Check for updates

SIESTA-SIPs: Massively Parallel Spectrum-Slicing Eigensolver for an *Ab Initio* Molecular Dynamics Package

Murat Keçeli,*[a,b] Fabiano Corsetti,[c] Carmen Campos,[d] Jose E. Roman,[d] Hong Zhang,[e] Álvaro Vázquez-Mayagoitia,[a,f] Peter Zapol,[g] and Albert F. Wagner[b]

Integration of Shift-and-Invert Parallel Spectral Transformation (SIPs) eigensolver (as implemented in the SLEPc library) into an *ab initio* molecular dynamics package, SIESTA, is described. The effectiveness of the code is demonstrated on applications to polyethylene chains, boron nitride sheets, and bulk water clusters. For problems with the same number of orbitals, the performance of the SLEPc eigensolver depends on the sparsity of the matrices involved, favoring reduced dimensional systems such as polyethylene or boron nitride sheets in comparison to bulk systems like water clusters. For all problems investigated, performance of SIESTA-SIPs exceeds the performance of SIESTA with default solver (ScaLAPACK) at the larger number of cores and the larger number of orbitals. A method that improves the load-balance with each iteration in the self-consistency cycle by exploiting the emerging knowledge of the eigenvalue spectrum is demonstrated. © 2018 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.25350

Introduction

The advent[1] of exascale computing in the early to mid 2020s is expected to dramatically increase the scale and complexity of systems that can be studied in the physical sciences.[2] For atomistic description of large systems, electronic structure calculations will constitute the major fraction of the computational demand on exascale resources.[2] For most electronic structure methods, the matrix form of quantum mechanical equations requires multiple solutions of an eigenvalue problem. This step can be the scaling bottleneck depending on the size of the problem and the approximations involved.[3] For some of the most popular methods, such as Hartree–Fock (HF) and density-functional theory (DFT), a large portion (20–80%) of the eigenvalues is required and the eigenvalue problem must be solved in each iteration of a self-consistent field cycle. The eigenvalue spectrum can have structure, including gaps and tight clusters. The number of eigenvalues scales with the number of electrons in the system leading to 10^4 to 10^5 eigenvalues for systems with 1000's of atoms depending on the chemical composition and the selected basis set.

Dense solvers for eigenvalue problems suffer from cubic scaling in computational time with system size and quadratic scaling in memory while iterative solvers are not efficient in computing large number of eigensolutions. However, one redeeming feature of large systems that is not utilized by these solvers is the decay of interactions with the separation distance between atoms in the system. This leads to a qualitative locality of interactions. There are a variety of DFT methods[4–7] that strictly enforce locality by employing rigorously localized atom-centered basis sets in the expansion of the electronic wavefunction. These results in a sparse Hamiltonian matrix to be diagonalized. HF-based methods can also employ expansions of strictly localized basis functions and in the case of semi-empirical methods,[8,9] other approximations can be used to partially compensate for any errors produced by strict locality.

Sparse matrices and massive parallelism motivate exploration of new kinds of eigensolvers[10–15] that have a small memory footprint and efficient parallelization. In this regard we have been developing the Shift-and-Invert Parallel Spectral Transformation (SIPs) eigensolver, which is very suitable for

[a] *M. Keçeli, A. F. Wagner*
Chemical Sciences and Engineering Division, Argonne National Laboratory, Argonne, Illinois 60439
E-mail: keceli@anl.gov
[b] *M. Keçeli, Á. Vázquez-Mayagoitia*
Computational Science Division, Argonne National Laboratory, Argonne, Illinois 60439
[c] *F. Corsetti*
Departments of Materials and Physics and the Thomas Young Centre for Theory and Simulation of Materials, Imperial College London, London, SW7 2AZ, United Kingdom
[d] *C. Campos, J. E. Roman*
D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Camí de Vera s/n, València, 46022, Spain
[e] *H. Zhang*
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439
[f] *Á. Vázquez-Mayagoitia*
Argonne Leadership Computing Facility, Argonne National Laboratory, Argonne, Illinois 60439
[g] *P. Zapol*
Materials Science Division, Argonne National Laboratory, Argonne, Illinois 60439

Contract grant sponsor: Argonne National Laboratory, and by Basic Energy Science, provided by the Director, Office of Science, of the U.S. Department of Energy; Contract grant number: DE-AC02-06CH11357; Contract grant sponsor: The U.S. Department of Energy Office of Science, Office of Basic Energy Sciences Division of Materials Science and Engineering; Contract grant number: DE-AC02-06CH11357; Contract grant sponsor: The Spanish Ministry of Economy and Competitiveness; Contract grant number: TIN2016-75985-P (including European Commission FEDER funds)

© 2018 Wiley Periodicals, Inc.

quantum chemistry applications.[10,12,15] In the context of DFTB calculations,[16] we have documented[15] the scalability of SIPs for problems with up to 10^5 eigenvalues, its competitive advantages relative to dense eigensolvers[17,18] (less communications and reduced memory), and its performance dependence on the degree of sparsity of the Hamiltonian. Motivated in part by our success with this method, SIPs has been incorporated into SLEPc.[19] With the burden of adapting to new parallel computer architectures now assumed by library developers, SIPs is now available through SLEPc for any electronic structure method that produces sufficiently sparse Hamiltonian matrices.

In this paper, using the SLEPc library, we extend our SIPs performance studies to SIESTA DFT calculations. The SIESTA method[20] uses localized basis sets but unlike DFTB method it treats the Hamiltonian within the common DFT approximations. In this regard, it is similar to other methods (such as FHI-aims[5]) that employ localized basis sets but differ in the details of the basis set construction and application. As such the focus of this paper on SIESTA stands for an example for a class of DFT methods. The open-source SIESTA code is modular with parallelization of Hamiltonian construction but generally used with a dense eigensolver (ScaLAPACK[21]) that becomes the computational bottleneck for large problems. This paper describes our SIESTA-SIPs code that not only provides SIPs as an efficient sparse eigensolver, but systematically and automatically optimizes SIPs performance iteration by iteration in the self-consistency cycle as knowledge of the eigenvalue spectrum improves.

