly restarted Lanczos (IRL) to probe the spectrum dynamically as eigenvalues are being computed. The idea of using parallel contour integral based spectral projectors has been studied in [22,26,27,35]. However, the authors focused mostly on how to use the contour integral approach to compute eigenvalues within a single interval, and left the task of spectrum division to users. They did not discuss how to organize the entire computation in an efficient manner. In [35], the authors' comparison of the performance a parallel contour integral based spectral projection method with that of a single shift–invert Lanczos for computing 40 eigenvalues in a single interval may not be completely fair. In a single shift–invert Lanczos run, eigenvalues away from the shift may converge slowly. As a result, the cost of triangular substitutions was much higher than that associated with a factorization. We believe that a proper comparision should be made with respect to the use of MSIL [13], since multiple factorizations in complex arithmetic was used in the contour integral based approach.

In this paper, we focus on the issue of spectrum partitioning, which we believe is critical for achieving high efficiency regardless of whether a shift–invert IRL or a contour integral based spectral projection method is used to compute eigenvalues within each interval. We list a few objectives of spectrum division in Section 2, and discuss how the distribution of eigenvalues can be estimated through an inertia count and cubic spline fitting. The eigenvalue distribution profile enables us to obtain an optimal strategy for spectrum division. We present a simple cost model that can be used to develop some general strategies for partitioning the spectrum and allocating computational resources in an efficient way. Implementation details of MSIL and MCISPM are discussed in Section 3. In Section 4, we present a number of examples that illustrate the effect of different spectrum division schemes on the overall performance of eigenvalue computations for a set of test problems from density functional theory (DFT). We also compare the performance of MSIL with that of MCISPM on test problems with different sizes and sparsity patterns, and discuss the how spectrum division affects the strong and weak scaling properties of both approaches.

2. Spectrum division

Although it is clear that dividing the spectrum into a number of smaller intervals will increase concurrency and reduce the cost of both the orthogonalization and the Rayleigh–Ritz calculation, it is less obvious how the division should be done to achieve optimal overall computational efficiency.

We first discuss the objectives we would like to achieve by dividing the spectrum in Section 2.1. Achieving some of these objectives requires us to have some knowledge about the distribution of eigenvalues. Such a distribution can be described in terms of a histogram of eigenvalues or, borrowing a terminology from physics, the *density of states* (DOS). We discuss how to obtain an approximate DOS in Section 2.2. If the distribution of eigenvalues within the desired part of the spectrum (with spectral gaps excluded) is nearly uniform, a good strategy for dividing the spectrum may be developed by examining a simple cost model that depends on the size of the problem, the total number of processors available and the relative cost of a factorization versus a triangular substitution. We discuss this model and general strategies for dividing the spectrum in Section 2.3. Certainly, for problems whose eigenvalues are not uniformly distributed, a more sophisticated model that takes into account the potential presence of clustered or multiple (degenerate) eigenvalues is needed. Nonetheless, the simple model we introduce here can be used as a starting point for developing a more sophisticated model to understand more general cases.

2.1. Objectives

In general, a spectrum division scheme should try to achieve the following objectives:

1. The eigenvalues within each interval should be computed in an efficient manner by a group of processors. This objective suggests that we should exclude spectral gaps from each interval because the presence of such a gap often slows down the convergence of both the shift–invert Lanczos algorithm and the subspace iteration applied to the contour integral representation of the projector. It also suggests that we should limit the number of eigenvalues in each interval to keep the cost of the orthogonalization and the Rayleigh–Ritz calculation under control.
2. The division should make it easy to group the processors in such a way that allows desired eigenvalues to be computed in a load balanced way.
3. Since factorization is expensive, the spectrum should be divided in a way that minimizes the total number of factorizations.

Some of these objectives may conflict with each other. There may be trade-offs among different objectives to optimize resource usage. The most important objective we should try to achieve may depend also on the type of available computational resource. For example, dividing the spectrum into fewer intervals and allocating more processors to each interval will reduce the cost of factorizations (objective 3). However, because each interval contains more eigenvalues, the cost of basis orthogonalization and the Rayleigh–Ritz calculation may become excessively high. Furthermore, the convergence of some of the eigenvalues within each interval may become slow, for example, in a SIL calculation. As a result, more triangular substitutions may be required in each interval to reach convergence, making it difficult to achieve the first objective.

2.2. Density of states estimation

It follows from the Sylvester's inertia theorem [32] that the number of eigenvalues of A that are within an interval $[\alpha, \beta]$ can be determined by performing two factorizations on the shifted matrices: $A - \alpha I = L_\alpha D_\alpha L_\alpha^T$ and $A - \beta I = L_\beta D_\beta L_\beta^T$. The difference between the number of negatives entries in D_β and D_α yields the eigenvalue count within $[\alpha, \beta]$. Consequently, an estimation of the density of states (DOS) can be obtained by partitioning the spectrum into a number of intervals and performing a number of LDL^T factorizations. The partitioning can be done by either estimating the upper and lower bounds of the desired part of the spectrum λ_{ub} and λ_{lb} respectively first and dividing $[\lambda_{lb}, \lambda_{ub}]$ evenly into s intervals or running an s-step Lanczos iteration first and using s Ritz values $\theta_1, \theta_2, \ldots, \theta_s$ returned from the Lanczos procedure to partition the spectrum and construct a DOS profile. In both cases, s can be chosen to be a relatively small constant, say 100 first. When a large amount of computational resource is available, one can perform multiple parallel factorizations simultaneously.

Furthermore, we may use cubic spline fitting to construct a cumulative density of state profile with high resolution from which a refined DOS can be constructed [8]. A cumulative density of states function $c(\lambda)$ gives the total number of eigenvalues less than or equal to λ. By counting the number of eigenvalues to the left of θ_i, for $i = 1, 2, \ldots, s$, we can construct a crude approximation to the true cumulative density of the spectrum. The crude approximation can be improved by connecting $(\theta_i, c(\theta_i))$ with $(\theta_{i+1}, c(\theta_{i+1}))$ using cubic splines. Fig. 2 shows that the approximate cumulative density of states function constructed from 100 inertia counts and cubic spline fitting is nearly indistinguishable from the exact cumulative density of states. We observe from the cubic spline fitting that the matrix used in this example has a spectral gap between -2.0 and -1.7, and the distribution of eigenvalues is nearly uniform below and above the spectral gap.

The cubic spline fitting of the cumulative density provides a reliable way to evaluate $c(\lambda)$ and its derivative for any λ in the region where the DOS is relatively smooth. Consequently, we can find the approximate location of the jth eigenvalue by solving $c(\lambda_j) = j$ using either Newton's method or the bisection method. This capability allows us to define the upper and lower bounds of each interval regardless how the spectrum is partitioned. When λ_j is nearly degenerate, which can be detected by the large slope of the CDOS near λ_j, we should avoid using λ_j as a break point to divide the spectrum, i.e., it is important not to divide a cluster of eigenvalues during spectrum slicing.

We should point out that a DOS estimation based on inertial counts introduces some overhead to the overall cost of eigenvalue calculation. However, we believe that such cost is justified because the lack of a good DOS estimation makes it difficult to organize the parallel spectrum slicing computation in an efficient manner. When computational resources are not utilized efficiently, the overall cost of solving the problem would increase further. For very large problems where performing the LDL^T factorization is not feasible, it is possible to use iterative methods to estimate the DOS [19].

2.3. Strategy

We now discuss how to achieve some of the objectives listed in Section 2.1. To facilitate our discussion, we list some of the notation used throughout this paper to describe various quantities related to the size of the problem as well as the size and configuration of the computing platform in Table 1.

We assume that we have access to a relatively large number of processors. There are a number of ways to achieve the objective of load balance. One way is to keep the granularity of the computation sufficiently fine by partitioning the spectrum into many small intervals and using a relatively large number of processors to compute eigenvalues within each interval. In this case, we may have more intervals than the number of processor groups. Each processor group may be

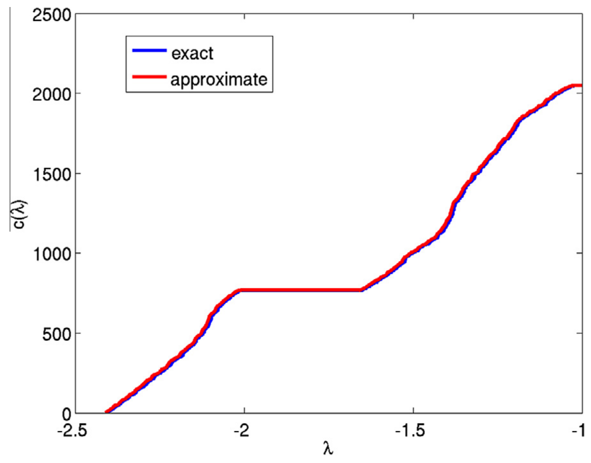


Fig. 2. The approximate cumulative density function of the first 2048 eigenvalues of a sparse matrix (red) constructed from 100 inertial counts and cubic spline fitting is nearly indistinguishable from the exact one (blue). (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this article.)

