Shift-and-Invert Parallel Spectral Transformation Eigensolver: Massively Parallel Performance for Density-Functional Based Tight-Binding

Murat Keçeli,[a] Hong Zhang,[b] Peter Zapol,[c] David A. Dixon,[d] and Albert F. Wagner*[a]

The Shift-and-invert parallel spectral transformations (SIPs), a computational approach to solve sparse eigenvalue problems, is developed for massively parallel architectures with exceptional parallel scalability and robustness. The capabilities of SIPs are demonstrated by diagonalization of density-functional based tight-binding (DFTB) Hamiltonian and overlap matrices for single-wall metallic carbon nanotubes, diamond nanowires, and bulk diamond crystals. The largest (smallest) example studied is a 128,000 (2000) atom nanotube for which ~330,000 (~5600) eigenvalues and eigenfunctions are obtained in ~190 (~5) seconds when parallelized over 266,144 (16,384) Blue Gene/Q cores. Weak scaling and strong scaling of SIPs are analyzed and the performance of SIPs is compared with other novel methods. Different matrix ordering methods are investigated to reduce the cost of the factorization step, which dominates the time-to-solution at the strong scaling limit. A parallel implementation of assembling the density matrix from the distributed eigenvectors is demonstrated. © 2015 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.24254

Introduction

There is a substantial and growing demand for more accurate and faster computational approaches to treat the electronic structure of large systems. Part of this demand arises out of the practical need to discover and synthesize increasingly complex materials essential for alternative energy technologies[1] including energy capture, utilization, and storage. Modeling of novel materials, which often are primarily inorganic, involves consideration of possible compositions and metastable structures, multiple oxidation states, and ionic solution chemistries. Direct simulation of such processes is an important tool in establishing the critical steps and key principles governing the formation of new materials.

Synthesis design is just one of many problems whose solution requires the study of large numbers of interacting atoms to discover the mechanisms at work in the system. For decades, methods based on tight-binding density functional theory (DFTB)[2–5] and semiempirical molecular orbital (SEMO) theory[6–9] have had an important role to play in the study of large systems. They are computationally inexpensive and are of sufficient accuracy to discover potential key steps in the system's evolution. Such methods use minimal basis sets, reduce the number of two electron integrals to no more than $O(N^2)$, and parameterize all integrals to fit experimental or high quality *ab initio* training set data, which makes them inherently much faster than the full *ab initio* methods such as molecular orbital theory[10] or Kohn-Sham DFT.[11,12] These methods are also more appropriate than most molecular mechanics methods for reactive processes, especially for those reactions where oxidation states change or where the electronic rearrangements are not parameterized in the force field. They can be usefully accurate, because, in principle, they can be trained to a data set of high accuracy with a wide range of bonding motifs and system sizes. The semiempirical nature could avoid the accumulations of small errors in energy per bond as system size increases, incorporate non-bonded interactions as well as additional correlation effects,[13,14] treat electronically excited states,[15,16] and be incorporated into hybrid methods.[17–19] If appropriately parallelized on large-scale computers, semi-empirical and tight-binding methods could offer usefully accurate electronic structure calculations for either survey type high-throughput calculations systems of thousands of atoms, or detailed simulations of very large systems beyond many thousands of atoms. The former requires efficient strong scaling, while the latter requires efficient weak scaling characteristics for the parallelization in a unified framework.

A key step in many of the electronic structure methods is the need to diagonalize a matrix and retrieve the eigenvalues and the eigenvectors, hence, there is a continued effort to do it more efficiently.[20–24] This paper describes, Shift-and-Invert

[a] *M. Keçeli, A. F. Wagner*
Chemical Sciences and Engineering Division, Argonne National Laboratory, Argonne, Illinois, 60439
E-mail: wagner@anl.gov
[b] *H. Zhang*
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 60439
[c] *P. Zapol*
Argonne National Laboratory, Materials Science Division, Argonne, Illinois, 60439
[d] *D. A. Dixon*
Department of Chemistry, the University of Alabama, Tuscaloosa, Alabama, 35487
Contract grant sponsor: U. S. DOE (Office of Science); Contract grant number: DE-AC02-06CH11357; Contract grant sponsor: The University of Alabama (Robert Ramsay Fund)

Parallel spectral transformations method (SIPs), for the massive parallelization of a sparse eigensolver, initially for DFTB applications. The SIPs approach was originally developed by us,[20] and we describe here a newly developed massively parallel version with improved robustness and scalability with applications across hundreds of thousands of cores of a Blue Gene/Q supercomputer. Essential to the success of this approach is the sparsity of the Hamiltonian matrix; hence, it is especially appropriate for DFTB and SEMO type electronic structure methods. However, SIPs should be useful for any large but relatively sparse matrices with a considerable fraction of requested eigensolutions (eigenvalues and eigenvectors) such as large-scale DFT or molecular orbital methods.

The paper is organized as follows: we describe the main features of DFTB and SIPs in the next section and then present our numerical results with a detailed performance analysis and compare the SIPs approach to other methods before the conclusions.

Theory and Algorithmic Details

Density-functional tight-binding theory

The density-functional tight-binding method[2–5] (DFTB) is based on a Taylor series expansion of the total energy in Kohn-Sham density functional theory[11,12] (KS-DFT). The expansion is over the density fluctuations, $\delta\rho$, around some reference electron density ρ_0. The total energy up to second order contributions is given by:

$$E=\sum_{i}^{\mathrm{occ}}\langle\Psi_i|\hat{H}_0|\Psi_i\rangle+E_{\mathrm{rep}}+\frac{1}{2}\sum_{\alpha\beta}\Upsilon_{\alpha\beta}\left(U_\alpha,U_\beta,R_{\alpha\beta}\right)\Delta q_\alpha\Delta q_\beta. \quad (1)$$

E_{rep} is the repulsion energy which is approximated by fits to DFT calculations while $\Upsilon_{\alpha\beta}$ is the second derivative of the total energy with respect to an atomic charge, Δq_a are differences of actual and neutral (or formal oxidation state) atomic Mulliken charges, and $U_\alpha$and $R_{\alpha\beta}$ are the Hubbard parameters and interatomic distances, respectively. The Hamiltonian in the first term is given by:

$$\hat{H}_0=-\frac{1}{2}\nabla^2+V_{\mathrm{ext}}+\frac{1}{2}\int\frac{\rho_0(\boldsymbol{r}')}{|\boldsymbol{r}\text{-}\boldsymbol{r}'|}d\boldsymbol{r}'+V_{\mathrm{xc}}[\rho_0], \quad (2)$$

where V_{ext} is the potential energy of external interactions and V_{xc} is the exchange-correlation potential. The Kohn-Sham orbitals, Ψ_i, in eq. (1) are a linear combination of atomic orbitals, whose radial components are rigorously localized by a parameterized confinement potential. This expansion leads to a generalized eigenvalue problem in the form of

$$\boldsymbol{Hc_i}=\lambda_i\boldsymbol{Sc_i}, \quad (3)$$

where $\boldsymbol{S}$ is the real, symmetric, and positive definite overlap matrix of N non-orthogonal basis functions, $\boldsymbol{H}$ is the real, symmetric Hamiltonian matrix, $\boldsymbol{c_i}$ is the eigenvector containing the coefficients of the atomic basis functions, and λ_i is the corresponding eigenvalue for state i. The minimal basis and localization of the atomic orbitals make H and S matrices typically sparse for large systems. The last term in eq. (1) requires computation of $\Upsilon_{\alpha\beta}$ and Δq_a and leads to self-consistent charges via an iterative procedure.[4] This model is known as the self-consistent-charge DFTB (SCC-DFTB) method. Ignoring the last term leads to the non-self-consistent DFTB method.[25] Both SCC and non-SCC DFTB methods share common features with semiempirical methods in that minimal basis sets are used, many of the integrals are omitted, and terms are parameterized to reproduce external data, in this case selected full DFT energies.

The approximations in DFTB methods decrease of computational time by two or three orders of magnitude relative to conventional DFT methods and allow simulations beyond thousands of atoms.[25] The bottleneck in DFTB calculations is the matrix diagonalization step [solving eq. (3)], which formally scales cubically with the system size. Alternative approaches such as divide-and-conquer methods,[26] linear scaling algorithms,[27–29] and other novel methods[30] have been applied to avoid the diagonalization step, as discussed in the "Comparisons with Other Methods" section.

