SIPs: Shift-and-Invert Parallel Spectral Transformations

HONG ZHANG
Illinois Institute of Technology
and
BARRY SMITH, MICHAEL STERNBERG, and PETER ZAPOL
Argonne National Laboratory

SIPs is a new efficient and robust software package implementing multiple shift-and-invert spectral transformations on parallel computers. Built on top of SLEPc and PETSc, it can compute very large numbers of eigenpairs for sparse symmetric generalized eigenvalue problems. The development of SIPs is motivated by applications in nanoscale materials modeling, in which the growing size of the matrices and the pathological eigenvalue distribution challenge the efficiency and robustness of the solver. In this article, we present a parallel eigenvalue algorithm based on distributed spectrum slicing. We describe the object-oriented design and implementation techniques in SIPs, and demonstrate its numerical performance on an advanced distributed computer.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Eigenvalues and eigenvectors (direct and iterative methods), sparse, structured, and very large systems*; G.4 [**Mathematical Software**]—*Parallel and vector implementations, efficiency, reliability and robustness*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Parallelism, sparse eigenvalue computation, spectral transformation

ACM Reference Format:

Zhang, H., Smith, B., Sternberg, M., and Zapol, P. 2007. SIPs: Shift-and-invert parallel spectral transformations. ACM Trans. Math. Softw., 33, 2, Article 9 (June 2007), 19 pages. DOI = 10.1145/1236463.1236464 http://doi.acm.org/10.1145/1236463.1236464

This work was supported by the Mathematical, Information, Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, and by the Basic Energy Sciences—Materials Science, bothe of the Office of Science, U.S. Department of Energy, under Contract W31 109 ENG 38.
Author's address: H. Zhang, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4844; email: hzhang@mcs.anl.gov.

1. INTRODUCTION

This work is motivated by increasing computational demands in nanoscale materials modeling. In this field, electronic structure methods are used to minimize the configurational energy and to calculate other physical properties of a system of atoms. The mathematical core of these calculations is a generalized eigenvalue problem of the form:

$$Ax_i = \lambda_i B x_i\,, \qquad i = i_{\min}, \ldots, i_{\max}\,, \tag{1}$$

where A and B are $n \times n$ real symmetric matrices, B is positive definite, and $i_{\min}$ and $i_{\max}$ is the index range of requested eigenpairs. In our discussion and numerical tests, the eigenvectors are normalized as $x_i^T B x_i = 1$.

We use the density-functional-based tight-binding (DFTB) method [Elstner et al. 1998] for materials modeling applications. The eigenproblems posed by DFTB as studied in this article are distinguished by several features:

(1) The matrix pencil (A, B) is large and sparse. Its size n is proportional to the number of atoms in the model. With an ultimate goal of simulating 50,000 atoms, the matrices are expected to be as large as $n = 200{,}000$.

(2) A large number of eigensolutions are requested; for example, 60% eigenvalues and the associated eigenvectors are wanted based on current DFTB applications.

(3) The spectrum is pathologically difficult. It has clusters of hundreds of tightly packed eigenvalues and very poor average relative eigenvalue separation:

$$\text{ave}\left(\frac{\lambda_{i+1} - \lambda_i}{\lambda_n - \lambda_1}\right) = O(n^{-1}) \quad \text{(typically} \approx 10^{-5} \text{ in our study).}$$

Even worse, some clusters are adjacent to gaps that have lengths far larger than the average eigenvalue separation:

$$\lambda_{j+1} - \lambda_j \gg \text{ave}\,(\lambda_{i+1} - \lambda_i)\,.$$

(4) The global coupling of the nonzero elements in (A, B) gives rise to not-very-sparse, or even to dense, matrix factorizations. For example, the matrix factors of matrices with $n = 16{,}000$ under study, have sparse densities ranging from 7% to 50%. By conventional sparse matrix standards, 7% is still extremely dense.

(5) The simulation requires a significant number of iterations (possibly 1000s) of Equation (1) with closely related matrices A and B.[1]

In this article, we consider the eigenvalue problem Equation (1) with the features described above. It should be noted that these features place our problem in a class separate from well-studied sparse eigenproblems arising in finite element methods. Those problems are larger by one or two orders of magnitude, are more sparse, and the number of eigensolutions requested is typically lower, in the range of 1 to 1000. Our discussion focuses on computing eigenvalues

[1]The matrix A contains a perturbation dependent on the solution vectors x, so that a fixpoint solution is sought for a given position of atoms and fixed B. In an outer iteration, both matrices depend on the positions of atoms being optimized or time-stepped.

ordered increasingly from $i_{\min}$ to $i_{\max}$, and their associated eigenvectors. The discussion can easily be applied to other general forms, for example, computing all eigenvalues in $[a, b]$ and their eigenvectors.

Existing eigensolvers are customarily developed based on two types of matrix storage: dense and sparse storage. They are typically solved by using *direct methods* and *iterative methods*, respectively.

Direct methods compute all or almost all, eigensolutions of dense matrices. They exhibit $O(n^3)$ time and $O(n^2)$ memory complexity. Library software based on direct methods is provided in the LAPACK [Anderson et al. 1999] and ScaLAPACK [Blackford et al. 1997] packages. Both have been used for DFTB, but time and memory scaling prevents advancing to the larger nanoscale systems of current interest. In particular, our experience has shown that extensive communication requirements in ScaLAPACK cause scaling problems on workstation clusters with commodity networking hardware.

Iterative methods such as Lanczos [1950] and Jacobi-Davidson [Sleijpen and van der Vorst 1996] are widely used for extracting a few extreme eigensolutions of sparse matrices. Their time and space complexity is bound above by the complexity of direct methods, yet they are usually much more efficient when the matrices involved are indeed sparse. The available software includes ARPACK [Lehoucq et al. 1998] and several others [Hernandez et al. 2005; Marques 1995; Wu and Simon 1997]. When interior eigenvalues are requested, a practical approach is to replace (A, B) by a shift-and-invert operation [Ericsson and Ruhe 1980]:

$$Ax = \lambda Bx \iff (A - \sigma B)x = (\lambda - \sigma)Bx, \quad \sigma \neq \lambda.$$

Setting

$$C = B(A - \sigma B)^{-1} \text{ and } y = Bx,$$

we get an equivalent eigenvalue problem:

$$Cy = \tilde{\lambda} y, \qquad \tilde{\lambda} = \frac{1}{\lambda - \sigma}. \tag{2}$$

Employing Lanczos iterations to (2) leads to eigensolutions of the original equation (1) that are close to the shift σ.

DFTB and related methods for materials modeling require a large fraction of eigensolutions—typically the lower 50% of the spectrum—with accurate accounting for all requested eigenpairs and reliable orthogonalization in degenerate or nearly degenerate subspaces. Without significant customization, none of the available iterative eigenvalue packages are able to provide sufficient efficiency and robustness crucial to the modeling process. In this article we propose SIPs, a new software package implementing shift-and-invert parallel spectral transformations on top of the existing iterative eigensolver. We introduce an eigenvalue algorithm in Section 2 and describe its implementation in Section 3. The implementation includes an object-oriented software design for performance, portability, and reusability and techniques that build efficiency and robustness into the proposed eigensolver. In Section 4 we present numerical experiments using SIPs and compare our results with those from ScaLAPACK.