Table 1
Notation used to describe the problems as well as the configuration of the computing platform.

Symbol	Description
n	Matrix size
nev	The total number of desired eigenvalues
p	The total number of processors (cores) available
l	The number of intervals into which the spectrum is divided
k	The average number of eigenvalues per interval ($=nev/l$)
q	The number of processors (cores) used per interval ($=p/l$)

responsible for computing eigenvalues in multiple intervals. If eigenvalues within each interval can be computed quickly, then good load balance can be expected from fine grained parallelism.

However, there are a couple of issues associated with this approach. First of all, because there is a limit on how far a sparse LDL^T factorization and a triangular substitution can scale with respect to the number of processors, the granularity of each parallel task cannot be made arbitrarily small.

Secondly, it is generally more costly to partition a task into m subtasks and assign all processors to work in parallel on each subtask, one after another, than to divide the processors into m groups, with each group being responsible for a single subtask, unless the parallel efficiency of having all processors work on each subtask is perfect. To see why this is the case, let us assume that the total amount of work required to compute eigenvalues within an interval is C, and the wall clock time required to compute these eigenvalues using p processors is proportional to C/p^{η}, where the parameter $0 < \eta < 1$ is used to account for parallel efficiency. If we now divide the interval into m subintervals and use all p processors to compute $1/m$ of the eigenvalues at a time, and repeat the process m times, the wall clock time is still proportional to C/p^{η} if the parallel efficiency does not change with respect to the number of eigenvalues to be computed. However, if we use p/m processors to compute eigenvalues within each one of the m subintervals with the same parallel efficiency, the wall clock time is proportional to

$$\frac{C/m}{(p/m)^{\eta}} = \frac{C}{p^{\eta}} m^{\eta-1} < \frac{C}{p^{\eta}}.$$

Therefore, it is generally more effective to partition a group of processors in the same way the interval is partitioned so that eigenvalues within each subinterval are computed by a subgroup of processors.

This observation suggests that a better way to achieve load balance is to partition the spectrum in such a way that the amount of work associated with each interval is roughly the same. In this case, the processors can be divided into the same number of groups as the number of interval, with each group being responsible for computing the eigenvalues within a single subinterval. This approach may not be feasible if the number of available processors is relatively small compared to the number of eigenvalues to be computed. However, this is not the regime that we are primarily concerned with in this paper. Some difficulties may also arise when the eigenvalues of interest are not evenly distributed. In this case, the spectrum may be divided in such a way that the amount of work associated with different intervals is not exactly the same. As a result, the grouping of the processors need to be adjusted accordingly so that it is consistent with the amount of work associated with each interval. Making this adjustment is possible when the DOS profile is available.

Although it is generally not easy to answer the question of how to partition the desired part of the spectrum, and how many processors to allocate to each interval in order to optimize the use of computational resources, a reasonable strategy may be developed if we can characterize the computational cost using a simple model. Using such a model, we can then analyze how other objectives listed in Section 2.1 may be achieved and how the effectiveness of a partition may be affected by a number of factors such as the size of the problem, the relative cost and efficiency of factorization versus triangular substitution, and the number of processors available.

To simplify our analysis, let us assume that eigenvalues are uniformly distributed in the desired part of the spectrum. Using the notation in Table 1, we assume that p and nev are relatively large. To be more precise, let us assume that $p = c_p n$ and $nev = c_n n$, where c_p and c_n are constants. We assume that the number of eigenvalues within each interval k is sufficiently small so that the cost of computing eigenvalues by either the shift–invert Lanczos method or the contour integral based projection method is dominated by sparse matrix factorizations and triangular substitutions. We will show through extensive numerical experiments that in the vicinity of the optimal k value, the cost of the orthogonalization and the Rayleigh–Ritz computations are indeed much lower compared to the cost of sparse factorizations and triangular substitutions. So the simple analysis presented in this section allows us to better understand the trade-offs in the range of k that is of interest to us. For completeness, we also discuss in more detail about the model for the wall clock time in the Appendix, in which we add the cost for the matrix–matrix multiplication (MM) to form the projected problem and to obtain the orthogonal eigenvectors (orthogonalization), and the cost for the Rayleigh–Ritz procedure (RR) for solving the projected problem. We will see that when k and n are not very large, the cost of the MM and RR procedure is indeed smaller than that of sparse factorizations and triangular substitutions.

The wall-clock time for performing a factorization on a single processor can be estimated by $c_f n^{\alpha_f}$ for some constant c_f and α_f. The wall clock time for a single triangular substitution performed on a single processor can be modeled as $c_s n^{\alpha_s}$ for some constant c_s and α_s. The constants $c_f, c_s, \alpha_f, \alpha_s$ are problem and/or machine dependent. The matrices used in the numerical experiments in Section 4 mainly originate from density functional theory based electronic structure calculations that are discretized by finite difference, finite elements or real space local basis expansion techniques. The sparsity of these matrices are similar to that of the discretized Laplacian operator. The discretized operators that share a sparsity pattern similar to the discretized Laplacian operator also appear frequently in other scientific computing problems such as the Poisson equation and the Helmholtz equation. For d-dimensional Laplacian type of operators discretized by finite difference and ordered by the nested dissection technique [10], the values of α_f, α_s are given in the second and the third columns of Table 2. For other general sparse matrices, the values of α_f and α_s will depend on the number of nonzeros in the factor L, which is in turn determined by matrix ordering scheme (e.g. [11]) used to permute the rows and columns of A to minimize the amount of nonzero fills in L.

Table 2
Asymptotic wall clock time using one processor for the factorization cost α_f, the triangular solve cost α_s, and the optimal number of eigenvalues k^{opt} for a Laplacian operator discretized by second order finite difference method in dimension d.

d	α_f	α_s	k^{opt} $(\eta_f = \eta_s)$
1	1	1	$\mathcal{O}(1)$
2	$\frac{3}{2}$	1	$\mathcal{O}(n^{\frac{1}{2}})$
3	2	$\frac{4}{3}$	$\mathcal{O}(n^{\frac{2}{3}})$

We model the parallel efficiency for the factorization and the triangular substitution by two parameters η_f, η_s respectively. These parameters satisfy $\eta_f, \eta_s \in (0,1)$, and usually $\eta_f \geqslant \eta_s$. When p processors are used to perform a factorization and a triangular substitution, the wall clock time required is proportional to $c_f n^{\alpha_f}/p^{\eta_f}$ and $c_s n^{\alpha_s}/p^{\eta_s}$, respectively. To simplify our discussion, we assume that the range of k we consider corresponds to cases in which the cost associated with the orthogonalization and the Rayleigh–Ritz calculation is relatively low compared to that associated with factorization and triangular substitutions. In addition, we assume that the number of triangular substitutions used per eigenvalue is constant across the range of k under consideration.

Given the assumptions above, the wall clock time $W(k)$ of the calculation can be expressed by

$$W(k) = \frac{c_f n^{\alpha_f}}{\left(\frac{p}{nev/k}\right)^{\eta_f}} + \frac{c_s n^{\alpha_s} k}{\left(\frac{p}{nev/k}\right)^{\eta_s}}. \tag{2}$$

Minimizing $W(k)$ with respect to k gives rise to

$$k^{\text{opt}} = \left(\frac{\eta_f c_f}{(1-\eta_s)c_s}\right)^{\frac{1}{\eta_f - \eta_s + 1}} \left(\frac{c_n}{c_p}\right)^{\frac{\eta_f - \eta_s}{\eta_f - \eta_s + 1}} n^{\frac{\alpha_f - \alpha_s}{\eta_f - \eta_s + 1}}. \tag{3}$$

Although it is possible to include extra terms in (2) to account for the cost of the orthogonalization and the Rayleigh–Ritz calculation, introducing these terms makes it difficult to obtain an analytic and transparent expression for k^{opt}. Also as we show through numerical experiments in Section 4, the costs of the orthogonalization and the Rayleigh–Ritz calculations are relatively low in the range of k that we consider. Therefore the true minimizer for the more sophisticated model is likely to be quite close to the solution given by (3).

The expression given by (3) indicates that choice of an optimal k depends on the problem size n as well as the relative cost and parallel efficiency of factorization and triangular substitutions, when c_n/c_p is a constant. When the parallel efficiencies of both the factorization and the triangular substitution are high, i.e., when both η_f and η_s are close to 1, we should place more eigenvalues in each interval without increasing the cost of the orthogonalization and the Rayleigh–Ritz calculation too much. When η_f and η_s are relatively small, k^{opt} will be smaller as well. A smaller k^{opt} would suggest dividing the spectrum into more intervals and breaking the processors into smaller groups. However, there is often a lower bound on the number of processors in each group due to the memory requirement for holding the lower triangular factor of $A - \sigma I$ or $A - z_i I$. We should also point out that the parallel efficiency parameters η_f and η_s depend in a nontrivial way on the actual number processors used in the factorization and triangular substitutions, which is ultimately related to c_p. When the number of processors used in each factorization or triangular substitution increases, both η_f and η_s may decrease. Such loss of parallel efficiency implies that we need to divide the spectrum into more intervals with fewer eigenvalues in each interval as the total number of processors increases.