Shift-and-invert parallel spectral transformations

At the core of many electronic structure calculations is the eigenvalue problem of eq. (3). SIPs[20] was developed to solve this kind of sparse symmetric generalized eigenvalue problems in parallel for large-scale applications where many eigenpairs are required. The requirement of a large portion of the eigensolutions precludes conventional iterative methods, such as Lanczos[31], Jacobi-Davidson,[32] or similar recently developed algorithms,[24] that target a few extreme eigensolutions. SIPs addresses this issue by applying a multiple shift-and-invert eigenvalue algorithm over requested spectrum in parallel. In this algorithm, the eigenproblem in eq. (3) is first transformed with a shift, σ:

$$(\boldsymbol{H}-\sigma\boldsymbol{S})c=(\lambda-\sigma)\boldsymbol{S}c, \quad (4)$$

and then an inversion leads to a standard eigenvalue problem,

$$(\boldsymbol{H}-\sigma\boldsymbol{S})^{-1}\boldsymbol{S}c=\frac{1}{(\lambda-\sigma)}c, \quad (5)$$

with a transformed spectrum. Since $(\boldsymbol{H}-\sigma\boldsymbol{S})$ is nonsingular unless σ is an eigenvalue, it has a symmetric-indefinite triangular factorization:

$$\boldsymbol{H}-\sigma\boldsymbol{S}=\boldsymbol{LDL^T}, \quad (6)$$

where $\boldsymbol{L}$ is a lower triangular matrix with ones on the diagonal, and $\boldsymbol{D}$ is a block-diagonal matrix with at most 2×2 blocks. Once the matrix factors are computed, the eigensolutions of eq. (3) can be found with Lanczos iterations.[33] These iterations converge quickly for the eigenvalues that are close to the shift σ. Therefore, the key idea in this algorithm is to slice the eigenvalue spectrum into a set of intervals and, then in parallel, to find for each interval multiple shifts to efficiently

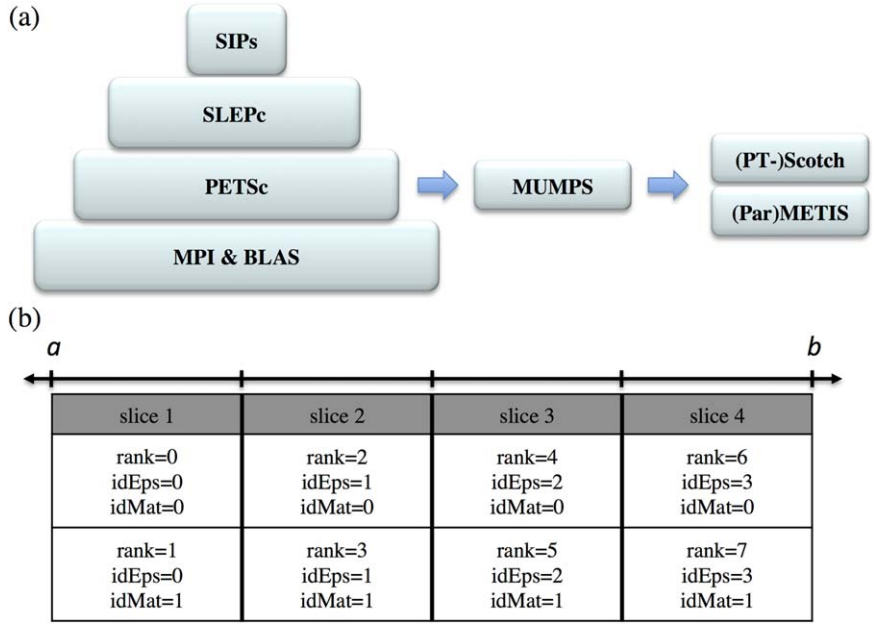

Figure 1. a) Software structure of SIPs. The layered structure indicates the requirements of the individual software packages, and arrows indicate the interfaces to external libraries. b) The MPI communication layout of SIPs for 4 slices (2 MPI ranks per slice) within the interval (a, b). idMat ranks communicators within a slice while idEps ranks communicators between slices. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

compute all of the eigensolutions in the interval. The exact number of eigensolutions in an interval is determined by computing the matrix inertia[34] at each end of the interval.

The design objective of the implementation of SIPs in a software package is to deliver high performance with limited effort for development and maintenance and to enable portability and reusability. Figure 1a shows the layered software structure of SIPs and the associated external libraries. PETSc[35,36] (Portable, Extensible Toolkit for Scientific Computation) provides parallel data structures and operations for various mathematical objects through a consistent interface with built-in profiling and error-checking capabilities. SLEPc[37,38] (Scalable Library for Eigenvalue Problem Computations) provides solvers for sparse eigenvalue problems. PETSc also provides an interface of external libraries for specific tasks, e.g. MUMPS (MUltifrontal Massively Parallel Solver)[39,40] is used to perform sparse matrix triangular factorizations and linear solves in parallel. MUMPS also has an interface for other external libraries such as (Par)METIS,[41,42] and (PT-)Scotch[43] to perform a (parallel) sparsity analysis that optimizes the factorization. A clear advantage of using libraries is that the improvements in performance and reliability become almost free as these libraries are actively maintained and updated. Moreover because of the large user base of these libraries, they provide portability across various computer architectures.

The SIPs algorithm can be briefly described in three steps. Step 1: Partition a given interval (a, b) into a given number of non-overlapping slices, s, where the k-th slice corresponds to a subinterval (a_k, b_k), and $k=1,\ldots, s$. Step 2: Divide p processors into s groups of processors and create MPI communicators as shown in Figure 1b (to be discussed in more detail shortly). Step 3: Compute all eigensolutions over the k-th slice by the k-th group of processors in parallel using the SLEPc Krylov-Schur algorithm with multiple shift-and-invert spectral transformations.

Here, we present the improvements in SIPs (compared to SIPs07[20]) that lead to a massively parallel performance and increased robustness in terms of five major tasks.

1. *Organize subgroups of communicators*: The MPI communicators[44] are organized as a two-dimensional process grid as depicted in Figure 1b to achieve two levels of parallelism. The coarse-grained parallelism (horizontal dimension in Fig. 1b) is achieved by distributing spectrum slices to subgroups of processors. We eliminate the overhead of communication between slices that was present in SIPs07 by using fixed boundaries for each slice (see task 2). Communication between slices is only required at the final stage of density matrix assembly. The fine-grained parallelism (vertical dimension in Fig. 1b) is achieved by performing parallel matrix operations in each slice. The matrices are stored in a distributed fashion across the members of this subgroup and all matrix operations are done concurrently. Therefore, communication within these subgroups of processors is substantial. While coarse-grained parallelism can be achieved on loosely coupled nodes, the subgroups of processors for fine grained-parallelism should be on a single node, or nodes with faster interconnects to achieve efficient parallelization.
2. *Balance parallel workload*: SIPs07 had overlapping slice boundaries, and these were adjusted dynamically to balance the workload. SIPs07 was run again with the adjusted boundaries, and these results were reported in the earlier study. While this approach is feasible with small number of cores, we found that it is not cost effective, especially at the strong scaling limit, where the

Table 1. Information for the smallest and largest examples of the three classes.

Label[a]	N[b]	N_a[c]	N_{nz}^H %[d]	N_{nz}^L %[e]
CNT8000	8,000	2,000	1.81	12.47
CNT512000	512,000	128,000	0.03	0.21
DNW8000	8,000	2,000	6.05	27.78
DNW128000	128,000	32,000	0.38	1.87
BDC8000	8,000	2,000	6.05	57.95
BDC64000	64,000	16,000	0.76	25.00

[a] We label an example by the number of basis functions after the class acronym. [b] The number of basis functions, which is also equal to the matrix size. [c] The number of atoms in the supercell. [d] Nonzero density percentage of original matrices, ***H*** and ***S***. [e] Nonzero density percentage of matrix factor, ***L***.

factorization is the major cost. In this limit, number of slices is large enough such that no additional shifts are performed in each slice; hence, the computational cost of each slice is similar. We also note that load-balance can be improved with the approximate knowledge of the eigenvalue spectrum, e.g. from the eigenvalues of a previous SCF iteration.

3. *Select shifts*: SIPs07 performed shift selections and used an external library, ARPACK,[45] for Lanczos iterations. A major improvement in SIPs is that, we switched to the recently developed spectrum-slicing solver in SLEPc,[21] which selects the shifts to compute all the eigenvalues and the eigenvectors in a given interval. SLEPc uses sophisticated techniques such as thick-restart Lanczos, Krylov update, and deflation to improve the robustness and parallel scalability.[21] SLEPc is also able to detect and apply a small perturbation if a shift corresponds to an eigenvalue; however, we haven't observed any instances of that in our applications.
4. *Bookkeeping*: All of the eigensolutions are stored locally over subgroups of processes for each slice. SLEPc performs the bookkeeping of these solutions and guarantees the correct number of eigensolutions by performing matrix inertia calculations on the boundaries of each slice.
5. *Ensure orthogonality of eigenvectors*: SLEPc library ensures the orthogonality of the eigenvectors while SIPs07 used the ARPACK for that purpose. Default parameters in SLEPc was found satisfactory for the error tolerance.

SLEPc library provided an exceptional robustness by performing the last three tasks and allowed us to focus on the parallel performance of SIPs.

Numerical Results and Discussions

All the computations discussed in this section were run on the IBM Blue Gene/Q supercomputers (Vesta and Mira) in the ALCF at Argonne National Laboratory.[46] On either machine, one node has 16 cores at 1.6 GHz and 16 GB of total memory.