The paper is organized as follows: we describe the main features of SIESTA and of SIPs in the Methods section and then review important features of the SIESTA-SIPs combination in the Implementation section. We describe our illustrative calculations and performance analysis in the Test Calculations and Discussion sections, respectively. In the Conclusions section we summarize and discuss future prospects.

Methods

SIESTA was one of the earliest general-purpose codes to implement self-consistent DFT simulations within a linear-scaling methodology.[22] The basis of the SIESTA method,[23] still used today, consists of:

- norm-conserving pseudopotentials[24] factorized in the separable and non-local Kleinman–Bylander form,[25] with the local part taken as an analytical function optimized for smoothness;
- a non-orthogonal basis set of numerical atomic orbitals (NAOs) of finite range obtained by solving the isolated pseudoatom problem with a soft confinement potential, and including multiple-ζ and polarization orbitals;[4,26–28]
- a real-space grid for representing the electronic density.

This scheme allows for the use of a minimal basis set of reasonable accuracy.[4,27,28] Standard calculations employ a double-ζ basis with a single polarization shell (DZP), for which the fraction of occupied eigenstates is around 20%.

The strictly localized basis orbitals make the Hamiltonian and overlap matrices formally sparse. This means that both the time taken to calculate them and the memory needed to store them scales linearly with the system size. Furthermore, although the density matrix is not sparse, the calculation of the electron density only requires those elements for which the overlap matrix is non-zero. The sparsity of the matrices was exploited by the original linear-scaling solver used by SIESTA,[23] which implements the Kim–Mauri–Galli[29,30] (KMG) functional; the method is based on optimizing a set of localized Wannier-like functions (from which the density matrix is calculated) with respect to the total energy.

Exact diagonalization was later introduced to the code through the LAPACK/ScaLAPACK libraries.[23] Although no longer linear-scaling, the great efficiency and parallel scalability that these provide, as well as the convenience in having access to the eigenvalues, have resulted in diagonalization becoming the default for most applications of SIESTA.[22]

Despite its notable success, ScaLAPACK diagonalization does not exploit any of the distinct characteristics of the SIESTA Hamiltonian, and starts losing efficiency for modern massively parallel architectures (thousands of cores).[3] Therefore, in recent years there has been a concerted effort to investigate and implement a variety of new solvers in the code, using ScaLAPACK as a benchmark against which to measure their performance. So far, these have included a new implementation of the orbital-minimization method[13] (OMM, related to KMG), a pole expansion with selected inversion method[31,32] (PEXSI), and a highly parallelizable two-step eigenvalue solver[3,33] (ELPA).

Exact diagonalization methods solve the generalized eigenvalue problem given by,

$$\boldsymbol{H}\boldsymbol{c_i} = \lambda_i \boldsymbol{S}\boldsymbol{c_i}. \tag{1}$$

In this equation, $\boldsymbol{H}$ is a real, symmetric Hamiltonian matrix, S is a real, symmetric, and positive definite overlap matrix of N non-orthogonal basis functions, $\boldsymbol{c_i}$ is the eigenvector containing the coefficients of the atomic basis functions, and λ_i is the corresponding eigenvalue for state i. The localization of the atomic orbitals makes $\boldsymbol{H}$ and $\boldsymbol{S}$ matrices typically sparse for large systems. After the eigensolutions are obtained, the density matrix has to be computed to update $\boldsymbol{H}$ until self-consistency is achieved. The density matrix, $\boldsymbol{Q}$, and energy density matrix, $\boldsymbol{P}$, are computed by the following formulas:

$$\boldsymbol{Q} = \sum_{i=1}^{n_{\mathrm{req}}} w_i |\boldsymbol{c_i}\rangle\langle\boldsymbol{c_i}|, \tag{2}$$

$$\boldsymbol{P} = \sum_{i=1}^{n_{\mathrm{req}}} w_i \lambda_i |\boldsymbol{c_i}\rangle\langle\boldsymbol{c_i}|, \tag{3}$$

where the number of required eigenvalues, n_{req} and weight factors, w_i, are computed by SIESTA depending on the number of electrons and the electronic temperature, which can be set by the user.

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

An alternative to the explicit diagonalization method implemented in ScaLAPACK is the spectrum slicing technique.[34] This method allows the computation of all eigenvalues (and corresponding eigenvectors) contained in a given interval, and hence has potentially less cost than methods that obtain the whole spectrum. The main building block is the Shift-and-Invert Lanczos method, which is combined with a dynamic shift selection that sweeps the interval in a smart way. Shift-and-Invert Lanczos operates with matrix $(\boldsymbol{H}-\sigma_1\boldsymbol{S})^{-1}\boldsymbol{S}$ to compute a few eigenvalues closest to the shift σ_1, and a byproduct of this computation is the inertia at this shift, which corresponds to the number of eigenvalues lying on the left of σ_1. When repeated with a different shift σ_2, the number of eigenvalues contained in the interval $[\sigma_1, \sigma_2]$ is then known, and this information can be used to decide whether this interval has been processed completely or not. Further shifts will extend the confidence interval until the user-requested interval has been totally covered. This technique relies on the computation of a symmetric-indefinite triangular factorization of matrix $\boldsymbol{H}-\sigma_1\boldsymbol{S}$, so it is important to minimize the number of such factorizations by carefully selecting the shifts. Another important ingredient is deflation of eigenvectors previously computed in the neighborhood of the current shift, which prevents the appearance of spurious duplicate eigenvalues, and also enforces the orthogonality of eigenvectors computed from different shifts.