While in general factorization is expected to scale better than a triangular substitution (e.g., $\eta_f > \eta_s$), our numerical experiments show that when c_p is not too large (e.g., $c_p = 0.1$), $\eta_f \approx \eta_s$, and they do not depend strongly on c_p. This can be explained by the additional overheads incurred during the sparse matrix factorization phase, such as the analysis step necessary to determine a good pivoting strategy and the distribution of the sparse matrix over the processor group. By taking $\eta_f \approx \eta_s$, Eq. (3) can be simplified, and the minimizer of (3) has the form

$$k^{\text{opt}} = \left(\frac{\eta c_f}{(1-\eta)c_s}\right) n^{\alpha_f - \alpha_s}, \tag{4}$$

where $\eta = \eta_f = \eta_s$. It is interesting to see that in this case, k^{opt} is independent of c_p which determines the total number of processors used in the computation, and we should put more eigenvalues in each interval as the problem size n increases. Furthermore, for Laplacian type operators defined in d-dimensional space, the optimal number of eigenvalues to place in each interval also depends on d because both α_f and α_s are functions of d in this case. The fourth column of Table 2 gives the dependency of k^{opt} on n for the Laplacian type of operators in different spatial dimensions.

It is easy to show that when the factorization and triangular substitutions have roughly the same parallel efficiency, the wall clock time associated with k^{opt} is

$$W(k^{\text{opt}}) = \frac{c_f n^{\alpha_f}}{u} + \frac{\eta}{1-\eta}\frac{c_f n^{\alpha_f}}{u}, \tag{5}$$

where $u = (pk^{\text{opt}}/nev)^{\eta}$. The expression given by (5) indicates that the first and second terms of W are roughly the same at k^{opt} unless η is either close to 1 or 0. If $\eta = 0.5$, the second term in (5) is identical to the first term. When k is large, the second term in $W(k)$ can be much larger than the first term, which is usually an indication that such a choice of k is not optimal.

The cost model given by (2) allows us to make some prediction on the weak scaling property of MSIL and MCISPM. We now give an example in which both the problem size n and the number of eigenvalues nev, as well as the total number of processors p increase by a factor of 4.

If we increase the number of processors per interval q by a factor of 4 without changing the number of intervals l, the number of eigenvalues per interval k is also increased by a factor of 4 (due to the increase of the problem size). It follows from (2) that the total wall clock time for factorization will increase by a factor of $4^{\alpha_f - \eta_f}$ and the wall clock time for triangular substitution will increase by a factor of $4^{\alpha_s - \eta_s + 1}$. If we assume $\eta_f = \eta_s = 1/2$ and the costs associated with factorization and triangular substitutions are roughly the same before q and k are increased, then for 2D problems, we expect the total wall clock time to increase by a factor

$$\frac{4^{\alpha_f - \eta} + 4^{\alpha_s - \eta + 1}}{2} = \frac{4^1 + 4^{3/2}}{2} = 6.$$

If we keep the number of cores per interval q and the number of eigenvalues per interval k fixed, and simply increase the number of intervals l by a factor of 4, then based on the model (2), we expect the total wall clock time to be increased by a factor of

$$(4^{\alpha_f} + 4^{\alpha_s})/2 = \frac{4^{3/2} + 4^1}{2} = 6,$$

if the same assumptions about the relative cost of factorization and triangular substitutions before n and p are increased are made.

The same type of calculation shows that, under the same assumption about factorization and triangular substitutions, if we increase both k and q by a factor of 2, the total wall clock time is expected to increase by factor of 5.6.

This simple analysis shows that when using a larger number of processors to solve larger problems, it may be beneficial to split the spectrum into more intervals and putting more processors into each interval at the same time. This analysis is confirmed by our computational experiments which we show in Section 4.

3. Implementation issues

In this section, we discuss a number of implementation issues pertaining to both the MSIL and the MCISPM algorithms. As reported in [13], an important issue that may arise in spectral partitioning approaches is that eigenvectors returned from adjacent intervals may not be fully orthogonal. This issue is more likely to arise when eigenvalues that belong to two adjacent intervals are close to each other. As discussed in Section 2.2, this issue may be avoided by carefully partitioning the spectrum based on the DOS estimation such that a cluster of eigenvalues is not divided into different slices.

3.1. Multiple shift–invert Lanczos

Shift–invert is a widely used technique for computing interior eigenvalues when using the Lanczos algorithm. Because eigenvalues near a target shift σ are mapped to the extremal eigenvalues of the shifted and inverted matrix $(A - \sigma I)^{-1}$, through the equation

$$(A - \sigma I)^{-1} x = \frac{1}{\lambda - \sigma} x,$$

where (λ, x) is an eigenpair of A, the eigenvectors associated with these eigenvalues tend to converge rapidly when we apply the Lanczos procedure to $(A - \sigma I)^{-1}$.

When the desired part of the spectrum is divided into several intervals, we can apply the shift–invert Lanczos procedure in each interval simultaneously with an appropriately chosen shift so that eigenvalues within the interval can be computed quickly. Multi-shift was also suggested by Ruhe in his rational Krylov subspace method [23].

As discussed earlier, the partitioning of the spectrum can be obtained by first estimating the density of state profile using the technique described above, and using the simple model presented in Section 2.3 to decide how many intervals we should create with an appropriate number of eigenvalues in each interval. Such a decision may require us to perform some computational experiments (often referred to as *automatic tuning* in the high performance computing community), to estimate various parameters in the model. We should also avoid defining an interval that contains a sizable spectral gap even if that means we may end up with some intervals that contain fewer eigenvalues. This approach is very different from the one taken in [36], where each processor determines how many eigenvalues to compute in a dynamic way.

If the total number of processors available p is sufficiently large, which is the case that we are primarily concerned with, and if the eigenvalues are uniformly distributed, we can allocate p/l processors to each interval once the total number of intervals l is chosen.

In most cases, the target shift for each shift–invert Lanczos (SIL) run can be set as the mid-point of each interval. We perform at least one parallel LDL^T factorization in each interval.

One of the difficulties with using the Lanczos algorithm is that it is difficult to predict a priori how many Lanczos iterations it takes to obtain all eigenvalues within each interval. Even if the number of eigenvalues k within each interval is small, the dimension of the Krylov subspace generated by the standard Lanczos algorithm may be much larger. This problem can partially be solved by using the implicit restarting technique [29] to limit the dimension of the Krylov subspace from which approximate eigenvalues of $(A - \sigma I)^{-1}$ or A are extracted. Limiting the dimension of the Krylov subspace reduces the memory footprint of the calculation. It also prevents the cost of Rayleigh–Ritz procedure from becoming too high.

In an implicitly restarted Lanczos (IRL) procedure, we typically construct and update a Krylov subspace whose dimension is slightly larger than the number of desired eigenvalues k. In our experiments, we typically use $2k$ as the maximum subspace dimension. We perform $2k$ triangular substitutions in the first shift–invert IRL cycle. In each subsequent IRL cycle, we perform k additional triangular substitutions. We perform full orthogonalization of each basis vector against previous basis vectors in each restart. The cost of this orthogonalization is modest since we keep k small. Furthermore, in most cases, no further reorthogonalization is needed in addition to the first full orthogonalization.

We set a limit on the number of restart cycles in each shift–invert IRL run, so that the total number and cost of triangular substitutions do not become too high. We also set k to be slightly larger than the actual number of eigenvalues contained in an interval to reduce the likelihood of missing any eigenvalues within the interval. However, it is possible that some of the desired eigenvalues fail to converge before the limit on the restart cycles is reached. In that case, we may need to perform a post-processing procedure to capture the missing eigenpairs. Such a procedure would require an additional factorization and a few more triangular substitutions. Hence, in that case, the entire computation may not be well load balanced.

3.2. Contour integral based projection method

If P is an orthogonal projection operator that projects onto the invariant subspace Z associated with a subset of k eigenvalues of A, we may compute this invariant subspace by applying a subspace iteration to P [30].

Let V be a random k-dimensional subspace that contains contributions from all eigenvectors of A. In exact arithmetic, we need to apply P to V only once in order to obtain the desired the invariant subspace Z, if the exact P is available. In practice, because the projection operator P given by (1) is approximated by a numerical quadrature

$$P \approx \sum_{i=1}^{q} \omega_i (A - z_i I)^{-1} \tag{6}$$

for some appropriately chosen quadrature points z_i and weights ω_i $(i = 1, 2, \ldots, q)$, more than one iteration may be needed to obtain Z with sufficient accuracy.