Applications

We tested the SIPs software package with DFTB applications on three classes of carbon materials that were chosen in the earlier study:[20] single-wall metallic nanotubes (CNT), diamond nanowires (DNW), and bulk diamond crystals (BDC). All CNT, DNW, and BDC examples have three dimensional periodic boundary conditions applied; however, for physically one-dimensional examples, the CNT and DNW,[47] a vacuum space was applied in two dimensions orthogonal to the length to separate neighboring periodic images. Atomic positions in these systems are randomized by applying small random shifts (up to 0.15Å) from their equilibrium positions to approximate a typical structure in a dynamics simulation at an elevated temperature. This leads to lifting some of the degeneracies in the eigenvalue spectra and a broadening of the peaks in the density of states spectra. We have also performed calculations at the equilibrium geometry and these results are given in the Supporting Information.

The Hamiltonian and overlap matrices are produced using the same version of the DFTB software package as in our previous work.[20] The size of the matrices (number of rows) is equal to the number of basis functions (four times the number of atoms in an example). It should be noted that the Hamiltonian and the overlap matrices share identical nonzero patterns as discussed previously.[48] Table 1 gives information about the labeling notations, the number of basis functions, the number of atoms, and the nonzero density percentages of these examples, which are labeled by the class acronym followed by the number of basis functions, such as CNT8000. The nonzero density is calculated by dividing the number of nonzeros, N_{nz}, by the total number of elements in the upper triangular part of the matrices. The difference in the maximum size of the examples is because of differences in the computational intensities and memory requirements of the examples. Within each class, the number of atoms in the supercell is increased by powers of two from the minimum to the maximum size.

DFTB employs minimal basis-sets, hence approximately the lowest 60% of the eigensolutions is required in a typical application.[49] We label the energy range encompassing these eigensolutions as the global interval, and SIPs requires an estimation of this interval to search for these eigensolutions. We chose a global interval of (−0.8, 0.2 Hartree), which contains approximately 70% of eigenvalues for the CNT and 60% of eigenvalues for the DNW and BDC examples and divide it into uniform slices, where the number of slices and the number of processor cores (one MPI process per core) per slice are assigned by the user.

Sparsity and factorization

Sparse numerical algorithms store only the nonzero elements of a matrix and operate only on these elements, which can reduce the memory footprint and computational time significantly compared to dense numerical algorithms.[50] On the other hand, sparse numerical algorithms like SIPs, that require a factorization, have to address the problem of fill-ins. Fill-ins are basically the additional nonzeros that appear in the factors of the original matrices and they can hinder the performance of these algorithms. The initial sparsity patterns of our applications and the relationship to the ordering of the basis

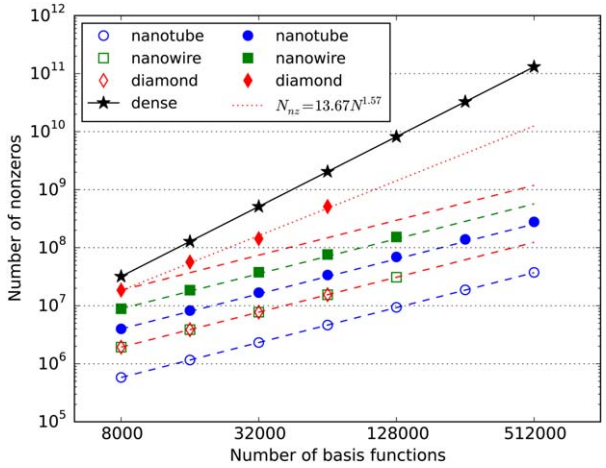

Figure 2. The plot of the number of nonzeros as a function of the number of basis functions for pre-factorization (open symbols) and post-factorization (solid symbols). Dashed lines show a linear increase in the number of nonzeros. Red dotted line shows a power fit (the equation is given in the legend) for the post-factorization number of nonzeros of BDC example. The quadratic growth of the number of elements in a matrix with dense storage is shown with stars and a solid line. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

functions are given in the Supporting Information. Since an ordering of the basis functions is simply a permutation of rows and columns of the Hamiltonian and overlap matrices, different sparsity patterns could be readily obtained for the same system without changing the value of the energy. While matrix ordering obviously does not change the number of nonzeros, it can significantly change the number of fill-ins. Unfortunately, finding the best ordering of a matrix that gives the minimum number of fill-ins is an NP-complete problem;[51] therefore, we are left with heuristics. We analyzed different matrix ordering methods that can reduce the number of fill-ins and found that Scotch[43] and METIS[41] libraries gave the least amount of fill-ins compared to reverse Cuthill-McKee algorithm[52,53] as implemented in Matlab,[54] and approximate minimum degree and approximate minimum fill algorithms as implemented in MUMPS.[55] We then compared the scalability of the parallel implementations of these libraries, PT-Scotch and ParMETIS, respectively, and found that PT-Scotch has a better scaling with number of cores. A detailed discussion of these results is given in the Supporting Information.

The growth of the number of nonzeros with problem size for original matrices and factors (based on Scotch library matrix ordering) of all examples is shown in Figure 2 and the last two columns of Table 1 lists the percentage of nonzero densities of the original matrices and their factors, respectively, for the smallest and largest examples of our three classes. The number of nonzeros of the original matrices increases linearly for all classes of the examples. This is simply a manifestation of the fact that the system size is larger than the extent of interactions governed by the cutoff distance of atomic orbital diffuse tails at about 0.55 nm for our DFTB applications. That is, the growth of the system size only increases the number of atoms, leaving the number of atoms interacting with a certain atom constant. After factorization, the number of nonzeros in the matrix still grows linearly for the CNT and DNW examples, but for the BDC examples, the number of fill-ins grows superlinearly yet not quadratically (a power fit gives an exponent of 1.57). The relationship between the scaling of the number of fill-ins and dimensionality of the systems is given in the context of finite-element methods.[56,57] It has been shown that the number of fill-ins grows linearly for essentially one-dimensional systems (long and thin) and grows with a power of 4/3 for three-dimensional problems, consistent with our results. The larger exponent we observe could either be an artifact of making a fit with 4 points, or our particular way of growing the system size for BDC (in a round-robin, one-at-a-time increase in each dimension).

Sparse matrix factorization is organized into two consecutive steps, symbolic factorization and numerical factorization, respectively, both of which are described in more detail in the Supporting Information. Symbolic factorization is done only once, since it depends only on the sparsity pattern of the matrix. Numerical factorization explicitly uses the slice-dependent shifts, and performed at both ends of a slice and at each additional shift. We plot the scaling of the symbolic and numerical factorization steps on one core (sequential) with the number of basis functions, i.e., matrix size in Figure 3. For the CNT and DNW examples, the computational time for both symbolic and numerical factorization steps scales linearly with system size and the numerical factorization step is generally faster by a factor of two as compared to the symbolic factorization. For the BDC examples[58] the computational time continues to scale linearly with system size for symbolic factorization, but scales near quadratically for numerical factorization as is consistent with the literature.[56,57]

Performance and scalability of SIPs

SIPs is based on two levels of parallelism, one due to distributed slicing of the spectrum and the other because of concurrency of the operations performed in a slice. Here, we investigate scaling of these two different levels of parallelism using CNT8000. Figure 4 shows, under two different conditions, the strong scaling (scaling with number of cores when the problem size is fixed) of the four important stages of the calculations: symbolic factorization, numerical factorization, solution, and orthogonalization. One condition (denoted as 'single slice' in the figure) has one slice containing all the required eigenvalues but applies increasing number of cores for this slice. The second condition has the restriction of one core per slice, but applies increasing number of slices to the eigenvalue spectrum. For each plot in Figure 4, the walltime for that stage corresponds to the total time spent in that stage.