We report on three systems of nanoscale materials with diverse characteristics, focusing on the distinct sparse densities of the matrix factorization, a dominating factor for the performance of SIPs. In Section 5 we summarize our conclusions and the impact of this work.

2. MULTIPLE SHIFT-AND-INVERT PARALLEL EIGENVALUE ALGORITHM

Based on the idea of distributed spectrum slicing, we propose concurrently using Lanczos iterations with multiple shift-and-invert spectral transformations on a distributed eigenvalue spectrum. This general approach has been studied by others, for example, Bostic and Fulton [1987], Teranishi et al. [2003] and Komzsik [2003], but our work is not limited to the algorithmic investigation. We present a novel algorithm, based on which the software SIPs is developed and used for solving applications in nanoscale modeling.

The Multiple Shift-and-Invert Parallel Eigenvalue Algorithm is described in Figure 1. Initially, the user provides a requested eigenvalue interval. We divide it into overlapping subintervals and assign each to a process. A process starts from a shift in the middle of its interval and picks new shifts at the left and right sides of the current shift. The bounds of processed subintervals are exchanged between neighboring processes during the computation of local eigensolutions. Using this information, each process adjusts its assigned subinterval, which redistributes the initially assigned subintervals dynamically and balances the parallel workload. Within each process, multiple shifts are selected one after the other for computing all eigensolutions in the assigned subinterval.

For a single shift, the shift-and-invert spectral transformation enhances convergence to the eigenpairs close to the shift. A well-chosen shift allows us to compute tens to hundreds of eigensolutions with one to several Lanczos runs. When a particular shift σ is chosen, we apply a matrix factorization:

$$A - \sigma B = LDL^T, \tag{3}$$

then feed it into a Lanczos iteration for generating a Krylov subspace. As a byproduct, Equation (3) also provides $\nu(A - \sigma B)$, the number of eigenvalues of (A, B) that are smaller than σ. For simplicity, we denote $\nu(\sigma) = \nu(A - \sigma B)$ and refer to it as *matrix inertia*. This number is used for validating the eigensolutions computed through the shift σ. Each shift incurs an expensive matrix factorization (3), and two further shifts σ_i, σ_j are needed for checking the validity of eigensolutions in the interval (σ_i, σ_j). To make our solver efficient and robust, we dynamically select a set of shifts that produces multiple sets of eigensolutions at minimum redundancy, and we reuse these shifts to validate the eigensolutions.

3. IMPLEMENTATION OF THE ALGORITHM

We now discuss the implementation of the Multiple Shift-and-Invert Parallel Eigenvalue Algorithm. We start from the software design, then explain the techniques for dynamically selecting shifts, bookkeeping locally computed eigensolutions, maintaining parallel job balance, ensuring global accuracy of

```
Input:   matrix pencil $(A, B)$;
         index range $i_{\min}$, $i_{\max}$ and/or requested eigenvalue spectrum $(\lambda_{\min}, \lambda_{\max})$;
         eigenvalue approximation $\{\hat{\lambda}_i\}$ (optional).
Output:  k-th process has its local list of eigenpairs indexed from
         $i_k$ to $i_{k+1} - 1$ $(i_0 = i_{\min}, i_{\mathrm{np}} = i_{\max})$.

Process k (k=0,...,np-1):
(1) Initialize:
    (a) set an initial shift $\sigma_0^{(k)}$; compute matrix inertia $\nu(\sigma_0^{(k)})$;
    (b) get initial local assigned spectrum $(\lambda_{\min}^{(k)}, \lambda_{\max}^{(k)})$ and associated index bound $i_{\min}^{(k)}$, $i_{\max}^{(k)}$;
    (c) initialize local computed spectrum $[\sigma_{\min}^{(k)}, \sigma_{\max}^{(k)}]=\{\sigma_0^{(k)}\}$.
(2) Do until the local list of eigensolutions is full:
    (a) compute eigenpairs that are close to the selected shift $\sigma_i^{(k)}$ by the shift-and-invert Lanczos iteration;
    (b) check validity of the computed eigenvalues against the eigenvalues that are already on
        the local list; add the new and desired eigensolutions to the list;
    (c) compute new possible shifts at left and right side of $\sigma_i^{(k)}$;
    (d) pick next shift $\sigma_{i+1}^{(k)}$ and compute $\nu(\sigma_{i+1}^{(k)})$; if $\sigma_{i+1}^{(k)} < \sigma_{\min}^{(k)}$ or $\sigma_{i+1}^{(k)} > \sigma_{\max}^{(k)}$, update
        computed spectrum $[\sigma_{\min}^{(k)}, \sigma_{\max}^{(k)}]$ and send $\sigma_{i+1}^{(k)}$, $\nu(\sigma_{i+1}^{(k)})$ to a neighboring process;
    (e) receive messages from neighboring processes and update its assigned spectrum $(\lambda_{\min}^{(k)}$,
        $\lambda_{\max}^{(k)})$ and index bounds $i_{\min}^{(k)}$, $i_{\max}^{(k)}$.
(3) Final check:
    (a) exchange $\lambda_{\min}^{(k)}$, $\lambda_{\max}^{(k)}$ and $i_{\min}^{(k)}$, $i_{\max}^{(k)}$ with neighboring processes;
    (b) delete duplicate eigensolutions.
```

Fig. 1. Multiple shift-and-invert parallel eigenvalue algorithm.

the eigensolutions, and organizing subgroups of MPI communicators for processing large-scale matrix operations.

3.1 Software Structure Design

The design objective of our eigensolver is to deliver high performance with limited effort for development and maintenance and to enable portability and reusability. Our design choices are guided by the following three major components of the algorithm:

(1) Sequential and parallel sparse matrix operations, for example, matrix-vector multiplication, matrix factorization, and triangular solve. These fundamental operations would dominate the performance of the eigensolver.
(2) Lanczos iterations with a single shift-and-invert spectral transformation.
(3) Sequential selection of multiple shifts, parallel distribution of the eigenvalue spectrum, and bookkeeping computed eigensolutions.

After examining software packages that implement (1) and (2), we choose PETSc [Balay et al. 2006] and its add-on package, SLEPc [Hernandez et al. 2005]. PETSc provides us with sequential and parallel data structures and basic operations that implement (1), while SLEPc offers built-in support for spectral transformation and Lanczos eigensolvers required by (2). The first two items listed in (3) have been studied by others. For instance, Grimes et al. [1994] and

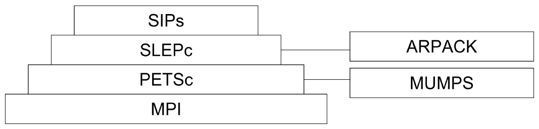

Fig. 2. Software structure.

Marques [1995] developed block Lanczos algorithms that include sophisticate shift selection strategies. The sequential software BCSLIB-EXT that implements the proposed algorithm [Grimes et al. 1994] and BLZPACK were tested on our DFTB eigenproblems and found to be unable to compute all the solutions when the requested eigenvalue spectrum has large disparities. Parallel Lanczos algorithms that have each process compute eigensolutions from different shift-and-invert spectral transformations were proposed by Bostic and Fulton [1987], Teranishi et al. [2003], and Komzsik [2003], along with numerical examples that demonstrate the feasibility of the idea of concurrent spectrum slicing. However, to our knowledge, no work has given integrated consideration of all the practical issues listed in (3), and there is no software package equipped with the robustness and efficiency required by the DFTB eigenvalue problems.