In the numerical experiments that we show in Section 4, we use the quadrature rule proposed in [22] to evaluate the contour integral (1). For each pole z_i, we must perform an LDL^T factorization of $A - z_i I$ in complex arithmetic. This is roughly four times more costly than an LDL^T factorization performed in SIL. The higher factorization cost would suggest that a larger number of processors should be allocated to each interval (or contour) to ensure that each factorization can be computed efficiently in parallel. These processors are further partitioned among n_q quadrature points. As a result, the number of processor groups and, consequently, the number of intervals l will decrease, and we need to place more eigenvalues in each interval. However, placing too many eigenvalues within a single interval will lead to a higher cost due to triangular substitutions. Since $(A - z_i I)^{-1}$ must be applied to all columns of V at each pole z_i, we must perform k triangular substitutions at each pole. Such increase must be taken into account when we decide what the optimal k should be.

Although it may appear that performing multiple triangular substitutions is an inherently parallel task, the fact that the triangular factors of $A - z_i I$ are distributed makes it difficult to achieve a high level of parallelism during triangular substitutions, even when a large number of right hand sides are available. Some gain in efficiency can be achieved by making use of level-3 BLAS with an appropriate block size when multiple substitutions are performed simultaneously. However, there is a limit on how large this type of gain can be. It typically does not exceed a factor of 3, and is quickly offset by a large increase in the number of right hand sides.

4. Computational experiments

In this section, we present a number of computational experiments that demonstrate the efficiency that can be achieved by both MSIL and MCISPM. These examples also illustrate the importance of the issues we discussed in the previous section and how one may achieve nearly optimal performance by making a proper choice of interval size and processor groups.

A number of medium sized sparse matrices are used in our experiments. The matrices are listed in Table 3. The `Graphene`X matrices are obtained from a discretization of the Kohn–Sham equation using an adaptive local basis expansion scheme [18]. The number X indicates the number of carbon atoms in the Graphene. The dimension of these matrices is proportional to the number of atoms in these quasi-2D materials. These matrices have roughly 1–4% nonzeros per column and 10–15% nonzero fill-in's in their L factors. They are denser than a discretized Laplacian with 5-point stencil, but should still be considered as sparse matrices. All other matrices are obtained from a finite difference discretization of the Kohn–Sham

Table 3
Problem characteristics.

Name	n	nnz	L_{nnz}	Factor ops
Graphene128	5120	1,026,560	3,100,160	2.56×10^9 ($\approx 97n^2$)
Graphene512	20,480	4,106,240	43,261,440	1.12×10^{11} ($\approx 267n^2$)
Graphene2048	81,920	16,424,960	135,445,760	5.35×10^{11} ($\approx 79n^2$)
Graphene8192	327,680	65,699,840	727,511,040	6.42×10^{12} ($\approx 60n^2$)
C60	17,576	212,390	19,573,813	3.74×10^{10} ($\approx 121n^2$)
GaAsH6	61,349	3,016,148	247,861,525	1.73×10^{12} ($\approx 460n^2$)
SiO	33,401	675,528	89,796,509	4.03×10^{11} ($\approx 361n^2$)

equation for 3D materials. They are produced by the PARSEC software package [14], and tend to be sparser than the Graphene matrices. However, as we can see from Table 3, their L-factors tend to be slightly denser than those associated with the Graphene problems.

We performed our numerical experiments on Hopper, a Cray XE6 supercomputer maintained at the National Energy Research Scientific Computer Center (NERSC) at Lawrence Berkeley National Laboratory. Each node on Hopper has 2 twelve-core AMD Magny Cours 2.1-GHz processors with a total of 32 gigabyte (GB) shared memory. However, memory access bandwidth and latency are nonuniform across all cores. Each core has its own 64 kilobytes (KB) L1 and 512 KB L2 caches. One 6 MB L3 cache is shared among 6 cores on the Magny Cours processor. There are four DDR3 1333-MHz memory channels per twelve-core Magny Cours processor.

The PARPACK software package [16,20] is used for MSIL experiments. We use a modified version of the FEAST program [22], which we call FEAST-Mod, to test the performance of MCISPM. Our modification allows multiple factorizations to be performed in parallel within a subgroup of processors. We set the convergence tolerance *tol* to 10^{-10} in both experiments, i.e., we terminate SIL or a subspace iteration when the relative residual norm of an approximate eigenpair (θ_i, z_i), defined by $\|Az_i - \theta_i z_i\| / |\theta_i|$ falls below *tol* for all $i = 1, 2, \ldots, nev$. We did not experience any robustness issues, in terms of missing eigenvalues or large residual norms, in our numerical experiments.

We use the MUMPS (Multifrontal Massively Parallel sparse Direct Solver) [1] to solve the linear system of equations in both MSIL and MCISPM. We should note that the performance of the sparse solver has a direct effect on the absolute performance of the eigenvalue calculation. However, some of the issues we will discuss below is independent of the sparse solver used in the computation.

4.1. MSIL

In this section, we study the parallel performance of *MSIL*, when it is used to compute roughly 10% of the lowest eigenvalues of matrices given in Table 3. We first show that the way the spectrum is partitioned and processors are grouped (i.e. the choice of an appropriate number of eigenvalues k for each interval) has a significant impact on the performance of *MSIL*. We then present the results of a strong scaling study, where we examine the performance of *MSIL* as we increase the number of processors p for a given problem. Finally, in the weak scaling study, which is performed using the Graphene matrices only, we investigate the performance characteristics of *MSIL* when the number of processors is increased at the same rate as the increase in matrix dimensions.

4.1.1. Spectrum division and processor groups

As we discussed earlier, for a given problem and a fixed number of processors p, the performance of *MSIL* depends on how the spectrum is divided and how many eigenvalues are placed in each interval. In Fig. 3, we show how the performance of MSIL changes as we change the way the desired part of the spectrum of the Graphene512 problem is partitioned. In this test, we used a total of $p = 512$ cores to compute $nev = 2048$ eigenvalues. As the spectrum is partitioned into l intervals with roughly $k = nev/l$ eigenvalues within each interval, the computational cores are divided into l groups with roughly $q = 512/l$ cores in each group.

The height of each color coded bar in Fig. 3 gives the total wall clock time used to compute all desired eigenvalues for a particular combination of l and q that satisfy $l \times q \approx 512$. The $l \times q$ combination associated with each bar is labeled on the horizontal axis. The blue portion of each bar shows the portion of the wall-clock time spent in factorization. The red portion represents the average wall clock time spent in triangular substitutions. Because the number of IRL restart cycles used in different intervals may be slightly different, there is some difference in the actual number of triangular substitutions performed in different intervals. The difference between the maximum and average amount of wall clock time spent in triangular substitutions over all intervals is shown by the green portion of each bar. This portion measures how well the computation is load balanced.

As the cost model and analysis given in Section 2.3 suggests the actual value of k^{opt} depends on a number of architecture and sparse solver dependent parameters such as c_f, c_s, η_f and η_s. We observe in Fig. 3 that by creating $l = 128$ intervals with $k \approx 16$ eigenvalues within each interval, the factorization, which is performed by $q = 4$ cores in parallel, is much more costly

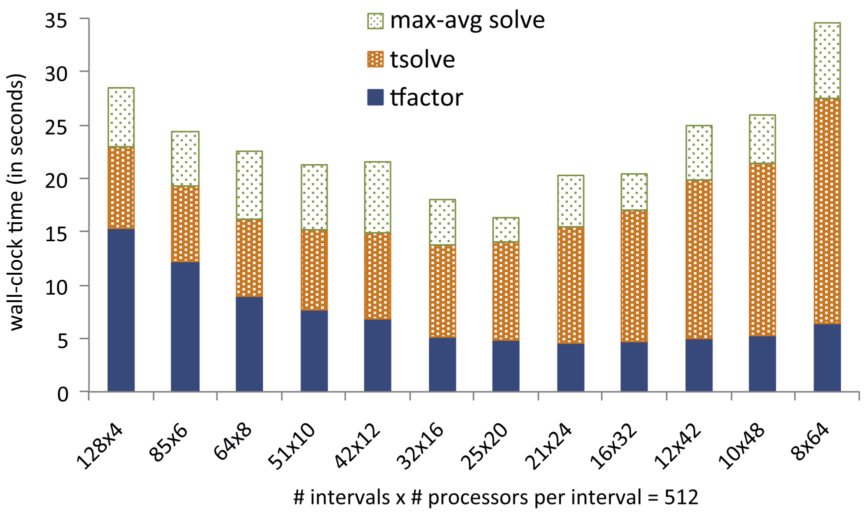


Fig. 3. The effect of interval size.

than triangular substitutions. As a result, the total wall clock time is far from optimal. On the other hand, when only $l = 8$ intervals are created with $k = 256$ eigenvalues per interval, an excessive amount of time is spent in triangular substitutions. This combination leads to a sharp increase in the total wall clock time also.