For the 'single slice' condition, these plots clearly show that effective scaling does not extend beyond 16 cores in the current computer architecture. The use of 16 cores is limiting in these applications since it is the number of cores in a shared-memory node of the supercomputer and going to more nodes leads to an increase in the communication time, which cannot be compensated for by the parallelization of arithmetic operations for this size of problem. For

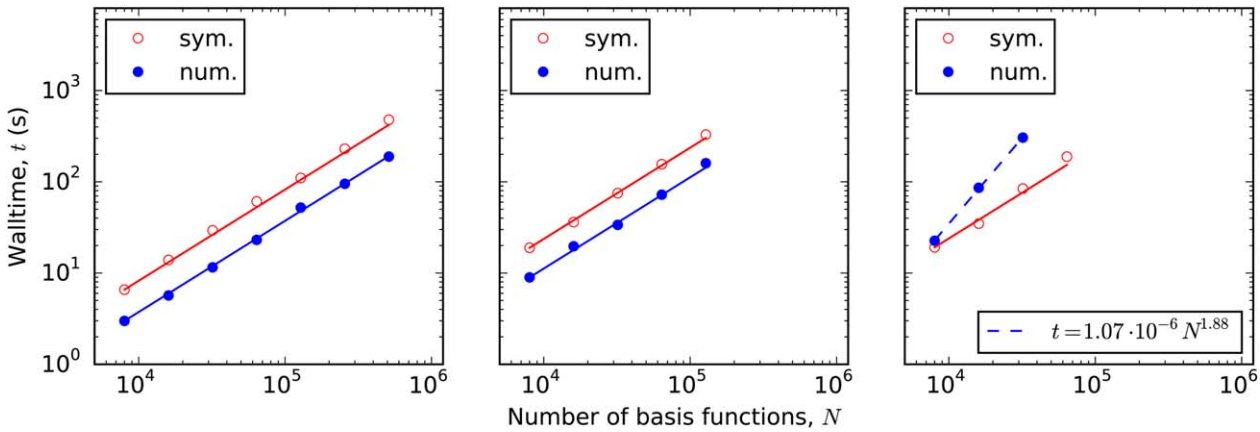

Figure 3. Walltime scaling with the number of basis functions for symbolic and numerical factorizations of the CNT (left), DNW (middle), and BDC (right) examples with Scotch-library matrix ordering. Solid lines show a linear increase in walltime as the number of basis functions increase while the dashed line in the right (BDC) plot shows a power fit for numerical factorization with the equation given in the legend. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

the 'single slice' condition, the results are consistent with the observation of others[47] that the solution step has the worst scaling.

For the condition of 'one core per slice', these plots show a better scaling for the numerical factorization, solution and orthogonalization stages. Adding more slices simply reduces the number of eigensolutions per slice, hence decreasing the computational time for these three stages. There is no sudden weakening of scaling beyond 16 cores since there is no overhead of communication between slices. The symbolic factorization stage does not scale, since we perform one symbolic factorization for each slice (symbolic factorizations for different slices are identical, but performed independently for each slice to avoid communication) and therefore increasing the number of slices does not change the cost of symbolic factorization. The numerical factorization scales much better compared to the 'single slice' condition since each additional slice reduces the total number of numerical factorizations per slice. The scaling levels off at 1024 slices (or cores) since we hit the minimum number of numerical factorizations possible per slice, which is two. This scaling can be easily modeled with Amdahl's law,[59] which can be formulated as:

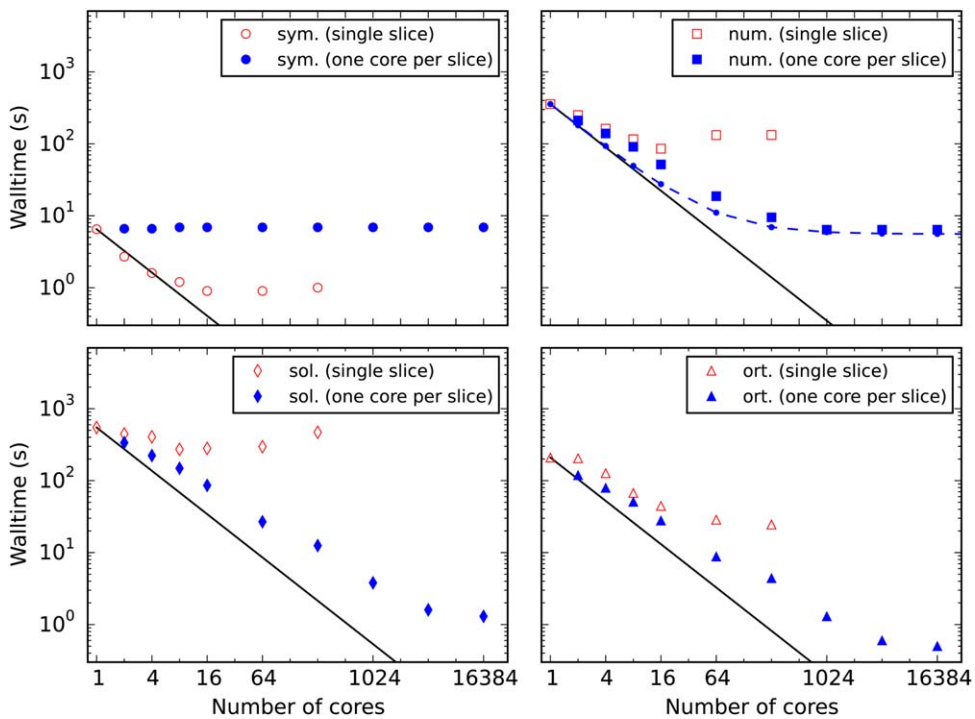

Figure 4. The decomposition of walltime scaling with the number of cores in terms of symbolic factorization (top left), numerical factorization (top right), solution (bottom left), and orthogonalization (bottom right) for CNT8000. Each plot shows the scaling using either all the cores for a single slice (up to 256 cores) or one core per slice (up to 16,384 slices). The straight lines show ideal linear scaling. The dashed line in the numerical factorization plot shows the change in time to solution based on Amdahl's law. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

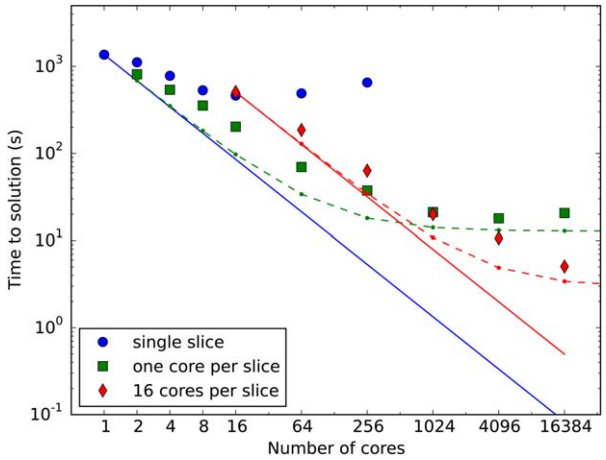

Figure 5. Strong scaling plot of CNT8000 using all cores for a single slice (blue circles), one core per slice (green squares), and 16 cores per slice (red diamonds). The straight line shows a linear decrease in time to solution with respect to number of cores. Dashed lines represent the change in time to solution based on Amdahl's law. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

$$t_p = t_1\left(f + \frac{(1-f)}{p}\right), \tag{7}$$

where t_p is the execution time with p cores (MPI processes), and $f \in [0,\ 1]$ is the fraction of time spent executing code that cannot be parallelized. In the strong scaling limit (p goes to infinity), speedup reaches the maximum value of $1/f$. For the example given in Figure 4, 128 numerical factorizations are required for a calculation with one slice and since we need to perform at least two numerical factorizations per slice, the maximum speedup is 64-fold. Figure 4 shows that SIPs with the 'one core per slice' condition reaches this limit at p=1024, but deviates from Amdahl's law for smaller number of slices as a result of load imbalance since number of factorizations for each slice is different. Figure 4 also suggests that the solution and orthogonalization stages have a similar limit at p=4096; however it is not possible to model it with using Amdahl's law using the profiling information we have from a single core run. The ultimate limit of slicing is obtained when there is at most one eigenvalue per slice; however, in practice, this limit depends strongly on the eigenvalue distribution. When there are large clusters of eigenvalues (a group of degenerate and/or nearly-degenerate eigenvalues), this limit is reached at a larger number of eigenvalues. The solution and orthogonalization scalings deviate from the ideal linear speedup due to this limit and load imbalance.

If the eigenvalue spectrum were known *a priori*, it would be possible to improve the scaling by adjusting the slice boundaries. Our experiments with non-uniform slices show that one can obtain up to a two-fold speedup for a small number of slices, a result consistent with the deviation from Amdahl's law in Figure 4, and the earlier SIPs study.[20] However, in the limit of a very large number of slices, these results also illustrate why uniform slices is the most cost-effective approach. Figure 4 shows that symbolic and numerical factorizations dominate the walltime in the limit of large slice number. For a serial run with one slice in Figure 4, the percentage of symbolic factorization, numerical factorization, solution, and orthogonalization stages are 0.6%, 31.8%, 48.9%, and 18.7%, respectively. For a parallel run with 16,384 slices, the relative costs are 45.7%, 41.9%, 8.9%, and 3.5% in the same order of stages. In the absence of any prior detailed information about the eigenvalue spectrum, it is the numerical factorization at each end of the uniform slice that provides the number of eigenvalues up to that slice. If that information were interpolated somehow to devise non-uniform slices with more load-balanced numbers of eigenvalues in each slice, another round of numerical factorization would still be needed at each end of the non-uniform slices to implement the SIPs method. Since in the limit of large numbers of slices, factorization is the leading cost of the calculation, non-uniform slicing would have to recoup this cost in superior performance in the other steps. Figure 4 shows that once the average number of eigenvalues per slice drops to a few tens of eigenvalues, further optimization of the number of eigenvalues per slice has diminishing returns. This is due to the fact that not only does the number of symbolic factorizations per slice remain the same, but also the number of numerical factorizations per slice reduces to the minimum number, two, because this is sufficient to obtain the few eigenvalues required in each slice. All gains in optimization of the slices are confined to the solution and orthogonalizaton step, the least costly operations in obtaining the eigensolutions in the large slice number limit. The only way non-uniform slicing can be cost-effective in this limit is if the slices can be determined without the additional cost of factorizations. In the context of self-consistent electronic structure calculations, this becomes possible after the first iteration of the self-consistent step, which gives an initial estimate of the eigenvalue spectrum. Studies of this approach are in progress.