We use the spectrum slicing method as implemented in SLEPc, the Scalable Library for Eigenvalue Problem Computations.[19] SLEPc is a freely available software package that provides a number of MPI-parallel eigensolvers, including thick-restart Lanczos. It relies on several components of PETSc[35] (Portable, Extensible Toolkit for Scientific Computation), in particular the data classes to manipulate vectors and matrices, as well as the linear solvers. For the parallel factorizations required in the spectrum slicing technique, PETSc integrates third-party libraries such as MUMPS.[36]

Implementation

The above description of the spectrum slicing technique assumes that a number of shifts are processed sequentially, one after the other, and all computations associated with each shift are done in parallel (Lanczos iterations, matrix factorization). For intervals that contain many eigenvalues, we could consider splitting the interval up front, and running the spectrum slicing method independently on each of the subintervals (slices). This represents a two-level hierarchical parallelism that will enable better scalability to larger numbers of processors. Despite being a very simple idea, its efficient implementation requires taking care of issues such as how to choose the subintervals (to be discussed later on) or how to manage the two levels of parallelism. We now briefly describe the approach used in SLEPc, which is an improvement on previous work.[12] The solver object receives matrices **H** and **S**, which have been defined in a given MPI communicator, as well as the interval of interest. Then the communicator is split internally into as many partitions as slices that have been

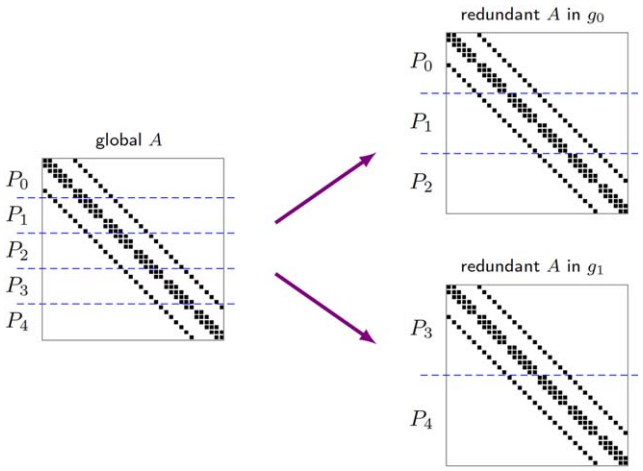

Figure 1. Parallel redistribution of matrices in the case of two subcommunicators g_0 and g_1, where the parent communicator has five processes. [Color figure can be viewed at wileyonlinelibrary.com]

requested for the interval. Each subcommunicator will process one slice. A schematic of the parallel redistribution of matrices for a simple case is given in Figure 1. In order for this multi-communicator approach to work correctly, a redundant copy of the **H** and **S** matrices is created on each subcommunicator, so that the spectrum slicing algorithm can be invoked. SLEPc provides user interface functions to manipulate the matrices (as well as the eigenvectors) in both the parent and child communicators, to avoid unnecessary data redistribution in the context of self-consistent iteration loops. This scheme has the additional benefit that symbolic factorizations can be reused (assuming that the sparsity pattern of the matrices does not change from one call to the next). Symbolic factorization is the first step of factorization followed by numerical factorization, which cannot be reused. More information on these two steps can be found in the Supporting Information for Ref. [15].

Another important consideration for the selection of slice boundaries is that eigenvalues should not coincide with any of the boundaries, since it results in singular matrices when a shift is performed. Such cases are detected by SLEPc at the factorization stage, using the relevant option from the MUMPS package. When a numerically singular matrix is detected, the shift is perturbed slightly and the factorization is repeated by SLEPc.

Thanks to modular structure of the SIESTA code, the availability of Fortran wrappers in PETSc and SLEPc libraries, and the common parallelization interface, MPI, integration of the eigensolver was straightforward. We can describe the integration process in terms of four main tasks.

The first task, which is common in all PETSc/SLEPc applications, is to replace MPI_Init() and MPI_Finalize() calls in the application code with corresponding initialize and finalize statements, i.e. SLEPcInitialize() and SLEPcFinalize() for SLEPc applications, since the latter routines calls the former automatically and this allows both PETSc, SLEPc and SIESTA to share a common parallel environment, i.e. MPI_COMM_WORLD.

The second task is to pass matrix elements from SIESTA to PETSc to create overlap and Hamiltonian matrices in PETSc

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

format. SIESTA uses block cyclic distribution for matrices in a format similar to compressed sparse row (CSR) storage. However, it is converted into dense format since it needs to be passed into ScaLAPACK. PETSc uses a scheme similar to CSR and allows insertion of matrix elements from any processors; hence it is compatible with any type of distribution in principle.

The third task is to create the Eigenvalue Problem Solver (EPS) object in SLEPc using the overlap and Hamiltonian matrices and solve the generalized eigenvalue problem. While there are many different solvers available in SLEPc the most appropriate one is the multi-communicator spectrum-slicing eigensolver since a large portion of eigensolutions are required. We note that this method requires transformation of the spectrum, which is handled by the spectral transformation object (ST) in SLEPc.

The fourth task is to return density and energy density matrix elements, which is computed by using the eigenvectors obtained in the third task [see eqs. (2) and (3)].

While implementation of these four tasks is sufficient for the integration of a new solver in SIESTA, it is crucial to perform them efficiently.