The minimum wall-clock time is achieved when $l = 25$ and $q = 20$ – about 80 eigenpairs per interval. In this case, the difference between the maximum and average times spent on triangular substitutions is relatively small indicating that the computation is reasonably well load balanced, and the wall-clock times used in factorization and triangular substitutions are roughly the same, too. This observation agrees well with our prediction made in Section 2.3. For other $l \times q$ combinations, the difference between the maximum and average times spent on triangular substitutions can be large. This indicates one of the challenges of MSIL, which is to ensure that eigenvalues in different intervals converge at approximately the same rate.

In Table 4, we give the percentage of the total time that *MSIL* spends on the orthogonalization procedure using matrix–matrix multiplication and the Rayleigh–Ritz computations (`pdsaupd`) for Graphene512. As this numerical experiment suggests, the cost of the orthogonalization and the Rayleigh–Ritz computations are much lower (ranging from 0.4% to only up to 2.1% for all combinations tested) compared to the cost of sparse factorization and triangular substitutions in the range of k that is of interest to us. These findings provide the justification for neglecting the cost of orthogonalization and Rayleigh–Ritz computations to obtain a simple cost model in Section 2.3.

4.1.2. Strong scaling

In this section, we examine how efficiently *MSIL* can utilize an increasing number of processors for a given problem. Since our model indicates that the wall clock time of MSIL depends on how the spectrum is partitioned as well as the relative cost and parallel efficiency of factorization and triangular substitutions, which are themselves a function of the number of processors used, performing a strong scaling study is not straightforward.

Fig. 4 shows the strong scaling of *MSIL* for Graphene matrices, which originates from quasi-2D problems. For each problem, we plot the best total wall clock time against the total number of processors used in the computation, where the best wallclock time is chosen from various $l \times q \approx p$ combinations for all p's. We label each data point by the optimal $l \times q$ combination associated with the best measured wall clock time. While the points on the dashed lines indicate the maximum wall-clock time taken by any processor group in the best combination, the points on the solid line indicate the average wall-clock time taken by all processor groups.

The general trend we observe is that as the number of processors increases for a given problem, there is an increase both in the number of intervals l used and in the number of processors q assigned to each interval. Intuitively, this can be interpreted as follows. Increasing the number of processors in an interval decreases factorization time. However, since the parallel efficiency of triangular substitution typically degrades faster than that associated with factorization, the decrease

Table 4
Time spent by *MSIL* in matrix–matrix multiplication for orthogonalization and Rayleigh–Ritz (MM+RR) computations in terms of the percentage of total wallclock time.

Combination	85 × 6	64 × 8	51 × 10	42 × 12	32 × 16	25 × 20	21 × 24	16 × 32	12 × 42	10 × 48	8 × 64
Nev/interval	24	32	40	48	64	82	98	128	171	205	256
% MM+RR	0.4	0.5	0.7	0.7	1	1.2	1.3	1.5	1.7	1.8	2.1

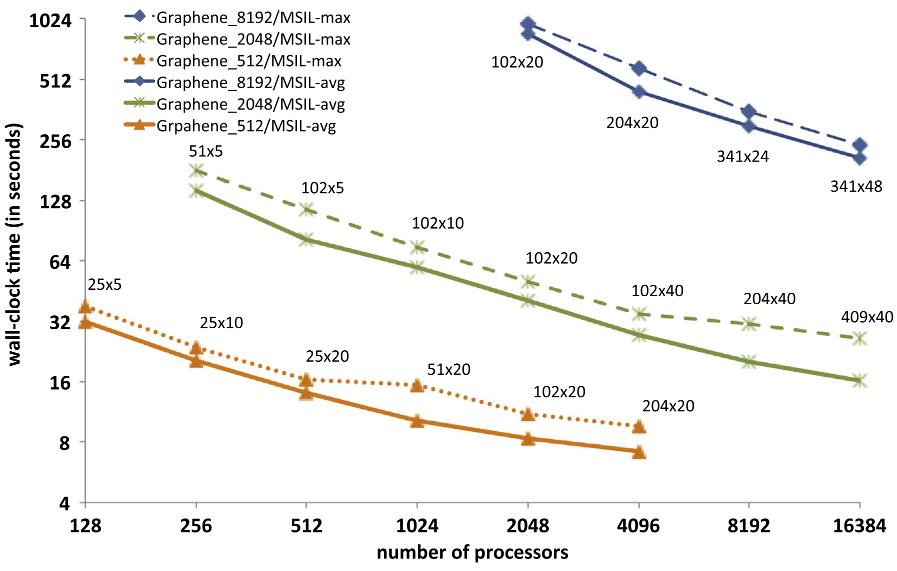


Fig. 4. The parallel (strong) scalability of *MSIL* for quasi-2D problems. The height of each data-point give the total wall clock time used in the computation. Data-points on the solid lines indicate average values, whereas data-points on the dashed lines indicate maximum values. Each data-point is labeled by the optimal $l \times q$ combination, where l is the number of processor groups and q is the number of processors within each group.

in the time for triangular substitutions is limited. By partitioning the spectrum into slightly more intervals, additional reduction in the time for triangular substitutions can be achieved, since each interval will have fewer linear equations to solve.

The observation above is also consistent with the cost model presented in Section 2.3. Our model indicates that k^{opt} will decrease as c_p increases, resulting in more intervals, when all other parameters are held fixed and $\eta_f > \eta_s$. However, we notice that the increase of the number of intervals l is slower than the increase of total number of cores. That means more cores are assigned to each interval also, as p increases. This strategy has the combined effect of reducing both the factorization and total triangular substitution times, as p increases.

4.1.3. Weak scaling

For the same reasons discussed in the previous section, the weak-scaling of *MSIL* is not easy to measure either. We are interested in learning how the wall clock time changes when both the problem size (n) and the number of cores (p) used in the computation are increased at the same rate. The analysis we presented in Section 2.3 indicates that the change in wall clock time depends on how the spectrum is partitioned as n and p increase at the same rate. As we move from Graphene512 ($n = 20,480$) to Graphene2048 ($n = 81,920$) in Table 5, the number of intervals associated with the best wall clock time increases from $l = 25$ to $l = 102$. As a result, the number of eigenvalues within each interval and the number of processors used for each interval stays the same. In this case, our model predicts that the wall clock time used for factorization will increase by a factor of 4^{α_f}, and the triangular substitution time will increase by a factor of 4^{α_s}, if c_f and c_s do not change as n changes. These changes are independent of η_f or η_s. Since Graphene is a 2D problem, $\alpha_f = 3/2$ and $\alpha_s = 1$. Using the factorization and substitution timing measurements recorded for $n = 20,480$ and $p = 512$, which are $t_f = 4.8$ and $t_s = 11.6$ respectively, we can estimate the expected factor of increase in wall clock time is roughly $(4^{3/2} \times 4.8 + 4 \times 11.6)/(4.8 + 11.6) \approx 3.2$. Table 5 shows that measured increase factor in wallclock time from $n = 20,480, p = 512$ to $n = 81,920, p = 2048$ is about $50.6/16.4 \approx 3.1$. Therefore the weak scaling from Graphene512 to Graphene2048 is nearly perfect.

A similar estimate of the increase in wall clock time can be made when n and p increase from $81,920$ and 2048 to $327,680$ and 8192 respectively. In this case, k increases from 80 to approximately 96 (a factor of as 1.2) l increases from 102 to 341. These changes yield an increase in factorization time by a factor of $4^{3/2}/1.2^{\eta_f}$, and in the time for triangular substitutions by a factor of $4/1.2^{\eta_s - 1}$. Using the simplified assumption that $\eta_f = \eta_s = 1/2$ and the measured t_f and t_s, we predict the total wall clock time to increase by a factor of $(4^{3/2} \times 14.8/\sqrt{1.2} + 4 \times \sqrt{1.2} \times 35.8)/(14.8 + 35.8) \approx 5.2$. Our measured increase is

Table 5
Weak scaling of *MSIL* for the 2D problems.

Problem	p	$q \times l$	k	t_f	Mean t_s	Max t_s	t_{wall}
Graphene512	512	20×25	82	4.8	9.2	11.6	16.4
Graphene2048	2048	20×102	80	14.8	25.6	35.8	50.6
Graphene8192	8192	24×341	96	108	190	245	353

$353/50.6 \approx 7$, which is slightly worse than our prediction. As can be seen in Table 5, the extra computational time mainly results from the degradation of the parallel performance of the factorization and triangular substitution routines, for which the increase of the computational time is $108/14.8 \approx 7.3$, and $245/35.8 \approx 6.8$ respectively. This is worse than the theoretical predictions.