For the CNT8000 example, Figure 5 shows a plot of the total time to solution with respect to the number of cores for three different methods of parallelization: using a single slice with all cores, using many slices with one core per slice, and using many slices with 16 cores per slice. For the 'single slice' condition, the scaling levels off at 16 cores as expected from the poor scaling of each stage of the calculation shown in Figure 4. The 'one core per slice' condition shows a better scaling compared to the 'single slice' condition, and it levels off at 1024 slices (cores) as expected from Amdahl's law given in eq. (7). The last method (16 cores per slice) is basically a combination of the first two approaches and has the best scaling beyond 1024 cores. When we use 16 cores per slice, the portion of the code that cannot be parallelized with additional slicing reduces significantly since the solution stages do not scale as well per slice as the factorizations. Therefore, we get a better maximum speedup compared to the 'single slice' condition. We can model this in the same spirit as Amdahl's law (shown with dashed lines), by using the computational time with one slice (16 cores) as our reference timing, t_1. Using 16 cores per slice we reach the slicing limit at a larger number of cores (16,384 cores) for this problem size as shown in Figure 5. The deviation from the limit of Amdahl's law is a result of the imperfect scaling of solution and orthogonalization stages as

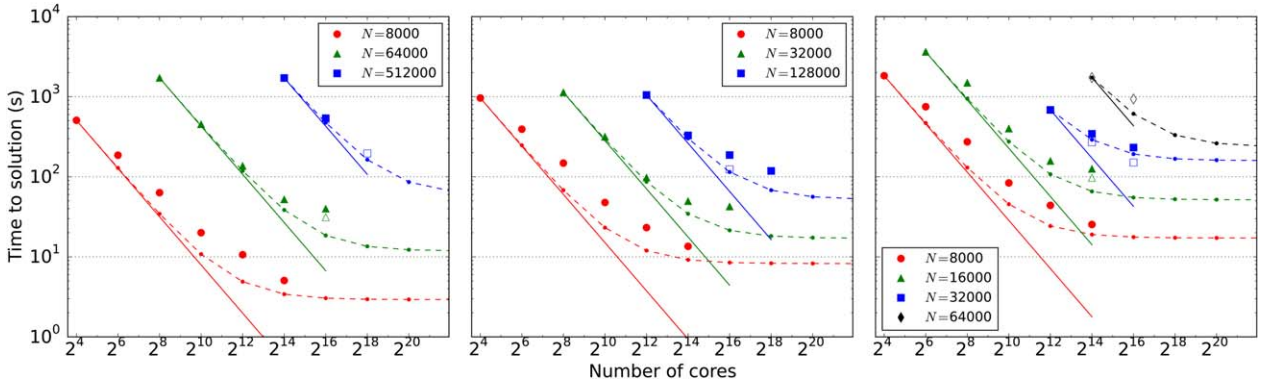

Figure 6. Strong scaling plots of the CNT (left), DNW (center), and BDC (right) examples with different number of basis functions represented by *N* in the legend. Straight lines correspond to a linear decrease in solution time with respect to number of cores. Dashed lines show the estimated time to solution based on Amdahl's law.

we see in Figure 4. We note that the available memory per core limits the applications one can perform with the 'one core per slice' condition. If the memory of the core is enough for the problem size, then 'one core per slice' is suitable until the limit of $\sim N/10$ slices, (e.g. around 800 in Figure 5) where *N* is the total number of basis functions. Beyond that point increasing number of slices would not decrease the computational time but adding more cores per slice will.

Figures 6 and 7 show strong and weak scaling plots, respectively, for all three classes of applications studied here. We use 16 cores (1 node) per slice for all of these examples with two exceptions (shown with unfilled symbols): 1) for the BDC64000 example, we assign 64 cores per slice since the available memory for 16 cores is not enough to store the factors; and 2) for selected examples at the maximum number of slices, we perform calculations with 32 cores per slice. We found that computations with 32 cores per slice are faster compared with 16 cores per slice when the problem size is large enough and the slicing limit has been reached. Unlike what we observe in Figure 5 for the smallest CNT example, the trade-off between the increase in communication time and reduction in arithmetic operations favors more cores per slice for larger problems.

The strong scaling plots in Figure 6 have ideal speed-up curves denoted with straight lines and curves based on Amdahl's law denoted with dashed lines. Both of them are drawn relative to the computation time obtained with lowest number of cores (or slices). As discussed earlier, we expect SIPs results to follow Amdahl's law rather than an ideal speedup curve, since there is a significant portion of the execution time that cannot be parallelized with more number of slices. By comparison of the smallest problem of each example (red filled circles), we see that the time-to-solution is relatively proportional to the number of nonzeros in the matrix factors. We also see that the CNT example has slightly better parallel efficiency in comparison to the DNW and BDC examples due to a better load balance in the absence of a gap in the eigenvalue distribution. The gaps in the DNW and BDC examples result in many idle processors since we slice the spectrum uniformly. We also observe that, as the problem size increases for a particular class of examples, the times to solution are closer to Amdahl's law curves than ideal speed-up asymptotes. This is mainly an artifact of drawing these curves relative to a calculation with more than one slice. The time-to-solution of the reference point is already suffering from load imbalance, and hence subsequent points seem to match the curves based on this result.

Figure 7 shows weak scaling plots for all three examples studied here. In Figure 8, the ratio, $r=N/\sqrt{p}$, the number of basis functions (*N*) divided by the square root of the number of cores (*p*), is fixed at 2000, 1000, 500, and 250. These ratios

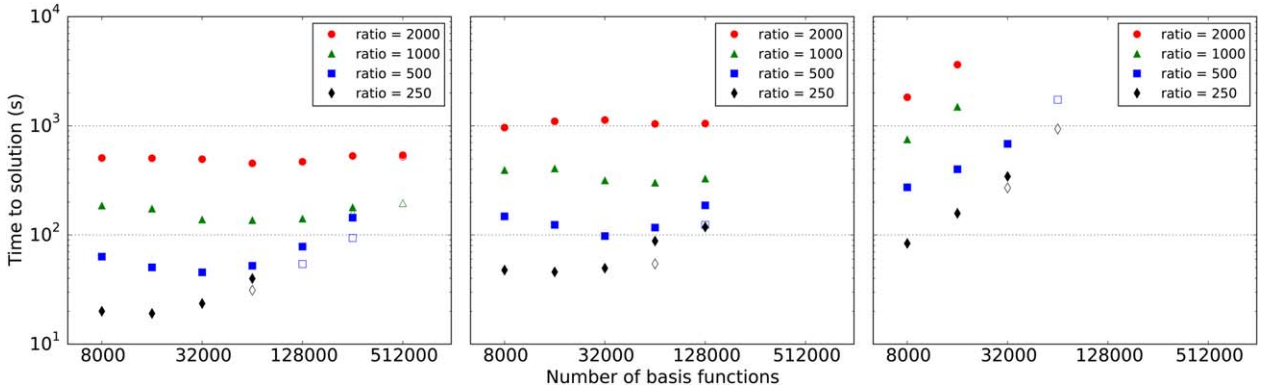

Figure 7. Weak scaling plots of the CNT (left), DNW (center), and BDC (right) examples with indicated fixed ratios of the number of basis functions to the square root of the number of cores. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