Improving Efficiency

Task 2 requires less computational resources compared to Task 3 and Task 4, however it may require significant data movement if not implemented carefully. We note that PETSc does not use cyclic distribution, so we need to choose a block size for SIESTA such that each process is responsible for a single block of adjacent rows. To achieve this, we use a maximum block size, b = ceiling (N/p), where N and p are numbers of rows and MPI processes, respectively, and ceiling (x) function returns the smallest integer larger than x. When N/p is an integer, one gets a perfect load balance with the given block size, but when it is not an integer, there will be a slight imbalance. Currently, a user can change this value only by changing the number of processors, since it is determined internally. Another factor that improves efficiency is to preallocate the matrix, so that a minimum number of "malloc" (memory allocation) operations are performed during matrix assembly. In SIESTA-SIPs one can specify the matrix type in run time through command line options, and we use PETSc MPISBAIJ format to exploit symmetry in the solution stage since all matrices involved here are symmetric. However, we note that SIESTA does not exploit the symmetry of the matrices in other stages of the calculation. Another note for Task 2 is that overlap matrices and sparsity patterns do not change during SCF iterations, meaning the overlap matrix needs to be created only at the beginning of the SCF cycles. All matrices are destroyed when SCF is converged. For a new molecular dynamics step, matrices should be recreated since the sparsity pattern might be different at each step.

Task 3 is the most demanding part of the computation. The boundaries of the slices, which set the search interval for eigenvalues, influence the efficiency of this task significantly. While the user might have a good idea about the global range for the search interval of the eigenvalues, the eigenvalue distribution is not generally known prior to the calculation. Using uniform width slices can result in poor load balance since eigenvalue distribution is generally not uniform and includes gaps. Hence, it is very important to have information on eigenvalue distribution and adjust slice boundaries such that load is distributed as evenly as possible. There are two different methods implemented in SIESTA-SIPs to obtain information on the eigenvalue distribution. The first method is based on inertia calculations performed at uniform width slice boundaries prior to the solution stage. We note that inertia calculations give information only about the number of eigenvalues at the specified points, thus the information is very crude when the number of slices is low. However, as the number of slices increases, inertias give a very good estimate of the eigenvalue distribution. We also note that inertia calculations are always performed at slice boundaries to ensure all the eigenvalues within a slice are computed, however, if slices are adjusted after inertia calculations, they need to be recomputed at new slice boundaries. The second method is using the eigenvalues from the previous iteration as an estimate for the subsequent iteration. This information is useful only when eigenvalue distribution ceases to change significantly in SCF iterations. We found that this generally happens after the first 3–5 iterations for systems with good SCF convergence behavior. Users can set a threshold for the absolute maximum difference between the eigenvalues of subsequent iterations to define when SIESTA-SIPs switches to this method instead of performing inertia calculations.

After an estimate of the eigenvalue distribution is obtained with either method, we need to partition N_e eigenvalues into N_s slices such that load is distributed as uniformly as possible. We determined three objectives that can help achieve this goal.

1. Eigenvalues separated by less than a threshold value, e.g. 10^{-6}, should be kept within the same slice. This is not only a performance requirement but also a requirement to ensure the orthogonality of the eigenvectors for such eigenvalues.
2. The number of eigenvalues in each slice should be close to the average number of eigenvalues per slice.
3. The eigenvalue range (the distance between the minimum and the maximum eigenvalues) within each slice should be minimized.

While it is a simple task to determine the eigenvalues that are close to each other as required by the first objective, satisfying the second and third objectives simultaneously is often difficult. One exception is for $N_e \leq N_s$. The trivial solution is to put each eigenvalue cluster as defined in objective 1 into a separate slice. However, this won't be an efficient usage of resources, since convergence of a single eigenvalue and few tens of eigenvalues have similar costs. Hence, once the average number of eigenvalues per slice is around 10, it is more advantageous to increase the number of cores per slice, rather than increasing the number of slices. However, for the most

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Table 1. Information for the given examples.

Label	Number of atoms	Number of rows	Nonzero density %[a]	Nonzero density %[b]
P64	384	2944	7.1	15.6
P256	1536	11,776	1.8	4.0
B64	128	1664	37.0	94.6
B256	512	6656	9.3	47.1
W128	384	2944	9.4	66.0
W512	1536	11,776	2.3	33.2

[a] Nonzero density percentages of the Hamiltonian and overlap matrices. [b] Nonzero density percentages of the matrices after the factorization.

common case with $N_e > N_s$, finding the optimum arrangement of slices is a difficult task.

Heuristic cluster analysis algorithms, such as *k*-means clustering, can be instrumental in finding the optimum slicing of the eigenvalues. The objective of *k*-means clustering is to partition a set of data points into a given number of clusters by minimizing the within-cluster sum of square distances, which is similar to objective 3. We note that *k*-means clustering guarantees only a local minimum and the result depends on the initial partition of the eigenvalues. In our *k*-means implementation, we start from uniformly populated partitions of eigenvalues.

We tried different algorithms for slicing and have improved the load balance by more than a factor of two compared to using uniform width slices as discussed in the following section. However, we note that there is no guarantee that we can achieve the best load balance for different eigenvalue distributions and different number of slices. A delicate computational cost model that takes into account different stages of the solution (factorization, Lanczos scheme, orthogonalization) is required to improve the efficiency further.[14] Approximate techniques to estimate the eigenvalue distribution[37] will be also very useful to eliminate the cost of additional inertia calculations.

Test Calculations

We have performed test calculations to assess the accuracy and performance of SIESTA-SIPs implementation. We ran calculations both with the default solver (ScaLAPACK) and the SLEPc solver SIPs for all the test cases. We do not observe any numerical difference in the results as expected since there are no approximations involved except the fact that SLEPc solver uses an iterative scheme (Lanczos) to find eigensolutions while ScaLAPACK is a direct solver.

We performed test calculations on BG/Q supercomputers (Vesta and Mira) at Argonne National Laboratory. We choose different sizes of polyethylene, boron nitride monolayer, and water clusters as our test cases representing 1D, 2D, and 3D systems. We label the polyethylene, boron nitride, and water cases by Pn, Bn, and Wn, respectively, where *n* is the number of formula units. Table 1 provides the information needed to relate "*n*" to the number of atoms, number of rows of the Hamiltonian and overlap matrices, and sparsity of these matrices. Here, a measure for the sparsities of the matrices are given by the nonzero density percentages, which are obtained by dividing the number of nonzeros by the total number of elements in the lower half of a matrix. We note that our test calculations have fairly dense matrices after factorization, especially for two- and three-dimensional systems.