4.2. MCISPM

In this section, we examine the performance of MCISPM by reporting the wall clock time used by the FEAST-Mod program to compute roughly 10% of the lowest eigenvalues of each matrix. We first show that the best way to divide the spectrum to achieve maximum efficiency for a fixed amount of computational resource is a nontrivial problem. We then study how the computation scales with an increasing amount of computational resource in both the strong and weak sense.

An additional parameter that can significantly affect the performance of MCISPM is the number of quadrature points used in an interval. Using fewer number of quadrature points would require fewer factorizations within an interval. Such reduction would lead to significant savings in computational costs. However, fewer quadrature points may not yield a good approximation to the contour integral, and therefore may require more subspace iterations to attain the desired accuracy. So first we investigate how the overall performance of FEAST-Mod depends on the number of quadrature points used. We have chosen Graphene512 as a representative problem for the 2D cases and C60 for the 3D cases. In Table 6, we present how using 16 quadrature points per interval performs against the best overall option which includes using 8, 10, 12, 16, 20 and 24 quadrature points.

Our results indicate that using 16 quadrature points gives the best overall performance for the problems that we study. Therefore we use 16 quadrature points in approximating the contour integral in (6) in all our experiments. If the total number of cores used to compute eigenvalues within a single contour is q, then $q/16$ cores are assigned to each pole to perform the factorization of $A - z_i I$ and solve triangular systems required in the subspace iteration. In all experiments below, we apply two subspace iterations to the approximate projection operator constructed from the approximate contour integral of the resolvent. They yield eigenpairs whose relative residual norms are all below the tolerance of 10^{-10} (actually below 10^{-12} in most cases).

4.2.1. Spectrum division and processor groups

In Table 7, we report the total wall clock time t_{wall} for different $q \times l = p$ combinations. Although it is possible to compute all desired eigenvalues by enclosing them in a single contour, Table 7 shows that this may not be the optimal way of using the contour integral projection method for a fixed amount of computational resource. In this experiment, we use $p = 512$ cores to compute 2048 (10%) lowest eigenvalues of the Graphene512 matrix. Table 7 shows that the most efficient way to obtain all desired eigenvalues is to break the interval into $l = 8$ subintervals, with each subinterval containing $k = 256$ eigenvalues. We also divide 512 cores into 8 groups with $q = 64$ cores in each group. Each group of 64 cores is responsible for computing 256 eigenvalues within one interval.

In Table 7, we also report the time used for factorization (t_f), triangular substitutions (t_s) and matrix–matrix multiplication procedure (for orthogonalization) together with the Rayleigh–Ritz procedure (t_{mm+rr}), respectively. As seen in Table 7, when the number of eigenvalues in an interval is nearly optimal, e.g. $k = 256$ or $k = 512$, the time spent in MM and RR is relatively small, 1.4% and 5.9%, respectively. However, as the number of eigenvalues within an interval becomes too large, e.g. $k = 2048, t_{rr+mm}$ can be quite significant (32%).

We chose to include the cost of sparse matrix vector multiplications (SpMVs), which are necessary for the subspace projection, in the column labeled by t_{other}. The cost of SpMVs is usually much lower than that of triangular substitutions. Also

Table 6
Relative performance of using 16 quadrature points (QP) per interval with respect to using 8, 10, 12, 16, 20 or 24 QP. If 16 is not the best option, the number in parentheses indicate the QP that attain the best performance in each case. The intervals chosen do not contain spectral gaps. The total number of processor cores used is 128.

Nev/interval	50	100	200	400
Graphene512	1.03 (10)	1.0	1.0	1.0
C60	1.05 (8)	1.0	1.0	1.0

Table 7
The performance profile of MCISPM for 512 processors, using different combinations of $q \times l = 512$, to compute the lowest 2048 eigenvalues of Graphene512.

$q \times l$	k	t_{wall}	t_f	t_s	t_{mm+rr}	t_{other}
512×1	2048	466	11	246	150	60
256×2	1024	260	16	152	58	32
128×4	512	146	26	94	8.6	17
64×8	256	121	45	67	1.7	8

note that depending on the value of the shift and the physical layout of the processors assigned to different quadrature points, t_f and t_s will show variations. This potential load-imbalance is also reported in t_{other}. The relatively large value of t_{other} mainly stems from the non-optimal implementation of the parallel SpMVs. Currently in MCISPM, multiple SpMV tasks in an interval are distributed among the processors in a group, where each processor performs its share of SpMV tasks serially one after the other. Sparse matrix dense vector block multiplication (SpMM) or other methods reported in the literature can certainly be used to speed-up this part of the computation.

The reason that choosing $l = 1$ and $q = 512$ leads to the most time consuming combination can be explained by the significant amount of time spent in triangular substitutions, matrix–matrix multiplications and the Rayleigh–Ritz procedure as a result of putting too many eigenvalues into an interval. In general, we observe that the overall computation becomes less efficient, if the cost of triangular substitutions far exceeds that associated with factorization. As we pointed out in Section 2.3, the optimal k often makes the time spent in factorization and triangular substitutions roughly equal. For this test problem, as we increase l and reduce k and q for each interval accordingly, the cost of triangular substitutions decreases by a much larger margin than the increase in the factorization cost. This results in a reduction in total wall clock time. It is interesting to observe that when $p = 64$ and $l = 8$, which yields the best performance, the amount of time spent in factorization and triangular substitutions is comparable.

In Table 8, we report the best combinations of q and l for a given total core count for all test problems. The column labeled by $\min_q$ gives the minimal number of cores required to solve the problem due to memory constraints, and the column labeled by t_{wall} gives the total wall clock time used to compute 10% lowest eigenvalues of these test problems. Note that for C60, the best q is larger than the $\min_q$. When 16 cores are used to solve the problem, each core is assigned to a different pole, as a result, the factorization time dominates the calculation even though each interval has fewer eigenvalues, hence requires fewer triangular substitutions. The decrease in the time for triangular substitutions is not enough to offset the increase in factorization time. One observation that can be made based on Table 8 is that the minimum q for 3D problems is larger than that associated with 2D problems with a similar matrix size. In both cases, the optimal combination is obtained when the time spent in factorization and triangular substitutions are about the same.

4.2.2. Strong scaling

We observe from above that there is a tradeoff between using more cores to factor $A - z_i I$ (which results in having fewer intervals with more eigenvalues to compute in each interval) and splitting the spectrum into more intervals, each with fewer eigenvalues to compute, when the total number of cores is fixed. The optimal choice of q and l combination will depend on the total number of cores (p) available. Therefore, the strong scaling of MCISPM depends on how q and l are chosen.

Table 9 shows that if we simply increase l and reduce the number of eigenvalues k within each interval as the total number of cores increases, the wall clock time may not decrease at the same rate because the factorization time may start to dominate the computation. Likewise, simply increasing q without changing l may not be the right solution either because the performance of a triangular substitution benefits less from adding more cores. Hence, the cost of triangular substitutions may begin to dominate the computation as q increases.

It follows from the model established in Section 2.3 that a reasonable strategy for selecting q and k is to increase q without decreasing k when the cost of factorization is relatively high. As t_f is reduced significantly or when factorization cannot benefit much from an increase in q per interval, it then becomes more effective to increase the number of intervals l and reduce k per interval accordingly.

Table 10 shows that by increasing q from 256 to 512 as the total number of cores p is doubled from 8192 to 16,384, we can reduce t_f by a half and t_s by 40%. This leads to a significant reduction of the total wall clock time for computing 10% of the lowest eigenvalues of Graphene2048. At that point, the time spent in factorization becomes significantly lower than that spent in triangular substitutions. Thus, when the total number of cores is doubled further to $p = 131,072$, it is more effective to increase the number of intervals and reduce the number of eigenvalues within each interval. As a result, the time for triangular substitutions (t_s = 9 s) is significantly less than the factorization time ($t_f = 33$ seconds). If p is to be doubled again, we should then increase p per interval to reduce the factorization time and, consequently, the total wall clock time significantly.

4.2.3. Weak scaling

We observe from (3) that the optimal number of eigenvalues within each interval depends on the size of the problem as well as the relative cost and efficiency of the factorization and triangular substitution when c_n/c_p is held constant. When

Table 8
The optimal choice of (q, l) for a fixed number of cores (p) when MCISPM is used to solve test problems listed in Table 3.