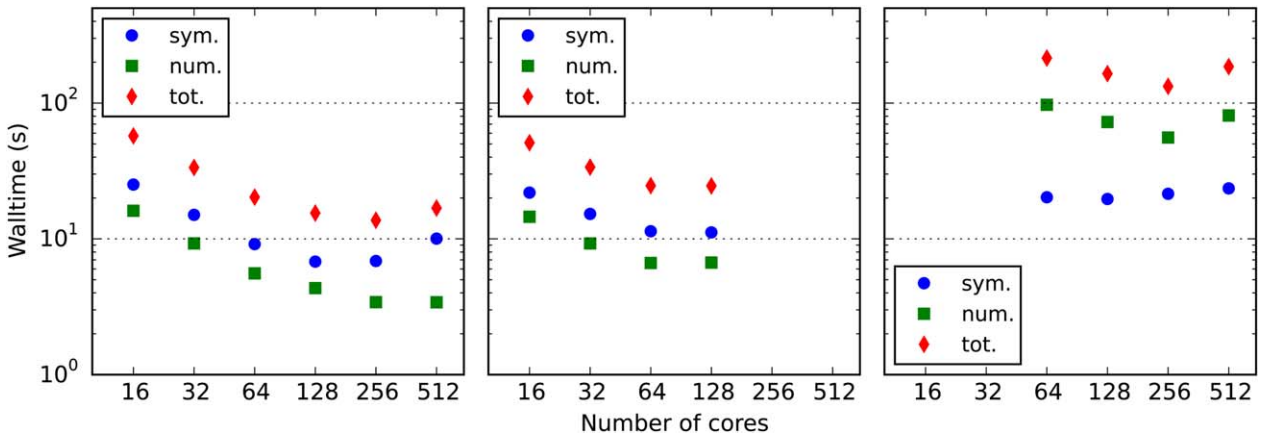

Figure 8. Strong scaling plots of symbolic (filled circles) and numerical factorizations (filled squares) of CNT (left), DNW (middle), and BDC (right) examples with the maximum number of basis functions (512,000, 128,000, and 64,000, respectively). Total cost of one symbolic and two numerical factorizations is shown with filled red diamonds. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

ideally lead to a constant time-to-solution, since doubling the problem size doubles both the number of eigensolutions and the operational cost of finding each. Figure 7 shows that one can achieve a constant time to solution for the CNT and DNW examples if the ratio of problem size to the computational power is held constant. However, this may not hold if one reaches the ultimate slicing limit of ~10 eigenvalues per slice. At this limit, time-to-solution is mainly determined by the cost of symbolic and numerical factorizations, which grows linearly with the system size and increasing the number of cores per slice is a more efficient approach than increasing the number of slices when one reaches the slicing limit. This can be seen by comparing unfilled symbols with filled symbols in Figures 6 and 7. On the other hand, for the BDC examples, time-to-solution could not be kept constant as the number of basis functions increases even at low numbers of slices. This is due to the fact that the cost of numerical factorization increases superlinearly as shown in Figure 3.

In Figure 8, we show the strong scaling plots of the symbolic and numerical factorizations of the largest examples that we studied. With red filled diamond symbols, we also show the total cost of one symbolic and two numerical factorizations, which corresponds to the largest portion of total time to solution at the maximum slicing limit. We observe that this cost can be reduced below 15, 30 and, 150 s if one can use more than 128 cores per slice for the CNT, DNW, and BDC examples, respectively. Hence, the total time to solution shown in Figures 6 and 7 can be reduced significantly when there are a large enough number of cores.

Density matrix assembly

In the SCC-DFTB scheme, the density matrix (DM) needs to be formed after obtaining the eigensolutions in order to update the charges in the last term of eq. (1). Density matrix assembly (DMA) has $O(N^3)$ scaling for dense matrices, but it reduces formally to $O(N^2)$ by exploiting the sparsity of DM, since we only need the elements of DM that have the same row and column indices as the nonzeros of the overlap matrix in DFTB calculations [see eq. (21) in Ref. 48 and Ref. 60]. Moreover, DMA can be efficiently parallelized as shown in Ref. 58. In their recent DFTB implementation, Scemama et al.[60] showed that observed scaling for DMA is $O(N^{1.1})$ and typically takes 3% of the total wall time.

In SIPs, the eigenvectors are distributed on different slices; hence the DM is assembled in parallel. We first form partial DMs on each slice and then perform an MPI all-reduce operation to sum the partial DMs. We present DMA timings as the percentage of the total wall clock time in Table 2. These results show that the DMA costs less than 2% of the overall computational time for our different use cases, which is consistent with Ref. 59. It is also clear from these results that adding more cores per slice can further reduce the DMA portion.

Comparisons with Other Methods

The main advantage of sparse eigensolvers, such as SIPs, compared with dense eigensolvers is the efficiency in memory consumption. On the other hand, the overhead of nonzero pointers and the irregularity in memory access can hinder the performance of sparse eigensolvers. Therefore, it is useful to determine this crossover point to identify which method should be preferred for different matrix sizes and sparsities.

Table 2. Percentage of the density matrix assembly (DMA) over the total wall clock time.

Label	Number of cores	Number of slices	DMA %
CNT8000	1024	64	0.55
CNT8000	1024	1024	1.09
CNT8000	16,384	512	0.28
CNT8000	16,384	1024	0.44
CNT512000	16,384	512	1.05
DNW8000	1024	64	0.83
DNW8000	16,384	64	0.31
DNW8000	16,384	1024	0.41
DNW64000	16,384	64	0.27
BDC8000	16,384	1024	0.22
BDC32000	16,384	1024	0.34

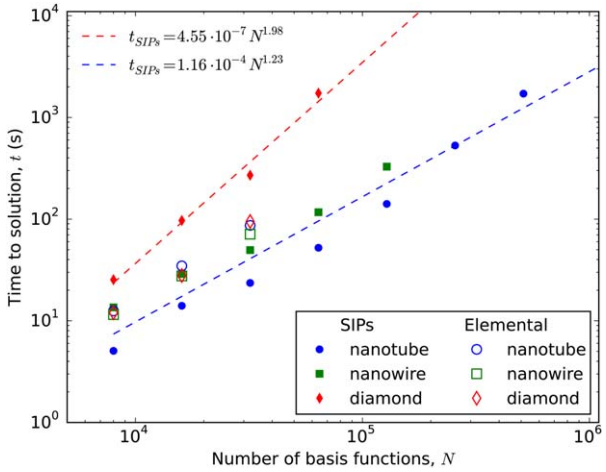

Figure 9. Problem size scaling plot for all examples calculated with SIPs (filled symbols) and Elemental (unfilled symbols) using 16,384 cores. *N* corresponds to number of basis functions (matrix size). Dashed lines show a power fit for BDC examples (red) and for the CNT and DNW examples (blue) computed by SIPs.

We compare SIPs with the generalized eigenvalue solver with a dense matrix storage in the recently developed linear algebra library, Elemental,[61,62] which has been shown[61,63] to have a better parallel performance compared to ScaLapack.[64] Our comparison is based on the computational time to find all of the eigensolutions in an interval corresponding to 60% (DNW, BDC) to 70% (CNT) of the spectrum. We utilize 16,384 cores for all the examples with a maximum of 32,000 basis functions[65] and use a square process grid for Elemental, which is essential to obtain the best performance.[62] Figure 9 shows that SIPs is almost two times faster than Elemental for CNT8000, suggesting that the crossover point is even lower than 8000 basis functions, where the nonzero density is 12.5% (see last column of Table 1). With larger examples, nonzero densities go even lower, hence the difference between SIPs and Elemental timings increases. For DNW, the crossover point is between DNW16000, and DNW32000, where the nonzero densities are 14.5% and 7.4%. The CNT and DNW examples suggest that a nonzero density lower than 14% is required for SIPs to outperform Elemental. On the other hand, for BDC, we see that Elemental is faster than SIPs for the given examples. For BDC32000, the largest BDC example with Elemental timings, the nonzero density of the factor is 28%, well above the crossover density for CNT and DNW. In Figure 9, we also plot power fits to the data points obtained for SIPs. The asymptotic scaling of SIPs for the BDC examples is quadratic based on this fit, which shows that SIPs will eventually outperform Elemental as the matrix size increases (nonzero density decreases), since dense-direct solvers have ultimately a more restrictive cubic scaling.

It should be noted that there are three caveats that limit the generality of such a comparison. 1) *Problem type*: The performance of dense solvers depends less on the specifics of the problem type as can be seen from the smaller difference of the results for different type of examples in Figure 9. However, the requested number of eigensolutions and distribution of the eigenvalues (gaps, clustering, etc) directly affect the performance of SIPs. For example, calculating 60% vs 30% of the eigenvalue spectrum has a proportional effect on the performance of SIPs while the effect is less for the solver in Elemental.[61] On the other hand, if there is high degeneracy in the spectrum, the performance of SIPs deteriorates as shown in the Supporting Information. 2) *Hardware*: We utilized 16,384 cores of a BG/Q system, which has a 5D torus interconnect. If we use more (less) number of cores, the crossover point can be reached at a higher (lower) density of nonzeros favoring SIPs (Elemental) since we found that Elemental reached the strong scaling limit for all examples while SIPs could make use of more cores. Nevertheless, if we use a machine with a slower interconnect, such as the clusters typically used in many research groups, the performance of SIPs should not be affected as long as the matrices fit into memory on a single node, while Elemental would suffer due to intense communications. 3) *Software*: We compared SIPs with Elemental because of its proven performance. However, there are other libraries, such as ELPA,[22] which have also been shown to scale on thousands of cores.[63] Obviously, using different libraries or improvements in these libraries can change the crossover point.

SIPs can also be compared with other novel methods that avoid diagonalization partially,[60,66] or completely, such as linear scaling methods.[67–70] Despite the advantage of enabling the study of very large systems, there are known issues in applying linear scaling methods to arbitrary systems. First, these methods are applicable only to systems that have a gap between occupied and unoccupied electronic states, that is, they cannot be applied to metallic or highly conjugated systems. Second, one needs to be able to control the error propagation as the system size grows. (See e.g. Ref. 71) We refer the readers to comprehensive reviews[28,29,72] for more details on these methods. A main advantage of SIPs is that it does not suffer from these drawbacks and is applicable generally to any system.