Although PETSc and SLEPc provide adequate data and solver objects for implementing (1) and (2), state-of-the-art special-purpose software packages exist that either outperform, or are more reliable than PETSc and SLEPc on specific tasks. Through interfaces provided by PETSc and SLEPc, we can easily make use of these desirable packages. Because the direct sparse matrix factorization and triangular solve dominate computational time, through a PETSc interface we link to the MUltifrontal Massively Parallel sparse direct Solver (MUMPS) [Amestoy et al. 2000, 2001] for such demanding computations. In addition, because of the pathological eigenvalue distribution, very few iterative eigenvalue packages are able to deliver reliable solutions for our DFTB models. Among them, we pick ARPACK, whose interface is provided by SLEPc.

To recapitulate, our task for the proposed algorithm (Figure 1) is to implement the component (3) as a new package on top of SLEPc and PETSc. We name this new package *S*hift-and-*I*nvert *P*arallel Spectral Transformation*s* (*SIPs*). Figure 2 illustrates our overall software design for implementing the Multiple Shift-and-Invert Parallel Eigenvalue Algorithm described in Section 2.

Through the interfaces of PETSc and SLEPc, SIPs easily uses the external eigenvalue package ARPACK and the parallel sparse direct solver MUMPS. The packages can be upgraded or replaced without programming changes to SIPs. SIPs itself implements the following major tasks:

(1) Select shifts;
(2) Bookkeep and validate eigensolutions;
(3) Balance parallel workload;
(4) Ensure global orthogonality of eigenvectors;
(5) Organize subgroups of MPI communicators.

We discuss technical details of these tasks in the next five subsections.

3.2 Select Shifts

In the beginning, an initial shift σ_0 is chosen at the midpoint of the subinterval $(\lambda^{(k)}_{\min}, \lambda^{(k)}_{\max})$, which is assigned to the kth process. With it, a set of eigenvalues λ_1, $\lambda_2, \ldots, \lambda_{nev}$ (in increasing order) close to σ_0 is computed. To determine the next two possible shifts, extending from the left and right of σ_0, we adopt a strategy similar to that proposed by Grimes et al. [1994]; that is, assuming the *radii of convergence* for neighboring shifts are about the same, we can select the new shift σ_1 extending from the right of σ_0 as:

$$\sigma_1 = \sigma_0 + 2\delta, \qquad \delta = \max(|\lambda_1 - \sigma_0|, |\lambda_{nev} - \sigma_0|). \tag{4}$$

Our actual code uses a slightly more conservative estimate than the one above.

We use a queue to store all the selected shifts waiting for processing. In this article, a shift is called *active* when it is taken from the head of this queue and currently participates in the eigenvalue computation, *pending* when it waits on the queue, and *used* after its Lanczos iterations are finished.

When a new shift σ_{new} is selected from an active shift, it is appended to the queue with the following data:

—σ_{left}, σ_{right}: neighboring active or used shifts (thus their matrix inertias have been computed). We call $(\sigma_{\text{left}}, \sigma_{\text{right}})$ a *pending interval* in which the eigenvalues are either uncomputed, or computed but have not gone through a validity check.

—$\nu(\sigma_{\text{left}})$, $\nu(\sigma_{\text{right}})$: $\nu_{\text{right}} - \nu_{\text{left}}$ is the number of pending eigenvalues located in $(\sigma_{\text{left}}, \sigma_{\text{right}})$.

—*isLext*, *isRext*: information about which side of the active shift σ_{new} is selected or extended from.

For example, if $\sigma_{\text{new}} = \sigma_1$ is selected from the right of an active σ_0; then it has the attached data $\sigma_{\text{left}} = \sigma_0$, $\sigma_{\text{right}} = \lambda_{\max}$, $\nu(\sigma_{\text{left}}) = \nu(\sigma_0)$, $\nu(\sigma_{\text{right}}) = i_{\max}$, *isLext* = *false*, and *isRext* = *true*. Similarly, σ_2 is extended from left of σ_0 and attached with the data about its pending interval $(\lambda_{\min}, \sigma_0)$.

After all eigensolutions are computed from the active σ_i, and the possible new shifts at each side of σ_i are selected and appended to the queue, we take σ_{i+1} from the head of the queue and proceed to the next round of Lanczos iterations until the queue becomes empty.

The shift selection process described here works well in normal situations, namely under the assumption that neighboring shifts have roughly the same radius of convergence. However the DFTB eigenvalue spectrum in general has a very large disparity, and sometimes an eigenvalue cluster is adjacent to a gap that is a thousand times larger than the convergence radius of the cluster, a common phenomenon in industrial applications [Komzsik 2003]. When this occurs, the next shift σ_{i+1} likely falls into the adjacent gap and is closer to the just computed cluster than to uncomputed eigenvalues located at the other side of the gap. Using it, the eigensolver would recompute the just obtained eigenvalue cluster at an extremely high number of Lanczos runs, or reach the maximum number of Lanczos runs without getting any converged eigensolutions.

To deal with this difficulty, we detect the gap and move the shift based on information from the pending interval $(\sigma_{\text{left}}, \sigma_{\text{right}})$. For example, if σ has *isRext* $=$ *true* (extended from the right side of its parent shift) and $\nu(\sigma) - \nu(\sigma_{\text{left}}) \approx 0$, that is there are none or few eigenvalues in $(\sigma_{\text{left}}, \sigma)$, then σ likely falls into a gap. In this case, we move σ toward the right side of the pending interval $(\sigma_{\text{left}}, \sigma_{\text{right}})$:

$$\sigma := (1 - \tau)\sigma + \tau\sigma_{\text{right}}, \qquad 0.5 \leq \tau < 1. \tag{5}$$

The move (5) is repeated if a single move is not sufficient. Note that Equation (5) assigns a new value to σ within its pending interval. While the pending interval remains unchanged, the move requires a matrix factorization (3), which should be applied only when necessary.

In DFTB calculations, the eigenvalue problem (1) is solved repeatedly, each with a slightly modified matrix pencil (A, B). The previously computed eigensolutions can be used as approximations to the solutions of the next eigenvalue problem. In SIPs, we use computed eigenvalues or the Rayleigh quotient of computed eigenvectors, denoted as $\{\hat{\lambda}_i\}$, as approximations to the new eigenvalues. When the approximations are available, the new shifts are selected based on this information. For example, the shift σ_1 in Equation (4) now can be selected as:

$$\sigma_1 = \hat{\lambda}_i, \quad i = \nu(\sigma_0) + nev.$$

The eigenvalue approximations $\{\hat{\lambda}_i\}$ also indicate possible gap locations, so the shifts would be chosen effectively away from gaps.