Problem	p	$q \times l$	$\min_q$	k	t_f	t_s	t_{mm+rr}	t_{other}	t_{wall}
Graphene128	128	16×8	16	64	3.4	1.7	0.2	0.7	6
Graphene512	512	64×8	64	256	45	67	1.7	8	121
Graphene2048	8192	256×32	256	256	73	100	8	27	208
C60	512	32×16	16	110	27	20	0.2	0.8	48
SiO	2048	256×16	256	208	47	54	1.5	5	108
GaAsH6	16,384	512×32	512	192	110	117	3	8	238

Table 9
The scaling of wall clock time for computing $n_\lambda = 8192$ lowest eigenvalues of Graphene2048 with respect to the total number of cores p used in the computation. We divide the desired part of the spectrum into l intervals, and use $q = 256$ cores to compute $k = 8192/l$ eigenvalues within each interval.

p	$q \times l$	k	t_f	t_s	t_{mm+rr}	t_{other}	t_{wall}
8192	256×32	256	73	100	8	27	208
16,384	256×64	128	73	50	1.6	18	143
32,768	256×128	64	73	25	0.6	5	104
65,536	256×256	32	73	12	0.2	3	89
131,072	256×512	16	73	6	0.1	5	85
262,144	256×1024	8	73	3	0.02	6	82

Table 10
The scaling of wall clock time for computing 8192 lowest eigenvalues of Graphene2048 with an optimal choice of q and l.

p	$q \times l$	k	t_f	t_s	t_{mm+rr}	t_{other}	t_{wall}
8192	256×32	256	73	100	8	27	208
16,384	512×32	256	37	69	5.7	23	136
32,768	512×64	128	37	36	1.6	14	89
131,072	512×256	32	37	9	0.2	2	48
262,144	1024×256	16	26	9	0.1	2	38

η_f, η_s, c_f and c_s do not vary much as n and p are increased, the optimal k should increase with respect to n, which is consistent with what we observe in our experiment.

In Table 11, we report the best wall clock time we observed when Graphene problems of increasing sizes are solved by an increasing number of cores. In particular, when the matrix size is increased by a factor of 4, we increase the total number of cores by a factor of 4 also. The best wall clock time measurements are obtained by increasing both k and q by a factor of two when both n and p are quadrupled.

The increase of wall clock time by a factor of 3.8 reported in Table 11 when the matrix size is increased from $n = 20,480$ to $n = 81,920$ is well within the range predicted by our simple model. Our prediction is based on the analysis presented in Section 2.3, which estimates the increase to be by a factor of

$$\frac{\frac{4^{3/2}}{2^{\eta}_{f}} t_f + \frac{4^{\alpha_s}}{2^{\eta_s - 1}} t_s}{t_f + t_s}, \qquad (7)$$

where $t_f = 27$ and $t_s = 24$ represent the cost of factorization and triangular substitutions before n and p are increased. If we use $\eta_f = \eta_s = 1/2$, (7) is approximately 5.6.

However, when the matrix size is increased from $n = 5,120$ to $n = 20,480$, the increase of wall clock time by a factor of 16 is large than we expected. We believe this kind of deviation from our predicted increase of a factor of 5.6 is due to the relative small size of the problem. When the problem size is too small, the prefactors of the estimated cost of factorization and triangular substitutions play a more prominent role, but they were neglected in our previous analysis. This can be see from the factorization time alone, which increases by a factor of more than 15, whereas the standard complexity analysis predicts an increase by a factor of 4 only. Such increase is directly related to the rapid increase in the number of nonzeros in the lower triangular factor, which exceeds a factor of 10 when the number of atoms in the Graphene is increased from 128 to 512.

Our main interest in this paper is on larger problems that are solved with a large number of cores. In this regime, it appears that the MCISPM has a good weak scaling property.

4.3. MSIL vs. MCISPM

Table 12 gives the best total wall-clock times achieved using MCISPM and MSIL to solve our test problems. The comparison between the best wall-clock times for the two methods reveals that MSIL is much faster than MCISPM for these problems.

Table 11
Weak scaling on Graphene problems.

Problem	n	p	t_{wall}	t_f	t_s	k	q	l
Graphene128	5,120	512	3.6	1.7	1.2	64	64	8
Graphene512	20,480	2048	55	27	24	128	128	16
Graphene2048	89,120	8192	208	65	104	256	256	32

Table 12
Comparison of the best wall-clock times achieved using MCISPM and MSIL to solve the problems listed in Table 3.

Problem	p	MCISPM			MSIL		
		$q \times l$	k	t_{wall}	$q \times l$	k	t_{wall}
Graphene512	512	64×8	256	125	25×20	82	16.4
Graphene2048	8192	256×32	256	208	256×32	32	28.2
C60	512	32×16	110	46	51×10	32	9.5
SiO	2048	256×16	208	98	64×32	52	43.1
GaAsH6	16,384	512×32	192	198	64×256	100	54

5. Concluding remarks

We investigated two parallel methods for computing a relatively large number of eigenpairs of a large sparse matrix. Both of these two methods are based on the general technique of dividing the spectrum of interest into several intervals and computing eigenvalues within each interval simultaneously. One method uses the shift–invert Lanczos procedure with implicitly restarts to compute eigenvalues within each interval. The other method constructs an approximate spectral projection operator P through a contour integral formulation and applies a few subspace iterations to obtain the desired eigenvalues. Both methods require solving a number of linear systems of equations. We assume in this paper that these equations can be solved by sparse direct methods such as the one implemented in the MUMPS software package [1].

We also assume that a large number of processors or computational cores are available to solve the eigenvalue problem. A practical question we examined is how we should partition the spectrum in a way to make optimal use of a vast amount of computational resource. We established a simple model to study the trade-off between sparse factorization and triangular substitutions and how this trade-off may affect the wall-clock time of the overall computation. Spectrum partitioning strategies based on the analysis of this model appears to yield good performance in our computational experiments. Our computational experiments indicate that MSIL gives much better performance results for the problems studied in this paper.

We remark that the optimal spectrum partition strategy and the actual performance of the eigensolver depends on the performance and the scalability of the sparse direct method. Although we only experimented with the use of MUMPS, many of the observations and trends we discussed are applicable to other solvers.

Acknowledgments

Support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy Office of Advanced Scientific Computing Research under the contract DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

Appendix

In the model presented in Section 2.3, we only take into account the cost for sparse factorization and triangular substitutions, and neglect the cost for the diagonalization procedure inside each interval. In this section we justify this simplification. The cost of the diagonalization procedure consists of two parts: the matrix–matrix multiplication procedure (MM) to form the projected problem and to obtain the orthogonal eigenvectors (orthogonalization), and the Rayleigh–Ritz procedure (RR) to solve the projected problem. The cost for MM scales as $c_{mm}nk^2$ and the cost for RR scales as $c_{rr}k^3$. Using notation similar to that used in Section 2.3, we denote by η_{mm} the parallel efficiency for the MM procedure. We also note that in many existing implementations of the spectrum slicing algorithm including our own, the RR procedure is not yet parallelized. Therefore a complete description of the model for the wall clock time can be written as

$$W^{\text{tot}}(k) = \frac{c_f n^{\alpha_f}}{\left(\frac{p}{nev/k}\right)^{\eta_f}} + \frac{c_s n^{\alpha_s} k}{\left(\frac{p}{nev/k}\right)^{\eta_s}} + \frac{c_{mm} nk^2}{\left(\frac{p}{nev/k}\right)^{\eta_{mm}}} + c_{rr}k^3. \tag{8}$$

Since the goal of the spectrum slicing method is to limit the diagonalization cost when k is large, we require the wall clock time of the MM and RR procedure to be much less than the wall clock time of either the factorization and the triangular solve process. This gives rise to the following set of conditions

$$\frac{c_{mm} nk^2}{\left(\frac{p}{nev/k}\right)^{\eta_{mm}}} \ll \frac{c_f n^{\alpha_f}}{\left(\frac{p}{nev/k}\right)^{\eta_f}}, \tag{9}$$

$$\frac{c_{mm} n k^2}{\left(\frac{p}{nev/k}\right)^{\eta_{mm}}} \ll \frac{c_s n^{\alpha_s} k}{\left(\frac{p}{nev/k}\right)^{\eta_s}}, \tag{10}$$

$$c_{rr} k^3 \ll \frac{c_f n^{\alpha_f}}{\left(\frac{p}{nev/k}\right)^{\eta_f}}, \tag{11}$$

$$c_{rr} k^3 \ll \frac{c_s n^{\alpha_s} k}{\left(\frac{p}{nev/k}\right)^{\eta_s}}, \tag{12}$$

which can be further simplified to yield

$$k \ll \left(\frac{c_p}{c_n}\right)^{\frac{\eta_{mm}-\eta_f}{2-\eta_{mm}+\eta_f}} \left(\frac{c_f}{c_{mm}}\right)^{\frac{1}{2-\eta_{mm}+\eta_f}} n^{\frac{\alpha_f-1}{2-\eta_{mm}+\eta_f}}, \tag{13}$$

$$k \ll \left(\frac{c_p}{c_n}\right)^{\frac{\eta_{mm}-\eta_s}{1-\eta_{mm}+\eta_s}} \left(\frac{c_s}{c_{mm}}\right)^{\frac{1}{1-\eta_{mm}+\eta_s}} n^{\frac{\alpha_s-1}{1-\eta_{mm}+\eta_s}}, \tag{14}$$

$$k \ll \left(\frac{c_p}{c_n}\right)^{-\frac{\eta_f}{3+\eta_f}} \left(\frac{c_f}{c_{rr}}\right)^{\frac{1}{3+\eta_f}} n^{\frac{\alpha_f}{3+\eta_f}}, \tag{15}$$

$$k \ll \left(\frac{c_p}{c_n}\right)^{-\frac{\eta_s}{2+\eta_s}} \left(\frac{c_s}{c_{rr}}\right)^{\frac{1}{2+\eta_s}} n^{\frac{\alpha_s}{2+\eta_s}}. \tag{16}$$