In Figure 2, we compare strong scaling of SIPs and ScaLAPACK solvers for polyethylene and water cluster examples. We obtained converged energy and gradient after 13 SCF iterations, and we plot the average time for a single SCF iteration. Error bars show the spread of SCF iteration timings with SIESTA-SIPs, since the iteration time can differ significantly with

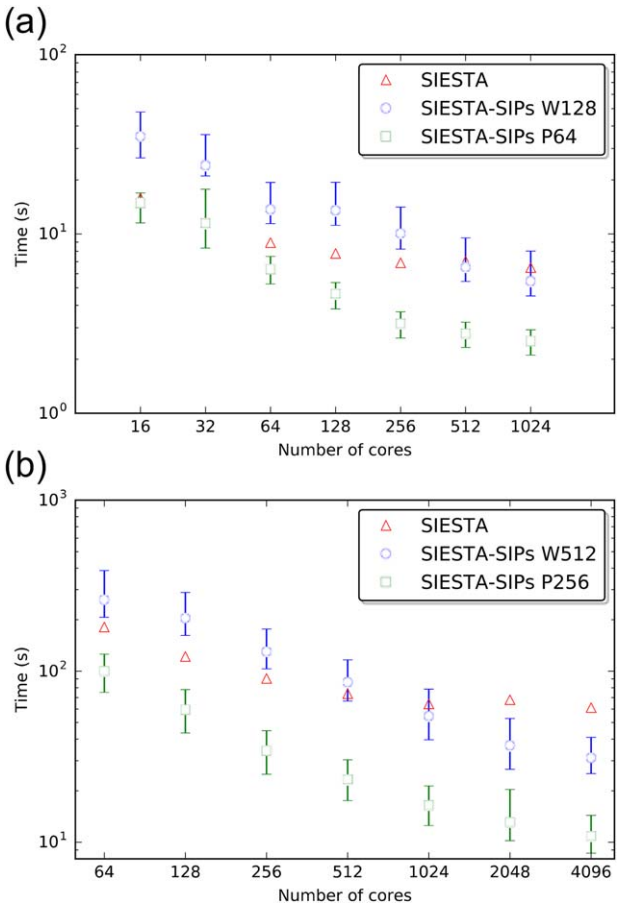

Figure 2. Strong scaling plot for the average time (seconds) of an SCF iteration obtained with SIESTA-SIPs and ScaLAPACK solvers, where DFT energy and gradient is calculated. Error bar shows the spread of SCF iterations with SIESTA-SIPs. Figure a) shows calculations with 384 atoms and 2944 orbitals (W128 and P64), while Figure b) shows calculations with 1536 atoms and 11,775 orbitals (W512 and P256). [Color figure can be viewed at wileyonlinelibrary.com]

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

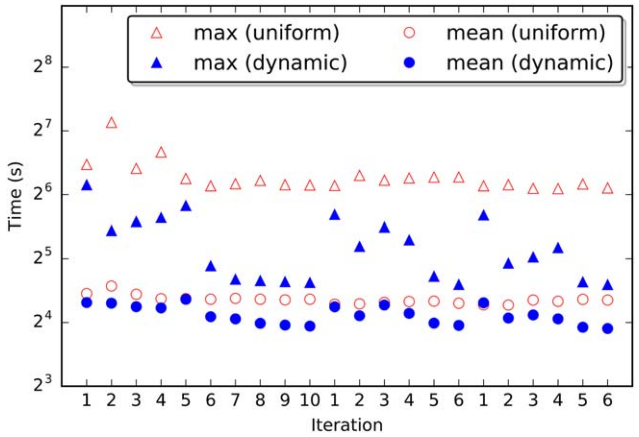

Figure 3. Plot of maximum (shown with triangle symbols) and mean (shown with circle symbols) time of the eigensolutions stage with SIESTA-SIPs for all slices at each SCF iteration numerated on the horizontal axis. Three molecular dynamics steps (consisting of 10, 6, and 6 SCF iterations) leading to a relaxed geometry for B64 example using 256 cores and 64 slices are shown. Unfilled red symbols refer to times obtained with uniform width slices, and filled blue markers refer to times obtained with dynamically adjusted slices. [Color figure can be viewed at wileyonlinelibrary.com]

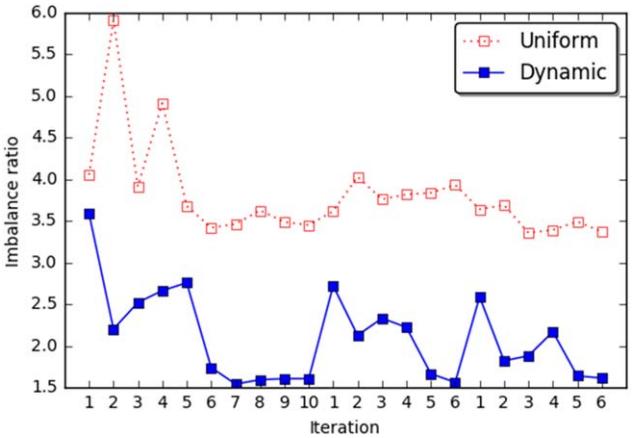

Figure 4. Plot of imbalance ratio (t_{max}/t_{mean}) of the workload on each slice for the computation of eigensolutions for the same case as Figure 3. Results of uniform width and dynamically adjusted slices are shown with unfilled and filled markers, respectively. [Color figure can be viewed at wileyonlinelibrary.com]

each iteration depending on the slicing of the eigenvalue distribution. The iteration time for ScaLAPACK is dictated by mainly the size of the matrix and is independent from the iteration or the system characteristics, hence there is a single data point in these plots for both test cases (because they are isoelectronic) and also there is no need for error bars. However for SIESTA-SIPs, iteration time reduces by a factor of 4 for the 1D problem compared to the 3D problem. For both problems, the spread of iteration times can reach a factor of 2. Figure 2 shows that with sufficient parallelization, SIPs is faster than dense solvers by a degree that increases with the size of the system.