When a shift is very close to an eigenvalue, the matrix factorization (3) is ill-conditioned. It is believed that it gives erroneous Krylov subspaces, and consequently results in inaccurate or incomplete eigensolutions, or that such shifts are useful only for computing isolated clusters of eigenvalues. Remarkably, in our intensive tests, we have never encountered such an incident. For some test problems, for example, the diamond crystal (see Section 4), the shifts frequently fall into eigenvalue clusters, or the initial shifts are intentionally set to the clustered eigenvalues computed from either ScaLAPACK or SIPs. In these cases, SIPs is still able to compute satisfying nearby eigensolutions without behavior distinguishable from that at other shifts.

While a serious investigation and rigorous analysis remains to be conducted, we believe the ARPACK and MUMPS generated nondeficient Krylov subspaces with the chosen parameters. From the rounding-error analysis for the inverse power method [Parlett 1998], most inaccuracies resulting from the ill-conditioned matrix factorization are perhaps in the directions of the computed vectors, which span the Krylov subspaces during the Lanczos iterations. These inaccuracies do not introduce any significant errors in the computed eigensolutions.

A difficulty we did encounter is the case in which clustering eigenvalues located away from the selected shift were computed with eigenvectors missing. This results in inaccurate eigenvalue indices and leads to Case 1 with unmatched indices as described in the next section. These eigensolutions will be recomputed by SIPs using closer shifts, as explained in Section 3.3.

When a shift is identical to an eigenvalue, the matrix operation returns a value indicating the singularity of the factorization. Adopting the industry standard solution [Komzsik 2003], we perturb the shift and repeat the matrix factorization. In other special cases, selected shifts need to be adjusted or dumped when their pending intervals become empty because of changes made before becoming active. In this article, however, we concentrate on major techniques used by SIPs and skip technical description of minor cases.

3.3 Bookkeep and Validate Eigensolutions

We use a structure array in C, named *Local Solution List*, to store eigensolutions computed by a process. Each element in the array represents an eigensolution:

```
typedef struct {
PetscReal  *val,*sval;      /* eigenvalues, eigenvalue singletons */
PetscInt   *smap;           /* maps singletons to eigenvalues */
Vec        *vec;            /* eigenvectors */
PetscInt   *status;         /* one of status: UNCOMPUT, COMPUT, or DONE */
PetscInt   *mult,*smult;/* multiplicity of eigenvalues and singletons */
} EVSOL;
```

The initial length of the list in process k is set as $i_{k+1} - i_k$ (see Figure 1). When a set of eigensolutions is computed through a shift σ, the eigenvalues are ordered as:

$$\lambda_1 \leq \cdots \leq \lambda_i < \sigma < \lambda_{i+1} \leq \cdots \leq \lambda_{\text{nev}}. \tag{6}$$

We use a degeneracy (multiplicity) tolerance, tol_{deg}, to group degenerate or nearly degenerate eigenvalues into *eigenvalue singletons* (groups of numerically degenerate eigenvalues). The maximum number of eigenvalues in a singleton, *smult*, is referred to as the singleton's multiplicity. Accordingly, any two adjacent eigenvalues in a singleton satisfy $0 \leq \lambda_{j+1} - \lambda_j \leq tol_{\text{deg}}$, and the the separation between the eigenvalues in and out of a singleton is larger than tol_{deg}.

Using the matrix inertia $\nu(\sigma)$, we can compute absolute indices for the computed eigenvalues; for example, λ_i is the $\nu(\sigma)$-th eigenvalue of (A, B). These values are checked or compared with the ones already on the Local Solution List. The newly computed eigensolutions assigned to this process are then added onto the list together with relevant data.

Because of the existence of multiple and clustering eigenvalues, some eigensolutions with values between λ_1 and λ_{nev} might not be computed. Without further validating the computed eigensolutions, those put on the list cannot be trusted yet. The validity check is implemented by verifying whether a computed eigenvalue is in a *trusted interval*, by which we mean the number of eigenvalues expected in the interval agrees with the number actually computed [Grimes et al. 1994].

Establishing a trust interval (a, b) requires matrix inertias at a and b, as well as the total number of computed eigenvalues in it. Because matrix factorization, Equation (3), is expensive, we want to limit its invocation to the necessary shift-and-invert Lanczos iterations. The trust intervals are then established by reusing the active and used shifts.

Before any eigensolution is computed, its status on the Local Solution List is initialized as *UNCOMPUT*. When a set of eigensolutions is computed through the initial shift σ_0 and added to the Local Solution List, the status of these solutions is upgraded to *COMPUT*. Assume a new shift σ_1 $(> \sigma_0)$ becomes active. It will generate a new set of eigensolutions in one of the following two cases:

Case 1. An eigenvalue $\lambda^{(1)}$ $(< \sigma_1)$ overlaps a previously computed eigenvalue $\lambda^{(0)}$ $(> \sigma_0)$. Using matrix inertia $\nu(\sigma_0)$ and $\nu(\sigma_1)$, the overlapping eigenvalues $\lambda^{(1)} = \lambda^{(0)}$ obtain two eigenvalue indices.

If both the indices and the eigenvalue multiplicities match, the number of actually computed eigensolutions in (σ_0, σ_1) matches the expected number $\nu(\sigma_1) - \nu(\sigma_0)$. Thus (σ_0, σ_1) is a trusted interval. All the solutions in this interval pass the validity check. Their status is then upgraded to *DONE*. Those not located inside a trusted interval have status *COMPUT*. Replicated eigensolutions are discarded.

When the indices do not match for all the overlapping eigenvalues, the newly computed eigensolutions are put on the list with status *COMPUT*. A new shift is selected somewhere between σ_0 and σ_1.

Case 2. None of the newly computed eigenvalues overlap the ones already on the Local Solution List. Then the status *COMPUT* is set for all newly computed eigensolutions. New shifts are selected at both sides of the active shift σ_1 by the shift selection process described in Section 3.2.

This bookkeeping process repeats recursively until all the solutions on the Local Solution List have status *DONE*.

3.4 Balance Parallel Workload

The workload for each process is proportional to the assigned number of eigensolutions $i_{k+1} - i_k$ or the length of the assigned eigenvalue spectrum $(\lambda^{(k)}_{\min}, \lambda^{(k)}_{\max})$ (see Figure 1), and can be dramatically affected by the eigenvalue distribution. In general, an accurate a priori workload estimate is impossible.

For the kth process, we name the interval $(\lambda^{(k)}_{\min}, \lambda^{(k)}_{\max})$ assigned to this process for eigenvalue computation the *assigned spectrum*. We call $[\sigma^{(k)}_{\min}, \sigma^{(k)}_{\max}]$ the *computed spectrum*, where $\sigma^{(k)}_{\min}$ and $\sigma^{(k)}_{\max}$ are the smallest and the largest local active or used shifts.

We balance parallel workload through dynamically updating the assigned spectrum during computation, which can be viewed as a particular implementation of diffusive load balancing [Corradi et al. 1999]. Initially, we distribute overlapping assigned spectra into processes, then reduce the overlap through neighboring exchanges of computed spectra.