Let us focus on the scaling with respect to n.We use the values of α_f, α_s in Table 2, and assume that $\eta_{mm} = \eta_f = \eta_s$.In order to satisfy all conditions set in Eq.(13) to Eq.(16), we find that $k \sim \mathcal{O}(1)$ for $d = 1, k \sim \mathcal{O}(1)$ for $d = 2$, and $k \sim \mathcal{O}(n^{1/3})$ for $d = 3$.Such scaling is more strict than those provided in Table 2, which agrees with our intuitive understanding that if k is too large, the cost for the orthogonalization and the Rayleigh–Ritz calculations will dominate.However, it is worth noting that usually the MM process has much smaller preconstant than that associated with a factorization or a triangular solve ($c_{mm} \ll c_f, c_s$), and the BLAS3 routine in the MM process is easier to parallelize than the factorization or triangular solve ($\eta_{mm} > \eta_f, \eta_s$).Therefore the bounds in Eqs.(13) and (14) can be excessively large for practical choice of n as in the numerical results section.If we neglect the condition in Eqs.(13) and (14) and only require Eqs.(15) and (16) to be satisfied, and assume $\eta_f = \eta_s = \eta$, we obtain $k \sim \mathcal{O}\left(n^{\frac{1}{3+\eta}}\right)$ for $d = 1$, $k \sim \mathcal{O}\left(n^{\frac{1}{2+\eta}}\right)$ for $d = 2$, and $k \sim \mathcal{O}\left(n^{\frac{4}{6+3\eta}}\right)$ for $d = 3$.Such scaling is only slightly more stringent than that shown in Table 2, and it agrees with the scaling in Table 2 when $\eta = 0$.This is because we assume the RR process is solved with a sequential routine.Furthermore, taking into account that the RR process has much smaller preconstant than factorization or triangular solve($c_{rr} \ll c_f, c_s$), we conclude that it is reasonable to simplify the discussion and focus on the cost for factorization and triangular solve as in Section 2.3 for practical choice of k and n.

References

[1] P. Amestoy, I. Duff, J.-Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (2001) 15–41.
[2] Andrzej Bajer, Mode-based frequency response analysis with frequency-dependent material properties, ASME Conf. Proc. 2008 (48396) (2008) 331–334.
[3] C.G. Baker, U.L. Hetmaniuk, R.B. Lehoucq, H.K. Thornquist, Anasazi software for the numerical solution of large-scale eigenvalue problems, ACM Trans. Math. Softw. 36 (3) (2009) 13:1–13:23.
[4] L.S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R.C. Whaley, ScaLAPACK User's Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
[5] Nicolas Bock, Matt Challacombe, An optimized sparse approximate matrix multiply, CoRR, abs/1203.1692, 2012.
[6] D.R. Bowler, T. Miyazaki, M.J. Gillan, Parallel sparse matrix multiplication for linear scaling electronic structure calculations, Comput. Phys. Commun. 137 (2) (2001) 255–273.
[7] F. Chatelin, Spectral Approximation of Linear Operators, Academic Press, New York, 1983.
[8] B. Fischer, Polynomial Based Iteration Methods for Symmetric Linear Systems, Wiley Teubner, Chichester, New York, 1996.
[9] R. Freund, N. Nachtigal, QMR: A quasi-minimal residual method for non-hermitian linear systems, Numer. Math. 60 (1991) 315–339.
[10] A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal. 10 (1973) 345–363.
[11] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1998) 359–392.
[12] A. Knyazev, M. Argentati, I. Lashuk, E. Ovtchinnikov, Block locally optimal preconditioned eigenvalue xolvers (BLOPEX) in Hypre and PETSc, SIAM J. Sci. Comput. 29 (5) (2007) 2224–2239.
[13] L. Krämer, E.D. Napoli, M. Galgon, B. Lang, P. Bientinesi, Dissecting the FEAST algorithm for generalized eigenproblems, J. Comput. Appl. Math. 244 (2013).
[14] L. Kronik, A. Makmal, M.L. Tiago, M.M.G. Alemany, M. Jain, X. Huang, Y. Saad, J.R. Chelikowsky, PARSEC–the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nano-structures, Phys. Status Solidi B 243 (2006) 1063–1079.
[15] R. Lehoucq, D. Sorensen, Implicitly restarted Lanczos method (Section 4.5), in: Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, SIAM, Philadelphia, 2000, pp. 67–81.

[16] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK Users' Guide – Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM, Philadelphia, PA, 1999.

[17] A.R. Levin, D. Zhang, E. Polizzi, FEAST fundamental framework for electronic structure calculations: reformulation and solution of the muffin-tin problem, Comput. Phys. Commun. 183 (11) (2012).

[18] L. Lin, J. Lu, L. Ying, W. E., Adaptive local basis set for Kohn–Sham density functional theory in a discontinuous Galerkin framework I: total energy calculation, J. Comput. Phys. 231 (2012) 2140–2154.

[19] L. Lin, C. Yang, Y. Saad, Approximating spectral densities of large matrices, http://arxiv.org/abs/1308.5467.

[20] K.J. Maschhoff, D.C. Sorensen, PARPACK: an efficient portable large scale eigenvalue package for distributed memory parallel architectures, in: J. Wasniewski, J. Dongarra, K. Madsen, D. Olesen (Eds.), Applied Parallel Computing in Industrial Problems and Optimization, Lecture Notes in Computer Science, vol. 1184, Springer-Verlag, Berlin, 1996.

[21] C.C. Paige, M.A. Saunders, Solutions of sparse indefinite systems of linear equations, SIAM J. Numer. Anal. 12 (1975) 617–629.

[22] E. Polizzi, Density-matrix-based algorithms for solving eigenvalue problems, Phys. Rev. B 79 (2009) 115112.

[23] A. Ruhe, Rational Krylov sequence methods for eigenvalue computation, Linear Algebra Appl. 58 (1984).

[24] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Comput. 7 (1986) 856–869.

[25] Y. Saad, A. Stathopoulos, J. Chelikowsky, K. Wu, Solution of large eigenvalue problems in electronic structure calculations, BIT 36 (1996) 563–578.

[26] T. Sakurai, H. Sugiura, A projection method for generalized eigenvalue problems, J. Comput. Appl. Math. 159 (2003) 119–128.

[27] Tetsuya Sakurai, Yoshihisa Kodaki, Hiroto Tadano, Daisuke Takahashi, Mitsuhisa Sato, Umpei Nagashima, A parallel method for large sparse generalized eigenvalue problems usin g a gridrpc system, Future Gen. Comput. Syst. 24 (6) (2008) 613–619.

[28] G. Schofield, J.R. Chelikowsky, Y. Saad, A spectrum slicing method for the Kohn–Sham problem, Comput. Phys. Commun. 183 (2012).

[29] D.C. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, SIAM J. Matrix Anal. Appl. 13 (1) (1992) 357–385.

[30] G.W. Stewart, Afternotes goes to graduate school: Lecture on advanced numerical analysis, SIAM, Philadelphia, 1998.

[31] Andreas Stathopoulos, James R. McCombs, PRIMME: preconditioned iterative multimethod eigensolver – methods and software description, ACM Trans. Math. Softw. 37 (2) (2010) 21:1–21:30.

[32] J.J. Sylvester, A demonstration of the theorem that every homogeneous quadratic polynomial is reducible by real orthogonal substitutions to the form of a sum of positive and negative squares, Philos. Mag. 23 (1852) 138–142.

[33] H.A. Van der Vorst, Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (2) (1992) 631–644.

[34] Zaiwen Wen, Chao Yang, Xin Liu, Yin Zhang, Trace-penalty minimization for large-scale eigenspace computation, Optimization, 2013 (Online).

[35] I. Yamazaki, H. Tadano, T. Sakurai, T. Ikegami, Performance comparison of parallel eigensolvers based on a contour integral method and a lanczos method, Parallel Comput. 39 (6–7) (2013) 280–290.

[36] H. Zhang, B. Smith, M. Sternberg, P. Zapol, SIPs: shift-and-invert parallel spectral transformations, ACM Trans. Math. Softw. 33 (2) (2007).