In Figure 3, we show how the mean and max iteration time per slice changes for a 3-step conjugate gradient minimization calculations with a total of 22 SCF iterations for the B64 example with 64 slices. The figure displays two different slicing strategies. The first is uniform slicing, where the slices have uniform width and the boundaries are kept the same throughout the calculations. In the second approach (dynamic), we adjust the slice boundaries at every iteration based on either inertia calculations or the eigenvalue distribution of the previous iteration. We perform inertia calculations in first step for the first two SCF iterations and for all subsequent iterations whose absolute maximum difference between eigenvalues of the two previous iterations exceeds a threshold of 0.001. Once this difference drops below the threshold, *k*-means clustering is used for all iterations to convergence. In subsequent steps, we do inertia calculations for the first iteration and apply the threshold test for differences between that iteration and the last iteration of the previous step. Figure 4 expresses load balance as the ratio of the max and mean times in Figure 3. As shown in Figure 4, after the first few iterations, uniform slicing results in a steady imbalance ratio of ~3.5. Dynamic slicing always results in a lower imbalance ratio that is highest as the start of a step but for the last iterations hovers around ~1.5. From the figure, the difference in the load imbalance ratio from the ideal, i.e., 1, is reduced by a factor of 7 for the last iterations of a step and by a factor of 2–3 overall.

In Figure 5, we show the eigenvalue distribution, slice boundaries, and the points for inertia calculations for each SCF iteration for the same boron nitride case in Figures 3 and 4. Sixteen slices are used in the figure for clarity. Since the eigenvalues in each slice of the eigenvalue spectrum are calculated independently and in parallel, we show how improved knowledge of the eigenvalue spectrum with each iteration can translate into altered slice boundaries that even out the computational cost in each slice for better load balance. This also applies to adjacent geometries in a trajectory calculation or a geometry optimization. As shown in Figure 4 for steps two and three, the threshold test of eigenvalue differences between the first iteration in the step and the last iteration in

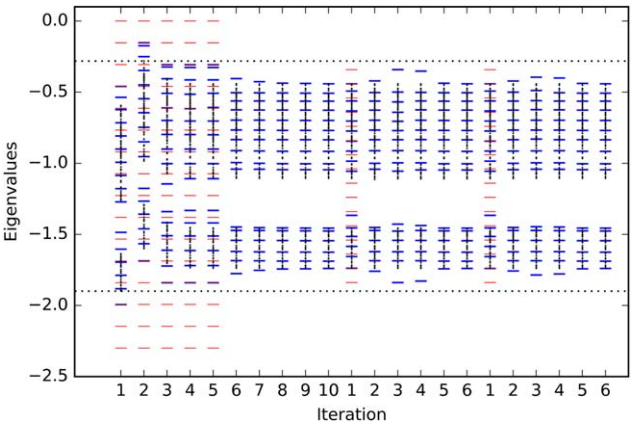

Figure 5. Plot of 1024 eigenvalues (black points) for structural relaxation example in Figure 3 only with 16 slices (blue lines). Black dotted lines indicate the minimum and maximum eigenvalues among all iterations. Red lines denote the points, where an inertia calculation is performed to arrange the slice edges. [Color figure can be viewed at wileyonlinelibrary.com]

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

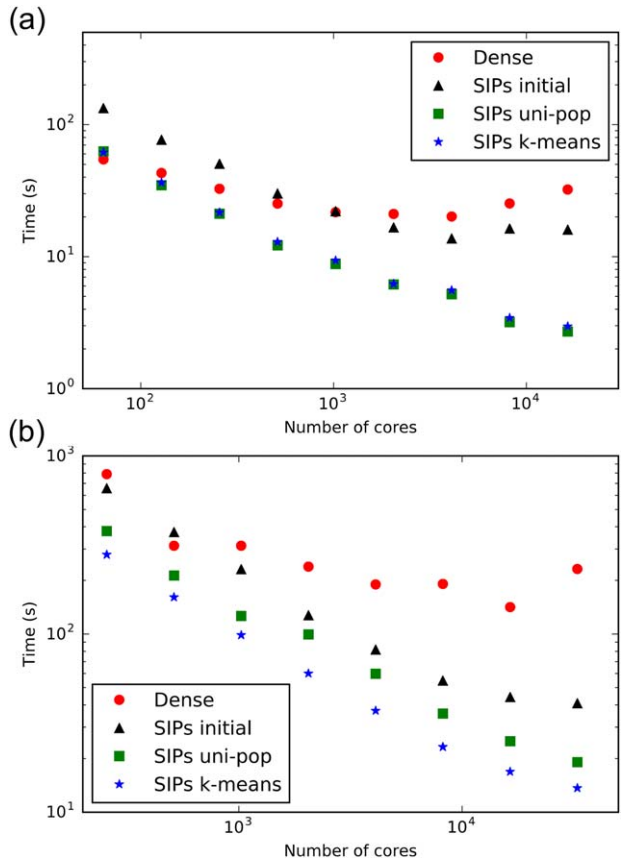

Figure 6. Strong scaling plot for the computation of 50% of eigenpairs using SIPs eigensolver in three different modes and a dense eigensolver based on ELSI-ELPA. Figures a) and b) show the results for 16,000 and 64,000 row matrices obtained from DFTB calculations in our previous study,[15] respectively. [Color figure can be viewed at wileyonlinelibrary.com]

the previous step turns on the more efficient *k*-means clustering for all subsequent iterations in the step.