PEXSI[30,73–76] (pole expansion and selected-inversion method) is another lower scaling method that stands apart from the linear scaling approaches because it is generally applicable for both insulators and metals like SIPs. PEXSI is based on the approximation of the Fermi−Dirac function using a pole expansion technique and computing the charge density by the selected-inversion algorithm.[74] The computational cost of a single SCF iteration of PEXSI method is $n_\mu \times n_p \times t_P$, where n_μ is the number of iterations to compute the chemical potential, n_p is the number of poles, and t_P is the cost of computing the charge density contribution of a single pole, which involves factorization and inversion steps.[74] Both the factorization and inversion steps scale with $N^{\frac{d+1}{2}}$, where d is 3 for bulk systems, 2 for slabs and surfaces, and 1 for long and thin systems. The number of iterations for the chemical potential varies significantly depending on the initial guess for the chemical potential, band gap, and the required accuracy.[76] The number of poles (typically 40-80[30,76]) in PEXSI is independent of the system size, but depends on the given electronic temperature and spectral width (the difference between max/min eigenvalues) of the Kohn-Sham Hamiltonian.

On the other hand, the computational cost for SIPs is $n_s \times t_S$, where n_s is the total number of shifts and t_S is the combined cost of one numerical factorization and the solution/orthogonalization step per shift. (We do not include the cost of symbolic factorization either in t_S or t_P since it has the same cost in both methods and needs to be performed only once.) The scaling of t_S is the same as t_P, but we expect it to be much smaller because of two reasons. First, all the matrices in SIPs are real and symmetric, while PEXSI involves complex symmetric matrices, which increase the cost of every multiplication up to four times and the memory requirement by two times. Second, the solution and orthogonalization stages in SIPs have a scaling lower than factorization, while the inversion step in PEXSI has the same scaling with factorization and requires more computational time and memory as shown in Ref. 76. On the other hand the main advantage of PEXSI is that the cofactor of t_P, $n_\mu \times n_p$, is independent of system size, while n_s depends on the system size (increasing linearly at most). However, all the shifts in SIPs can be performed independently, hence when run in parallel, n_s per slice can be reduced to 1 (2 with the current implementation). Similarly, all the calculations of PEXSI at different poles can be done independently, but the iterations over chemical potential should be performed sequentially, hence the cost reduces to $n_\mu \times t_P$. Therefore, SIPs should be superior when there are enough computer resources (e.g. at the strong scaling limit) so that n_s per slice is less than or comparable to n_μ. However, with fixed computer resources, PEXSI would eventually be superior as the problem size increases, since it has a lower scaling. Besides the scaling comparison, the additional information that SIPs provides about eigenvalues and eigenvectors can be very valuable in interpreting the electronic structure of the system or in calculating properties of excited states.

There are also other methods that are based on spectral transformations[77–79] similar to SIPs. Schofield *et al.*[77] use a spectrum slicing method with the Rayleigh—Ritz procedure, and focus on techniques that provide the correct number of eigenvalues per slice without performing factorizations. Briggs *et al.*[79] utilize a partitioned folded spectrum scheme and their method is applicable specifically for diagonally dominant matrices. Both of these methods have $O(N^3)$ computational cost. Aktulga *et al.*[23] recently report a parallel implementation of multiple shift-and-invert Lanczos (MSIL) method. Unlike SIPs, the length of each slice in MSIL cannot be chosen arbitrarily, and must be small enough to ensure convergence. In addition, MSIL does not check the correctness and completeness of computed solutions. For large problems with clustered eigenvalues, missing or getting extraneous eigensolutions often occurs. From our knowledge, SIPs is the only sparse eigenvalue solver that demonstrates a robust and strong scaling performance beyond hundreds of thousands of cores for DFTB applications.

Conclusions

We present a massively parallel Shift-and-Invert Parallel spectral transformation (SIPs) method with remarkable robustness. We demonstrate its applicability for systems beyond 500,000 basis functions and its parallel scaling beyond 250,000 cores. We modeled the scaling of SIPs with respect to number of slices and showed that each slice contains less than a few tens of eigenvalues at the strong scaling limit. The major computational cost of SIPs is factorization in this limit and it scales linearly with the system size for effectively one-dimensional systems such as carbon nanotubes and diamond nanowires.

We compared time-to-solution of SIPs with Elemental's dense-direct solver. We found that SIPs is faster than Elemental even for the smallest nanotube example with a post-factorization nonzero density of 12.5%. For three-dimensional systems such as the diamond examples with a much larger nonzero density (25% to 58%), we showed that the scaling of SIPs with the problem size is quadratic in contrast to the cubic scaling of dense-direct solvers. We also discuss how SIPs compares favorably to other novel methods, such as PEXSI, in the strong scaling limit although PEXSI would be more efficient as the system size increases when the computational resources are limited.

In summary, we show that SIPs is a scalable method applicable to very large systems without the applicability limitations (e.g. metals) of linear scaling methods or without the scalability limitations (cubic scaling) of dense solvers. Unlike other methods, knowledge of the eigenvalue spectrum improves the performance of SIPs in subsequent self-consistent iterations since load-balancing can be improved by adjusting slice boundaries. Furthermore, the symbolic factorization step of the first iteration need not be repeated when the nonzero structure remains unchanged, as is the case in tight-binding methods. We also note that SIPs is a generally applicable robust solver for sparse matrix problems where a large portion of eigensolutions is required and we have already contributed our implementation to the SLEPc library.

Acknowledgments

The authors thank Dr. Jeff Hammond (ANL and Intel) and Dr. Álvaro Vázquez-Mayagoitia (ANL) for useful discussions on competitive approaches to SIPs. MK thanks the PETSc, SLEPc, MUMPS, SCOTCH, and Elemental developers for their help on using these libraries. MK also acknowledges The Argonne Training Program on Extreme-Scale Computing (ATPESC). This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

Keywords: sparse matrices · spectrum slicing · generalized eigenvalue problem · semi-empirical methods · tight-binding DFT

How to cite this article: M. Keçeli, H. Zhang, P. Zapol, D. A. Dixon, A. F. Wagner. *J. Comput. Chem.* **2016**, *37*, 448–459. DOI: 10.1002/jcc.24254

Additional Supporting Information may be found in the online version of this article.

[1] Kalil, T.; Wadia, C. Materials Genome Initiative for Global Competitiveness http://www.whitehouse.gov/sites/default/files/microsites/ostp/materials_genome_initiative-final.pdf (accessed Feb 12, 2015).