Using *np* processes, we initially pick *np* points inside the requested global eigenvalue spectrum $(\lambda_{\min}, \lambda_{\max})$:

$$\lambda_{\min} < \sigma_0^{(0)} < \sigma_0^{(1)} < \cdots < \sigma_0^{(np-1)} < \lambda_{\max},$$

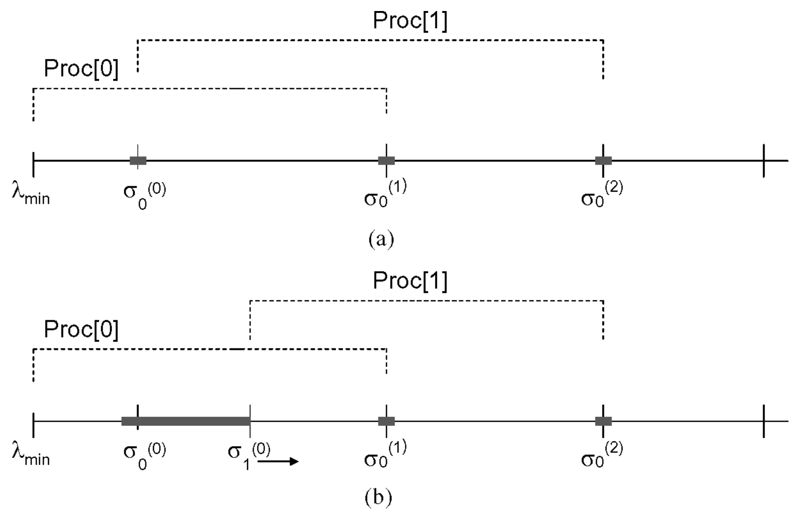

Fig. 3. Assigned spectrum (dashed line) and computed spectrum (bold line). (a) before updating. (b) after updating.

and form overlapping assigned spectra,

$$\left(\sigma_0^{(k-1)}, \sigma_0^{(k+1)}\right), \quad k = 0, \ldots, np-1, \quad \left(\sigma_0^{(-1)} = \lambda_{\min}, \sigma_0^{(np)} = \lambda_{\max}\right).$$

Process k takes $\sigma_0^{(k)}$ as its initial shift, and expands its initial computed spectrum $[\sigma_{\min}^{(k)} = \sigma_0^{(k)}, \sigma_{\max}^{(k)} = \sigma_0^{(k)}]$ outward by selecting new shifts from both sides of $\sigma_0^{(k)}$ as described in the previous sections. When a new $\sigma_{\min}^{(k)}$ or $\sigma_{\max}^{(k)}$ is computed (e.g., a new $\sigma_{\max}^{(k)}$), it is sent to the neighbor process $k+1$. Upon receiving it, process $k+1$ is informed that $[\sigma_0^{(k)}, \sigma_{\max}^{(k)}]$, a portion of its assigned spectrum, has been processed by process k. Then process $k+1$ updates its assigned spectrum by moving its $\lambda_{\min}^{(k+1)}$ from $\sigma_0^{(k)}$ inward to $\sigma_{\max}^{(k)}$. Figure 3 illustrates this scheme.

Assigning overlapping spectra enables some processes to compute more eigensolutions than others in the same time period. As computed spectra expand from the middle of assigned spectra at various rates, each process receives information about its neighbors' computed spectra and updates its own assigned spectrum. This procedure dynamically reassigns the workload among processes. At the end, np computed spectra cover the user requested global eigenvalue spectrum $(\lambda_{\min}, \lambda_{\max})$ with minimum overlap. Duplicate eigensolutions are dumped at the final phase of the computation, in Step (3) of Figure 1.

All processes implement this procedure by using asynchronous neighboring communication of short messages. We stress that the communication cost incurred is insignificant compared with the computational cost.

3.5 Accuracy of the Eigensolutions

In this section, we show how to ensure that the accuracy of the proposed eigensolver, by which we mean the residual norms

$$||r_i|| = ||Ax_i - \lambda_i Bx_i||_2, \tag{7}$$

fall within user specified tolerance, and that the orthogonalities:

$$\vartheta_{ij} = \left|x_i^T B x_j - \delta_{ij}\right|, \qquad \delta_{ij} = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j, \end{cases} \tag{8}$$

of *all* computed eigenpairs are confined to the uniformed order.

SIPs provides a set of user tunable parameters; among them, the three parameters required by the external Lanczos iteration package would strongly impact the accuracy and reliability of the computation:

—*tol*: the relative residual tolerance;

—*nev*: the number of eigenvalues requested for each given shift σ;

—*ncv*: the maximum dimension of the restart Krylov subspace.

A tight residual tolerance *tol* is crucial for obtaining a complete set of requested eigensolutions from a single shift-and-invert spectral transformation and ensuring the orthogonality of the computed eigenvectors within the same shift. The parameter *ncv* needs to be larger than the number of eigenvalues in a singleton.

The residual norms of all computed eigenpairs and the orthogonalities of the eigenvectors obtained from the same shift are inherited from the external eigenvalue software package that SIPs is built on, specifically, ARPACK [Lehoucq et al. 1998] in our current implementation. The ARPACK parameters *nev* and *ncv* are chosen to ensure that each eigenvalue singleton is computed through a single shift-and-invert transformation. Our numerical experiments show that ARPACK gives satisfying eigensolutions when the user provides sufficiently small error tolerance (see Section 4).

What remains to be addressed regarding the accuracy of eigensolutions is the orthogonality between eigenvectors computed from different shift-and-invert transformations.

Let us assume (λ_i, x_i) and (λ_j, x_j) are computed from two different shifts with the residual norms $||r_i||$ and $||r_j||$ bounded by tol. From $x_i^T r_j - x_j^T r_i$, we get the equation:

$$x_i^T B x_j = \frac{1}{\lambda_i - \lambda_j}\left(x_i^T r_j - x_j^T r_i\right).$$

Note, the eigenvalues in a singleton are computed through the same shift, thus $|\lambda_i - \lambda_j| \geq tol_{\text{deg}}$ because λ_i and λ_j belong to different singletons (see Section 3.3). The above equation then yields:

$$\left|x_i^T B x_j\right| \leq \frac{c}{|\lambda_i - \lambda_j|} tol \leq c\frac{tol}{tol_{\text{deg}}}, \text{ c is a constant depending on } B.$$

In SIPs, we set $tol_{\text{deg}} = \sqrt{tol}$ as the default, then the computed eigenvectors x_i and x_j are orthogonal within:

$$\left|x_i^T B x_j\right| \leq c\sqrt{tol},$$

which is satisfactory in most practical cases. Users have option to set tol_{deg} based on their own need.

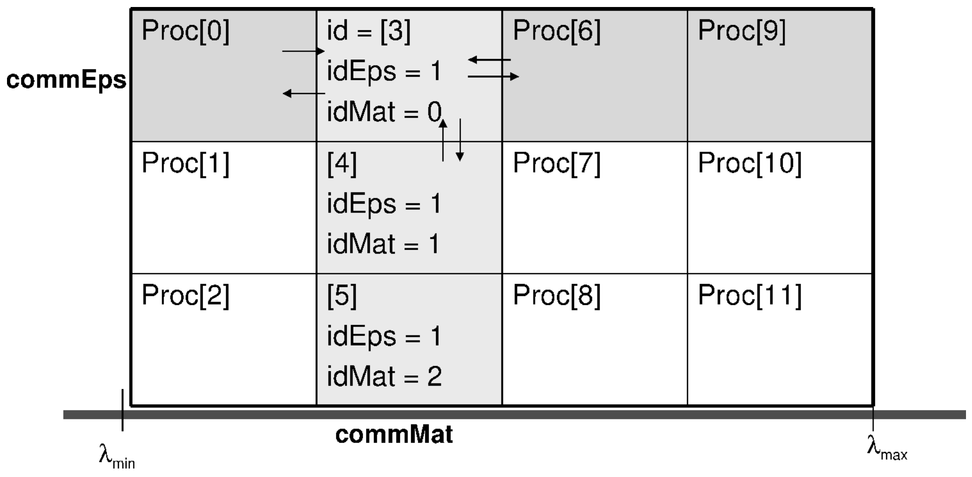

Fig. 4. Communicator layout.