We have also investigated the performance of SIPs as implemented in SLEPc directly and compared it with a more recently developed dense eigensolver ELPA[33] using the ELSI interface.[38] ELSI interface is required to convert the generalized eigenvalue problem to the standard eigenvalue problem. We note that we used ELPA version 2016.11 and there might be improvements in performance of ELPA with more recent versions and by accessing it directly without ELSI interface. We utilized the matrices obtained from our previous study[15] on DFTB calculations for metallic carbon nanotubes. These matrices have 16,000 and 64,000 rows and have nonzero density of 6.45% and 1.65% after factorization, respectively. We run these calculations on Theta, which is an Intel Xeon Phi based supercomputer located at Argonne Leadership Computing Facility. Figure 6 shows the strong scaling plots for computing 50% of the eigenpairs using ELPA and SLEPc-SIPs. In these plots we also compare three different slicing modes for SIPs. "SIPs initial" corresponds to the uniform slicing where we have no information about the eigenvalue spectrum. In this case, slices have uniform width based on the given search interval. "SIPs uni-pop" and "SIPs *k*-means" correspond to two different dynamic slicing strategies where the eigenvalue spectrum is known and slices are arranged to reduce the load imbalance. The former one refers to the strategy, which tries to balance the number of eigenvalues in each slice, while the latter one refers to the strategy, which uses the *k*-means clustering algorithm to find the clusters of eigenvalues and then adjust slices to avoid division of a cluster into different slices. For the smaller matrix with 16,000 rows, we see that the dense eigensolver is faster than SIPs when the number of cores utilized is less than 100. As we increase the number of cores, we see that SIPs exceeds the performance of the dense eigensolver even when there is no information about the eigenvalue spectrum. Time to solution with SIPs reduces down to 2.7 sec while the strong scaling limit of the dense eigensolver is 20.1 sec. For the larger matrix with 64,000 rows, performance of SLEPc-SIPs is better than the dense eigensolver even with 256 cores. Utilizing 32,768 cores and taking advantage of the sparsity of the matrices, time to solution is reduced by more than a factor of ten when SIPs is used in comparison to the dense eigensolver. We also see that *k*-means clustering algorithm works more effectively in comparison to equally populated slices, since the eigenvalues become more clustered as the problem size increases.

Another important parameter that affects the performance is the number of cores per slice. While the optimum choice depends on the problem type and hardware specifics, we found the following strategy to be very effective in these calculations. We use one core per slice as long as the memory per core is sufficient to store the matrix and the number of eigenvalues per slice is above ~10. Once we reach this limit, we increase the number of cores per slice proportional to the increase in the total number of cores. We also note that strong scaling efficiency may drop considerably if the number of cores per slice exceeds the number of cores per node on machines with slow interconnects. At that point increasing the number of slices further might be a better choice compared to increasing the number of cores per slice.

Discussion

The effectiveness of the SIESTA-SIPs code was demonstrated in test calculations on polyethylene chains, boron nitride sheets, and bulk water clusters. While the examples presented for SIESTA-SIPs do not include metallic systems, the eigensolver performance is not dependent on the presence of a band gap in the energy spectrum as shown on our calculations on metallic carbon nanotubes.[15]

For problems with the same number of orbitals, the performance of SIPs depends on the sparsity of the matrices involved, favoring reduced dimensional systems such as polyethylene or boron nitride sheets in comparison to bulk systems like water clusters. For problems of different size, the cross over point between any dense solver and SIPs decreases with the system size since SIPs has better scaling. This implies that SIESTA-SIPs will demonstrate improved performance over dense eigensolvers for problems significantly larger than what has been done here. We also note that while the results

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

reported here are limited to modest number of cores and system sizes, the performance of current implementation of SIESTA-SIPs is not generally limited by the diagonalization procedures. We observed that other parts of the calculation, i.e. fast Fourier transform during the initial assembly of the Hamiltonian can become the bottleneck of scalability as the number of cores and/or number of orbitals increase.

Conclusions

We report three significant developments for the SIPs method.

1. Different algorithms for slicing are developed, implemented and tested. These algorithms improve the load balance significantly for the final iterations of the SCF method.
2. A new implementation of the SIPs method is incorporated into the SLEPc package by two of the authors (JER and CC). The SLEPc implementation offers various advantages:
   - Well-documented, consistent and intuitive API with a BSD licensed open-source code base.
   - Active user support and maintenance with regular release cycles.
   - Access to other eigensolvers through command line options.
3. SLEPc-SIPs is integrated into SIESTA allowing users to utilize the SIPs solver in a scientific application directly.

Performance of SIPs depends on the sparsity of the matrices involved, hence, it is more efficient for reduced dimensional systems that yield sparser matrices. Periodic boundary condition calculations on polyethylene (1D), boron nitride sheets (2D), and bulk water clusters (3D) bear this out with approximately a factor of four variation in time to solution between polyethylene (fastest) and water clusters (slowest) problems with the same number of orbitals. We demonstrate the ability of SIPs to improve its load balancing with successive iterations in the SCF scheme that results in better load-balance and superior performance.

Quite recently SIPs has been incorporated into ELSI,[38] which also includes advanced dense eigensolvers such as ELPA[33] and a variety of novel solvers (such as OMM[39] and PEXSI[13,31]) that avoid the diagonalization step altogether. ELSI simplifies the integration of solvers into electronic structure codes by providing a common interface. ELSI will allow us to better define the cross over points in performance between different methods as a function of both problem size and dimensionality and available computer resources.

We finally note that SIESTA-SIPs code is available online and all the dependencies of the code can be obtained freely.[40]

Acknowledgment

We are grateful to the Argonne Leadership Computing Facility for computing resources. 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. This material is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.