[2] J. C. Slater, G. F. Koster, *Phys. Rev.* **1954**, *94*, 1498.
[3] W. Foulkes, R. Haydock, *Phys. Rev. B* **1989**, *39*, 12520.
[4] M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, T. Frauenheim, S. Suhai, G. Seifert, *Phys. Rev. B* **1998**, *58*, 7260.
[5] G. Seifert, *J Phys. Chem. A* **2007**, *111*, 5609.
[6] J. A. Pople, D. P. Santry, G. Segal, *J. Chem. Phys.* **1965**, *43*, S129.
[7] M. J. S. Dewar, W. Thiel, *J. Am. Chem. Soc.* **1977**, *99*, 4899.
[8] J. J. P. Stewart, *J. Comput. Chem.* **1989**, *10*, 209.
[9] W. Thiel, *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2014**, *4*, 145.
[10] R. J. Bartlett, M. Musiał, *Rev. Mod. Phys.* **2007**, *79*, 291.
[11] P. Hohenberg, W. Kohn, *Phys. Rev.* **1964**, *136*, B864.
[12] W. Kohn, L. Sham, *J. Phys. Rev.* **1965**, *140*, A1133.
[13] P. C. Redfern, P. Zapol, L. A. Curtiss, K. Raghavachari, *J. Phys. Chem. A* **2000**, *104*, 5850.
[14] S. Grimme, *Org. Lett.* **2010**, *12*, 4670.
[15] J. E. Ridley, M. C. Zerner, *Theor. Chim. Acta* **1976**, *42*, 223.
[16] Zerner, M. C. In *Reviews in Computational Chemistry*; K. B. Lipkowitz, D. B. Boyd, Eds.; Wiley-VCH, Inc., **1991**; pp 313–365.
[17] M. Svensson, S. Humbel, R. D. J. Froese, T. Matsubara, S. Sieber, K. Morokuma, *J Phys. Chem.* **1996**, *100*, 19357.
[18] Vreven, T.; Morokuma, K. In *Annual Reports in Computational Chemistry*; D. C. Spellmeyer, Ed.; Elsevier: Amsterdam, **2006**; Vol. 2, pp 35–51.
[19] Gao, J. In *Reviews in Computational Chemistry*; K. B. Lipkowitz, D. B. Boyd, Eds.; VCH Publishers, Inc.: New York, **2007**; pp 119–185.
[20] H. Zhang, B. Smith, M. Sternberg, P. Zapol, *ACM Trans. Math. Softw.* **2007**, *33*, 9.
[21] C. Campos, J. E. Román, *Numer. Algorithms* **2012**, *60*, 279.
[22] a. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, a. Heinecke, H. J. Bungartz, H. Lederer, *J Phys. Condens. Matter* **2014**, *26*, 213201.
[23] H. M. Aktulga, L. Lin, C. Haine, E. G. Ng, C. Yang, *Parallel Comput.* **2014**, *40*, 195.
[24] D. Zuev, E. Vecharynski, C. Yang, N. Orms, A. I. Krylov, *J. Comput. Chem.* **2015**, *36*, 273.
[25] M. Elstner, G. Seifert, *Philos. Trans. R. Soc. A* **2014**, *372*, 20120483.
[26] W. Yang, *Phys. Rev. Lett.* **1991**, *66*, 1438.
[27] A. D. Daniels, G. E. Scuseria, *J. Chem. Phys.* **1999**, *110*, 1321.
[28] S. Goedecker, *Rev. Mod. Phys.* **1999**, *71*, 1085.
[29] D. R. Bowler, T. Miyazaki, *Reports Prog. Phys.* **2011**, *036503*, 84.
[30] L. Lin, A. García, G. Huhs, C. Yang, *J. Phys. Condens. Matter* **2014**, *26*, 305503.
[31] C. Lanczos, *J Res. Natl. Bur. Stand. (1934).* **1950**, *45*, 255.
[32] G. L. G. Sleijpen, H. A. Van der Vorst, *SIAM Rev.* **2000**, *42*, 267.
[33] T. Ericsson, A. Ruhe, *Math. Comput.* **1980**, *35*, 1251.
[34] J. J. Sylvester, *Philos. Mag. IV* **1852**, *4*, 138.
[35] Balay, S.; Gropp, W. D. W.; McInnes, L. C. L.; Smith, B. F. B. In *Modern Software Tools in Scientific Computing*; E. Arge, A. M. Bruaset, H. P. Langtangen, Eds.; Birkhäuser Press, **1997**; pp 163–202.
[36] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, H. Zhang, *PETSc Users Manual, ANL-95/11 - Revision 3.5* **2014**,
[37] V. Hernandez, J. E. Román, V. Vidal, *ACM Trans. Math. Softw.* **2005**, *31*, 351.
[38] C. Campos, J. E. Román, E. Romero, A. Tomás, *SLEPc Users Manual, DSIC-II/24/02 - Revison 3.5* **2014**,
[39] P. R. Amestoy, I. S. Duff, J. Y. L'Excellent, J. Koster, *SIAM J. Matrix Anal. Appl.* **2001**, *23*, 15.
[40] P. R. Amestoy, A. Guermouche, J. Y. L'Excellent, S. Pralet, *Parallel Comput.* **2006**, *32*, 136.
[41] G. Karypis, V. Kumar, *SIAM J. Sci. Comput.* **1998**, *20*, 359.
[42] G. Karypis, V. Kumar, *J Parallel Distrib. Comput.* **1998**, *48*, 71.
[43] C. Chevalier, F. Pellegrini, *Parallel Comput.* **2008**, *34*, 318.
[44] W. Gropp, E. Lusk. In *Proceedings of Scalable Parallel Libraries Conference*; IEEE Comput. Soc. Press, **1993**; pp 160–165.
[45] R. B. Lehoucq, D. C. Sorensen, C. Yang, *Communication* **1998**, *6*, 147.
[46] Argonne Leadership Computing Facility https://www.alcf.anl.gov/user-guides/mira-cetus-vesta (accessed Mar 7, **2015**).
[47] The vacuum space applied for the nanowire examples was not large enough to prevent interactions with images. Please see the Supporting Information for more details.
[48] B. Aradi, B. Hourahine, T. Frauenheim, *J Phys. Chem. A* **2007**, *111*, 5678.
[49] For our examples, carbon atoms have 4 valence electrons and 4 orbitals/atom so one-half of the orbitals are doubly occupied, but, for an example for water clusters, two-thirds of the orbitals are doubly occupied.
[50] Pissanetzky, S. Sparse Matrix Technology, 1st ed.; Academic Press: London, **1984**.
[51] M. Yannakakis, *SIAM J. Algebr. Discret. Methods* **1981**, *2*, 77.
[52] E. Cuthill, J. McKee. In *Proceedings of the 1969 24th national conference*; ACM Press: New York, NY, USA, **1969**; pp 157–172.
[53] A. J. George. Computer implementation of the finite element procedure, PhD Dissertation, Stanford University, Stanford, CA, **1971**.
[54] J. R. Gilbert, C. Moler, R. Schreiber, *SIAM J. Matrix Anal. Appl.* **1992**, *13*, 333.
[55] Patrick Amestoy, Maurice Bremond, Alfredo Buttari, A. G.; Guillaume Joslin, Jean-Yves L'Excellent, Francois-Henry Rouet, B.; Weisbecker, U. and C. MUltifrontal Massively Parallel Solver (MUMPS) Version 4.10.0 http://mumps.enseeiht.fr/(accessed Dec 1, **2014**).
[56] A. George, *SIAM J. Numer. Anal.* **1973**, *10*, 345.
[57] J. Poulson. Fast parallel solution of heterogeneous 3D time-harmonic wave equations, PhD. Dissertation, The University of Texas at Austin, **2013**.
[58] A numerical factorization for the BDC64000 example cannot be performed on one core due to memory limitations.
[59] G. M. Amdahl. In *Proceedings of the AFIPS spring joint computer conference SE - AFIPS '67 (Spring)*; **1967**; pp 483–485.
[60] A. Scemama, N. Renon, M. Rapacioli, *J. Chem. Theory Comput.* **2014**, *10*, 2344.
[61] C. L. Janssen, I. M. B. Nielsen. *Parallel Computing in Quantum Chemistry*, 1st ed.; CRC Press, Taylor & Francis Group: Boca Raton, FL, **2008**.
[62] M. Petschow, E. Peise, P. Bientinesi, *SIAM J. Sci. Comput.* **2013**, *35*, C1.
[63] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, N. A. Romero, *ACM Trans. Math. Softw.* **2013**, *39*, 1.
[64] Gutheil, I.; Münchhalfen, J.; Grotendorst, J. In *Parallel Processing and Applied Mathematics SE - 3*; Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J., Eds.; Lecture Notes in Computer Science; Springer Berlin Heidelberg, **2014**; pp 26–35.
[65] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, R. C. Whaley, *Comput. Phys. Commun.* **1996**, *97*, 1.
[66] Due to the large computational cost of loading matrices into Elemental, we couldn't perform calculations with more than 32,000 basis functions.
[67] J. J. P. Stewart, P. Császár, P. Pulay, *J. Comput. Chem.* **1982**, *3*, 227.
[68] D. R. Bowler, T. Miyazaki, *J. Phys. Condens. Matter* **2010**, *22*, 074207.
[69] E. Rudberg, E. H. Rubensson, P. Sałek, *J. Chem. Theory Comput.* **2011**, *7*, 340.
[70] J. VandeVondele, U. Borštnik, J. Hutter, *J. Chem. Theory Comput.* **2012**, *8*, 3565.
[71] E. G. Birgin, J. M. Martínez, L. Martínez, G. B. Rocha, *J. Chem. Theory Comput.* **2013**, *9*, 1043.
[72] E. H. Rubensson, P. Sałek, *J. Comput. Chem.* **2005**, *26*, 1628.
[73] Y. Saad, J. R. Chelikowsky, S. M. Shontz, *SIAM Rev.* **2010**, *52*, 3.
[74] L. Lin, J. Lu, L. Ying, E. Weinan, *Chin. Ann. Math. Ser. B* **2009**, *30*, 729.
[75] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, *E. W. ACM Trans. Math. Softw.* **2011**, *37*, 1.
[76] L. Lin, C. Yang, J. Lu, L. Ying, W. E, *SIAM J. Sci. Comput.* **2011**, *33*, 1329.
[77] L. Lin, M. Chen, C. Yang, L. He, *J. Phys. Condens. Matter* **2013**, *25*, 295501.
[78] G. Schofield, J. R. Chelikowsky, Y. Saad, *Comput. Phys. Commun.* **2012**, *183*, 497.
[79] H. M. Aktulga, C. Yang, E. G. Ng, P. Maris, J. P. Vary, *Concurr. Comput. Pract. Exp.* **2014**, *26*, 2631.
[80] E. L. Briggs, C. T. Kelley, J. Bernholc. Parallel implementation of electronic structure eigensolver using a partitioned folded spectrum method http://arxiv.org/abs/1502.07806 (accessed Mar 7, 2015).

Received: 1 September 2015
Revised: 15 October 2015
Accepted: 25 October 2015
Published online on 17 November 2015