3.6 Organize Subgroups of MPI Communicators

Up to this point, the discussion on SIPs has assumed that Step (2) of the Multiple Shift-and-Invert Parallel Eigenvalue Algorithm (see Figure 1) is processed sequentially; that is, a single process holds the entire matrix pencil (A, B) and implements shift-and-invert Lanczos iterations at distributed shifts. As the size of the eigenproblem increases, the local memory space becomes a limiting factor for holding the matrix factorization and distributed eigenvectors. In this situation, we must have multiple processes to provide adequate storage space collectively and execute matrix operations concurrently.

We cope with this local memory limitation by organizing subgroups of MPI communicators [Gropp et al. 1999]. An MPI communicator defines a context or scope for parallel communication. For example, *MPI_COMM_SELF* and *MPI_COMM_WORLD* are two default communicators normally used in MPI-based parallel programs. New communicators can be created by spliting existing ones for restricted communications. In SIPs, we introduce two new communicators, called *commEps* and *commMat*, by spliting *MPI_COMM_WORLD* into a 2D process grid. Each communicator *commEps* and *commMat* has *npEps* and *npMat* processes, for which we assume $npEps \times npMat = np$, the total number of processes in *MPI_COMM_WORLD*. Within each *commMat*, *npMat* processes concurrently implement operations described in the Multiple Shift-and-Invert Parallel Eigenvalue Algorithm (Figure 1) as sequential operations, which are primarily matrix operations. Every process also belongs to a communicator *commEps*, by which they exchange information about computed spectra for balancing workload among *commEps*, as discussed in Section 3.4. Figure 4 illustrates the layout of a $4(npEps) \times 3(npMat)$ process grid.

The process grid partitioning scheme described here shares interesting similarity with the frequency and geometric domain decomposition proposed by Komzsik [2003].

4. NUMERICAL EXPERIMENTS AND COMPARISON

From algorithmic analysis and numerical experiments, we find that the performance of our eigensolver is dominated by the computational cost of matrix factorizations and triangular solves. The communication time is ignorable when $npMat = 1$, and remains insignificant as $npMat$ is increased to meet memory demand. When $npMat = 1$, SIPs exchanges only neighboring short messages that overlap with computation. When the matrix size becomes too large to be held on a single process, $npMat$ is increased to the minimum number of processes for storing the matrix factorization. For our largest tests, $npMat = 4$ was sufficient to satisfy the memory demand. The communications within the small MPI communicators *commMat* remain insignificant for the total execution time.

We tested SIPs on eigenvalue problems arising from DFTB models of various materials and dimensionality. To give a balanced evaluation on SIPs' performance, we present here numerical results for three representative systems:

(1) a single-wall carbon nanotube[2];
(2) a diamond nanowire[3];
(3) a diamond crystal.[4]

We built each system at varying physical sizes, resulting in a consistent set of matrix sizes up to $n = 64{,}000$. For the first two test systems, one-dimensional periodic boundary conditions are applied in the physical model, and three-dimensional ones for the last system. The nature of the test systems and their physical boundary conditions are reflected in distinct initial sparsity patterns of the matrices A and B, and the resulting sparse densities of their factorizations. For example, when $n = 16{,}000$, the sparse densities of the matrix factorizations are 7% for the nanotube (fairly sparse), 15% for the nanowire, and 50% for diamond (dense).

The numerical experiments were performed on a Linux cluster called *Jazz*, at Argonne National Laboratory. Jazz comprises 350 computing nodes, each with a 2.4 GHz Pentium Xeon processor and a connection to both Myrinet and Ethernet communication networks. Nodes are equipped with at least 1 GB RAM.

In our experiments, we set $ncv = 200$, and $nev \leq ncv/2$. For all tests except the few cases discussed below, we specify the input relative residual tolerance as $tol = 10^{-8}$. All eigensolutions achieve the absolute residual norm $||r_i||$ from $O(10^{-10})$ to $O(10^{-12})$, Equation (7), and the numerical orthogonality $\vartheta_{ij} < 10^{-8}$, Equation (8). As future work, we plan to add a scheme that automatically adjusts these sensitive parameters when clustering eigenvalues are detected. Other user input parameters, such as the global eigenvalue bound $(\lambda_{\min}, \lambda_{\max})$ would affect the load balance in the first run of the eigenvalue problem, because the initial local shifts are chosen from it. Without a priori

[2]A (10,0) *armchair*-tube with diameter 1.36 nm; 20 atoms on the circumference, at varying length.
[3]Oriented along the (001) crystal direction with (110) faces and a cross section of $(1.14\,\text{nm})^2$, 25 atoms per layer, at varying length.
[4]Same orientation as the nanowire, with periodic boundary conditions in all three dimensions, at varying size of the supercell.

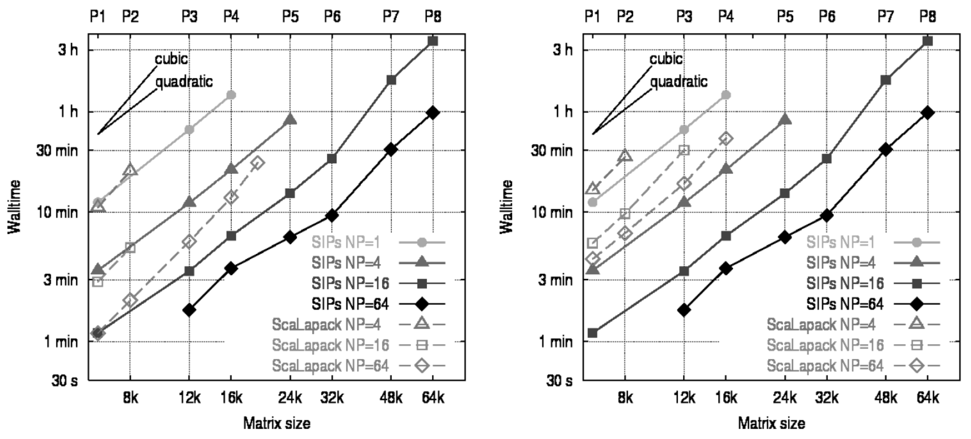

Fig. 5. Execution time for a single-wall carbon nanotube system on Myrinet (left) and Ethernet (right).

information about the eigenvalue spectrum, some processes get an empty *assigned spectrum*. Because the focus of this work is on the successive runs of the eigenvalue problem (1), these processes are left idle until next run in our current implementation. This leads to a load imbalance during the first run and typically results in about twice the execution time as for successive runs. Improving the job balancing when a priori spectral information is not available will also be considered in future work.