Keywords: DFT · SCF · eigensolver · sparse · *ab initio*

How to cite this article: M. Keçeli, F. Corsetti, C. Campos, J. E. Roman, H. Zhang, Á. Vázquez-Mayagoitia, P. Zapol, A. F. Wagner. *J. Comput. Chem*. **2018**, *39*, 1806–1814. DOI: 10.1002/jcc.25350

[1] National Strategic Computing Initiative Strategic Plan, www.whitehouse.gov/sites/whitehouse.gov/files/images/NSCI Strategic Plan.pdf (accessed Jan 1, **2016**).
[2] Basic Energy Sciences Exascale Requirements Review, https://science.energy.gov/~/media/bes/pdf/reports/2017/BES-EXA_rpt.pdf (accessed Jan 1, **2015**)
[3] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, P. R. Willems, *Parallel Comput.* **2011**, *37*, 783.
[4] E. Anglada, J. M. Soler, J. Junquera, E. Artacho, *Phys. Rev. B* **2002**, *66*, 205101.
[5] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, M. Scheffler, *Comput. Phys. Commun.* **2009**, *180*, 2175.
[6] D. Porezag, T. Frauenheim, T. Köhler, G. Seifert, R. Kaschner, *Phys. Rev. B* **1995**, *51*, 12947.
[7] G. Seifert, *J. Phys. Chem. A* **2007**, *111*, 5609.
[8] J. J. P. Stewart, *Reviews in Computational Chemistry*; John Wiley & Sons, Inc.: New York, **1990**; pp. 45–81.
[9] W. Thiel, *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2014**, *4*, 145.
[10] H. Zhang, B. Smith, M. Sternberg, P. Zapol, *ACM Trans. Math. Softw.* **2007**, *33*, 9.
[11] G. Schofield, J. R. Chelikowsky, Y. Saad, *Comput. Phys. Commun.* **2012**, *183*, 497.
[12] C. Campos, J. E. Román, *Numer. Algorithms* **2012**, *60*, 279.
[13] L. Lin, J. Lu, L. Ying, E. Weinan, *Chin. Ann. Math. Ser. B* **2009**, *30*, 729.
[14] H. M. Aktulga, L. Lin, C. Haine, E. G. Ng, C. Yang, *Parallel Comput.* **2014**, *40*, 195.
[15] M. Keçeli, H. Zhang, P. Zapol, D. A. Dixon, A. F. Wagner, *J. Comput. Chem.* **2016**, *37*, 448.
[16] M. Elstner, G. Seifert, *Philos. Trans. R. Soc. A* **2014**, *372*, 20120483.
[17] M. Petschow, E. Peise, P. Bientinesi, *SIAM J. Sci. Comput.* **2013**, *35*, C1.
[18] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, N. A. Romero, *ACM Trans. Math. Softw.* **2013**, *39*, 1.
[19] V. Hernandez, J. E. Román, V. Vidal, *ACM Trans. Math. Softw.* **2005**, *31*, 351.
[20] E. Artacho, E. Anglada, O. Diéguez, J. D. Gale, A. García, J. Junquera, R. M. Martin, P. Ordejón, J. M. Pruneda, D. Sánchez-Portal, J. M. Soler, *J. Phys. Condens. Matter* **2008**, *20*, 64208.
[21] 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.
[22] D. R. Bowler, T. Miyazaki, *Rep. Prog. Phys.* **2011**, *36503*, 84.
[23] J. M. Soler, E. Artacho, J. D. Gale, A. Garcìa, J. Junquera, P. Ordejòn, D. Sànchez-Portal, *J. Phys. Condens. Matter* **2002**, *14*, 2745.
[24] N. Troullier, J. L. Martins, *Phys. Rev. B* **1991**, *43*, 8861.
[25] L. Kleinman, D. M. Bylander, *Phys. Rev. Lett.* **1982**, *48*, 1425.
[26] E. Artacho, D. Sanchez-Portal, P. Ordejon, A. Garcia, J. M. Soler, *Phys. Stat. Sol.* **1999**, *215*, 809.
[27] J. Junquera, Ó. Paz, D. Sánchez-Portal, E. Artacho, *Phys. Rev. B* **2001**, *64*, 235111.

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

[28] F. Corsetti, M.-V. Fernández-Serra, J. M. Soler, E. Artacho, *J. Phys. Condens. Matter* **2013**, *25*, 435504.
[29] P. Ordejón, D. A. Drabold, R. M. Martin, M. P. Grumbach, *Phys. Rev. B* **1995**, *51*, 1456.
[30] J. Kim, F. Mauri, G. Galli, *Phys. Rev. B* **1995**, *52*, 1640.
[31] L. Lin, M. Chen, C. Yang, L. He, *J. Phys. Condens. Matter* **2013**, *25*, 295501.
[32] L. Lin, A. García, G. Huhs, C. Yang, *J. Phys. Condens. Matter* **2014**, *26*, 305503.
[33] 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.
[34] T. Ericsson, A. Ruhe, *Math. Comput.* **1980**, *35*, 1251.
[35] 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**.
[36] P. Amestoy, M. Bremond, A. G. Alfredo Buttari, G. Joslin, J.-Y. L'Excellent, B. Francois-Henry Rouet, U. Weisbecker, C. MUltifrontal Massively Parallel Solver (MUMPS) Version 4.10.0 http://mumps.enseeiht.fr/(accessed Dec 1, **2014**).
[37] L. Lin, Y. Saad, C. Yang, *SIAM Rev.* **2016**, *58*, 34.
[38] V. W. Yu, F. Corsetti, A. García, W. P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, Á. Vázquez-Mayagoitia, C. Yang, H. Yang, V. Blum, http://arxiv.org/abs/1705.11191, **2017**, p. 1.
[39] F. Corsetti, *Comput. Phys. Commun.* **2014**, *185*, 873.
[40] M. Keçeli, H. Zhang, C. Campos, J. E. Roman, SIESTA-SIPs, https://bitbucket.org/keceli/siesta-sips

Received: 23 December 2017
Revised: 23 March 2018
Accepted: 12 April 2018

1096987x, 2018, 22, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/jcc.25350 by Argonne National Laboratory, Wiley Online Library on [02/02/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License