Figures 5–7 show the execution time for computing the lowest 60% of the eigenvalues and the associated eigenvectors for the test systems. Timings are collected from the second run of the eigenvalue problem (1), in which the previously computed eigensolutions provide initial eigenvalue approximations $\{\hat{\lambda}_i\}$ as discussed in Section 3.2. This is because the eigenvalue problem (1) is solved many times using initial eigenvalue approximations except for the first run. When the initial approximation $\{\hat{\lambda}_i\}$ is not available, the execution time of the current version of SIPs is sensitive to the user input data ($\lambda_{\min}$, $\lambda_{\max}$) and the initial distribution of assigned spectra. As noted above, with sensible spectral bounds,[5] the initial run typically takes up to twice the time of the successive executions of the eigenvalue problem Equation (1).

Figure 5 shows the execution time for a single-wall carbon nanotube system on Myrinet (left) and Ethernet (right). For matrix sizes $n = 6,400$ to $n = 32,000$, we use $npMat = 1$ because the memory per node is sufficient for holding the entire matrix factorization. When $n \geq 48,000$, we increase to $npMat = 4$. The sudden increase of execution time from $n = 32,000$ to $n = 48,000$ is caused by the delay of parallel matrix factorizations and triangular solves as implemented by MUMPS. Normally, one should not anticipate an ideal speedup of *np* when increasing the number of processes from 1 to *np* because of communication, algorithmic limitations and other overhead in parallel processing. We

[5]We note that in a materials modeling application like DFTB, the eigenvalue search interval ($\lambda_{\min}$, $\lambda_{\max}$) is known reasonably well, either from general materials properties or from traditional calculations of a smaller system.

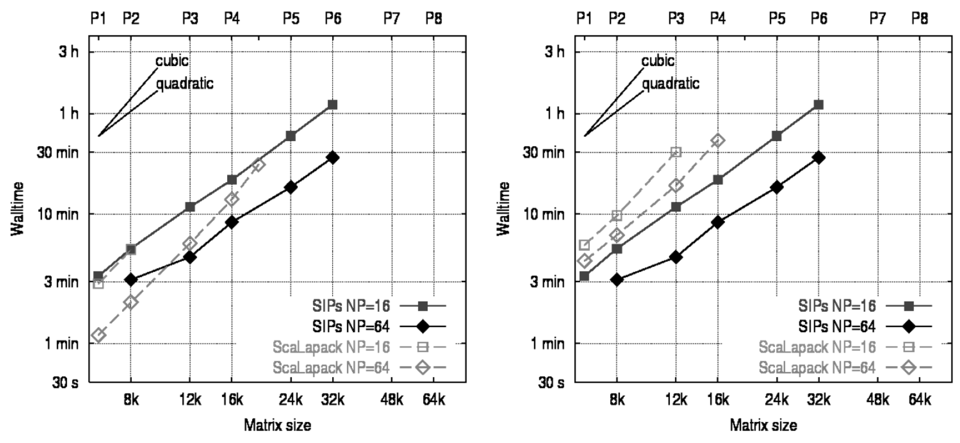

Fig. 6. Execution time for a diamond nanowire system on Myrinet (left) and Ethernet (right).

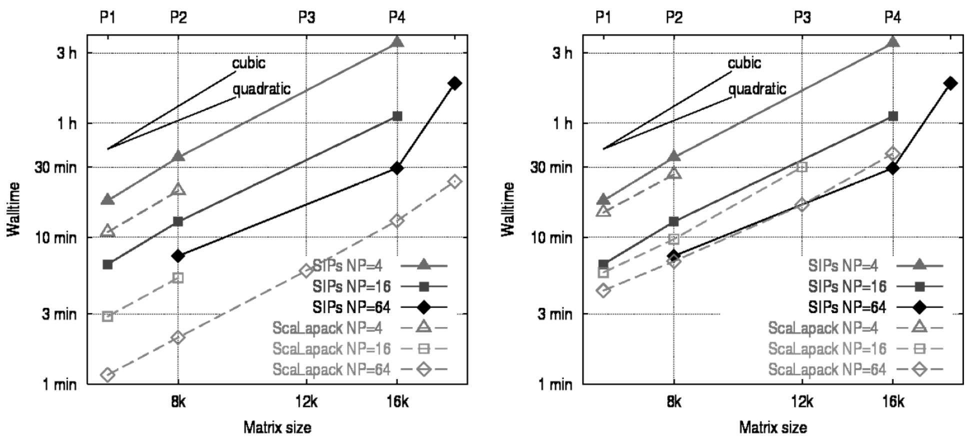

Fig. 7. Execution time for a diamond crystal system on Myrinet (left) and Ethernet (right).

find that for the same test system on the same number of processes, the execution time with *npMat* $= 4$ is approximately twice and five times as long as the case with *npMat* $= 1$ on Myrinet and Ethernet respectively. This indicates the speedup of matrix factorization and triangular solve implemented by MUMPS is approximately 2 from 1 process to 4 processes on Myrinet. We also observe much better speedups on our systems by MUMPS when the number of processes is increased from 4 processes to 16 or 64. When the problem size n becomes large (e.g., $n \geq 48{,}000$), we notice that among more than ten thousands of computed eigensolutions, one or two pairs of eigenvectors lose orthogonality. These eigenvectors are computed from the same shift. Therefore, we tighten the error tolerance to $tol = 10^{-11}$ for the cases of $n = 48{,}000$ and $n = 64{,}000$. The resulting execution times are about 3% higher than the cases with $tol = 10^{-8}$. Actually, it is only necessary to apply such strict error tolerance to the few shifts from which highly clustered eigensolutions are computed. What we need here is an *a priori* estimate for eigenvalue clusters, based on which the error tolerances can be adjusted dynamically. This is a subject for future development of SIPs.

Figure 5 includes a comparison with ScaLapack timings, specifically the subroutine PDSYGVX(), which is an expert driver routine for the symmetric generalized eigenproblem 1. All our ScaLAPACK tests refer to this subroutine. The figure clearly shows that the execution time scales with the problem size n as $O(n^2)$ for SIPs and $O(n^3)$ for ScaLAPACK. For a fixed problem size n, SIPs achieves a speedup of 3 or higher whenever the number of processes is increased 4 times. Because the matrices involved are sparse for this system, SIPs is significantly faster than ScaLAPACK; for example, for a problem size $n = 16{,}000$ using 64 processes, SIPs takes approximately 4 minutes vs. ScaLAPACK's 13, and 37 minutes on Myrinet and Ethernet respectively. ScaLAPACK fails to compute problems with a size larger than 19,200 due to memory limitations, while SIPs solves problems up to $n = 64{,}000$. Thus far, we have not seen any indication of memory restriction for SIPs.

Figure 6 shows the execution time for a diamond nanowire system with $npMat = 1$. The matrix factors involved are about twice as dense as in the previous system, with a density quantified somewhere between sparse and dense. SIPs takes a much longer execution time on this system as compared to the previous one that has sparser matrix factorizations, for example, 27 minutes versus 9 minutes for $n = 32{,}000$ and $np = 64$. However, the SIPs execution time still scales $O(n^2)$ with the problem size n. As the problem size increases, SIPs becomes faster than ScaLAPACK with increased performance on both Myrinet and Ethernet.

For the last system, a diamond crystal, the matrix factorizations are dense, with approximately 50% nonzero entries. Moreover, the spectrum has larger clusters and gaps than other tested systems. We must provide tighter error tolerance $tol \leq 10^{-12}$ to ARPACK for computing a complete set of eigensolutions from each shift. Figure 7 shows that ScaLAPACK is faster than SIPs on Myrinet, but its expensive communication cost on 64 processes with Ethernet pulls its performance behind SIPs. Because the matrices involved are dense, SIPs scales with the problem size n worse than $O(n^2)$, but still scales better than $O(n^3)$ (note the slopes of the curve). When $n > 16{,}000$, the memory available per node becomes too small for the matrix factorization, so we increase $npMat = 1$ to $npMat = 4$; hence, the execution time jumps at $n = 19,200$.

Summarizing all three test systems, we find the following. First, SIPs requires far less memory than does ScaLAPACK, which enables solutions of much larger eigenvalue problems. Second, ScaLAPACK requires extensive data communications as the number of processes or the problem size increases. Its performance is heavily affected by the speed of network. SIPs' communication cost is ignorable when $npMat = 1$, and remains insignificant for $npMat = 4$. Third, for matrices with sparse factorizations, the computational time for SIPs scales as $O(n^2)$ versus $O(n^3)$ for ScaLAPACK. SIPs' computational time strongly depends on the sparsity of the matrix factorization Equation (3). Although the number of collected data points is insufficient to draw a concluding scale, all systems together bear out the theoretical prediction $O(n * nnz)$, where nnz is the number of nonzero entries in the matrix factorization Equation (3). For matrices with sparse factorizations, the computational time is significantly smaller than for the ones with dense factorizations, for example, 21 minutes for the carbon nanotube system versus 3.5 hours on the diamond crystal system with $np = 4$ and $n = 16{,}000$. Moreover, SIPs is robust, giving accurate solutions for all the tested systems that have extremely pathological spectrums. For example, the carbon nanotube system with $n = 64{,}000$ has more than 30,000 requested eigenvalues clustered in a relatively tiny interval $(-0.9, 0.1)$. SIPs delivers all eigensolutions with the residual norms $||r_i|| \leq O(10^{-10})$, Equation (7), orthogonality $\vartheta_{ij} = O(10^{-8})$, Equation (8), and satisfying efficiency.

5. CONCLUSIONS

This article describes SIPs, a new efficient and robust software package implementing multiple shift-and-invert spectral transformations on parallel computers. It is developed on top of PETSc, SLEPc, ARPACK, and MUMPS for computing a large number of solutions of sparse real symmetric generalized eigenvalue problems.

We presented the algorithm and detailed implementation techniques. We demonstrated parallel numerical experiments on a set of selected eigenvalue problems from nanoscale materials modeling. A comparison of SIPs with ScaLAPACK on both fast and slow communication networks shows that SIPs (1) requires much less memory; (2) scales $O(n^2)$ vs. ScaLAPACK's $O(n^3)$ with the problem size n when the shifted matrix $(A - \sigma B)$ has sparse or not-very-dense matrix factorization; and (3) is robust, capable of computing large and pathological eigenvalue problems with high accuracy.

The object-oriented design makes SIPs applicable to most available Lanczos-based eigensolvers, especially the solvers provided or interfaced by SLEPc. Through this design, SIPs immediately inherits flexibility and portability from PETSc, functionalities of eigenvalue computation from SLEPc, and performance and robustness from the state-of-the-art external sparse matrix solvers (MUMPS and ARPACK in our current implementation).

The work reported here is not restricted to the eigenvalue problems posed by the DFTB method. It is a general robust eigensolver applicable to a wide range of sparse symmetric generalized eigenvalue problems. SIPs or its design and algorithmic approach can be adopted to leverage existing sparse eigenvalue software packages.

ACKNOWLEDGMENTS

We thank Satish Balay for assisting with software installations, YunKai Zhou for customizations of ARPACK, and Gary Kedziora for helpful discussions on load balancing. We also thank the referees for their constructive comments that improved the quality of this article.

REFERENCES

Amestoy, P. R., Duff, I. S., Koster, J., and L'Excellent, J.-Y. 2001. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl. 23*, 1, 15–41.

Amestoy, P. R., Duff, I. S., and L'Excellent, J.-Y. 2000. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Meth. Appl. Mech. Eng. 184*, 501–520.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. 1999. *LAPACK User's Guide, third edition*. SIAM, Philadelphia.

Balay, S., Buschelman, K., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., McInnes, L. C., Smith, B., and Zhang, H. 2006. PETSc users manual. Tech. Rep. ANL 95/11—Revision 2.3.1, Argonne National Laboratory.

Blackford, L., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1997. *ScaLAPACK User's Guide*. SIAM, Philadelphia.

Bostic, S. W. and Fulton, R. E. 1987. Implementaion of the Lanczos method for structural vibration analysis on a parallel computer. *Comput. Struct. 25,* 3, 395–403.

Corradi, A., Leonardi, L., and Zambonelli, F. 1999. Diffusive load-balancing policies for dynamic applications. *IEEE Concurrency 7*, 1, 22–31.

Elstner, M., Porezag, D., Jungnickel, G., Elsner, J., Haugk, M., Frauenheim, T., Suhai, S., and Seifert, G. 1998. Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties. *Phys. Rev. B 58,* 11, 7260–7268.

Ericsson, T. and Ruhe, A. 1980. The spectral transformation Lanczos method. *Math. Comp 34*, 1251–1268.

Grimes, R. G., Lewis, J. G., and Simon, H. D. 1994. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM J. Matrix Anal. Appl. 15,* 1 (Jan.), 228–272.

Gropp, W., Lusk, E., and Skjellum, A. 1999. *Using MPI, 2nd Edition*. MIT Press, Cambridge.

Hernandez, V., Roman, J. E., and Vidal, V. 2005. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Soft. 31,* 3 (Sept.), 351–362.

Komzsik, L. 2003. *The Lanczos Method, Evolution and Appication*. SIAM, Philadelphia.

Lanczos, C. 1950. An iteration method for the solution of eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand 45*, 255–282.

Lehoucq, R. B., Sorensen, D. C., and Yang, C. 1998. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia.

Marques, O. A. 1995. BLZPACK: Description and user's guide. Tech. Rep. TR/PA/95/30, CERFACS, Toulouse, France.

Parlett, B. N. 1998. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia.

Sleijpen, G. L. G. and van der Vorst, H. A. 1996. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl. 17*, 401–425.

Teranishi, K., Raghavan, P., and Yang, C. 2003. Time-memory trade-offs using sparse matrix methods for large-scale eigenvalue problems. In *Proceedings of the 2003 International Conference on Computational Science and its Applications, Lecture notes in Computer Science 2677*, Ed. V. Kumar, M. L. Gavrilova C. J. K. Tan, and P. L'Ecuyer. 840–847.

Wu, K. and Simon, H. 1997. A parallel Lanczos method for symmetric generalized eigenvalue problems. Tech. Rep. LBNL-41284, Lawrence Berkeley National Laboratory.

Received July 2005; revised April 2006; accepted May 2006