Computer Physics Communications 222 (2018) 267–285

Contents lists available at ScienceDirect

Computer Physics Communications

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

ELSI: A unified software interface for Kohn–Sham electronic structure solvers☆

Victor Wen-zhe Yu [a], Fabiano Corsetti [b], Alberto García [c], William P. Huhn [a], Mathias Jacquelin [d], Weile Jia [d,e], Björn Lange [a], Lin Lin [d,e], Jianfeng Lu [f], Wenhui Mi [a], Ali Seifitokaldani [a], Álvaro Vázquez-Mayagoitia [g], Chao Yang [d], Haizhao Yang [f], Volker Blum [a,*]

[a] Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27707, United States
[b] Departments of Materials and Physics, and the Thomas Young Centre for Theory and Simulation of Materials, Imperial College London, London SW7 2AZ, United Kingdom
[c] Institut de Ciència de Materials de Barcelona (ICMAB-CSIC), Bellaterra E-08193, Spain
[d] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, United States
[e] Department of Mathematics, University of California, Berkeley, CA 94720, United States
[f] Department of Mathematics, Duke University, Durham, NC 27707, United States
[g] Argonne Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439, United States

ARTICLE INFO

Article history:
Received 1 June 2017
Received in revised form 28 August 2017
Accepted 4 September 2017
Available online 15 September 2017

Keywords:
Density-functional theory
Kohn–Sham eigenvalue problem
Parallel computing

ABSTRACT

Solving the electronic structure from a generalized or standard eigenproblem is often the bottleneck in large scale calculations based on Kohn–Sham density-functional theory. This problem must be addressed by essentially all current electronic structure codes, based on similar matrix expressions, and by high-performance computation. We here present a unified software interface, ELSI, to access different strategies that address the Kohn–Sham eigenvalue problem. Currently supported algorithms include the dense generalized eigensolver library ELPA, the orbital minimization method implemented in libOMM, and the pole expansion and selected inversion (PEXSI) approach with lower computational complexity for semilocal density functionals. The ELSI interface aims to simplify the implementation and optimal use of the different strategies, by offering (a) a unified software framework designed for the electronic structure solvers in Kohn–Sham density-functional theory; (b) reasonable default parameters for a chosen solver; (c) automatic conversion between input and internal working matrix formats, and in the future (d) recommendation of the optimal solver depending on the specific problem. Comparative benchmarks are shown for system sizes up to 11,520 atoms (172,800 basis functions) on distributed memory supercomputing architectures.

Program summary

Program title: ELSI Interface
Program Files doi: http://dx.doi.org/10.17632/y8vzhzdm62.1
Licensing provisions: BSD 3-clause
Programming language: Fortran 2003, with interface to C/C++
External routines/libraries: MPI, BLAS, LAPACK, ScaLAPACK, ELPA, libOMM, PEXSI, ParMETIS, SuperLU_DIST
Nature of problem: Solving the electronic structure from a generalized or standard eigenvalue problem in calculations based on Kohn–Sham density functional theory (KS-DFT).
Solution method: To connect the KS-DFT codes and the KS electronic structure solvers, ELSI provides a unified software interface with reasonable default parameters, hierarchical control over the interface and the solvers, and automatic conversions between input and internal working matrix formats. Supported solvers are: ELPA (dense generalized eigensolver), libOMM (orbital minimization method), and PEXSI (pole expansion and selected inversion method).
Restrictions: The ELSI interface requires complete information of the Hamiltonian matrix.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: volker.blum@duke.edu (V. Blum).

http://dx.doi.org/10.1016/j.cpc.2017.09.007
0010-4655/

1. Introduction

Molecular and materials simulations based on Kohn–Sham (KS) [1] and generalized Kohn–Sham (gKS) [2,3] density-functional theory (DFT) are widely used to provide atomic-scale insights, understanding, and predictions across a wide range of disciplines in the sciences and in engineering. The number of DFT-related publications has grown rapidly over recent decades [4–6], exceeding 20,000 in 2016 [6]. In particular, simulations based on semilocal and hybrid density functionals serve as the production workhorses for a broad range of applications. Advances in both computational methods and high-performance computing hardware render it feasible to model large systems consisting of thousands of atoms, and linear scaling KS-DFT [7–9] can reach system sizes of millions of atoms [10,11]. Higher levels of density functional approximations, like the Random Phase Approximation (RPA), can be formulated to scale linearly with system size as well [12,13].

However, approaches for which the computational effort scales lower than $O(N^3)$, where N is some measure of the system size, are, arguably, not yet fully established as mainstream methods of the field. There are several reasons for this status. Perhaps the simplest reason is that formally $O(N^3)$ scaling approaches that solve an algebraic eigenvalue problem are generally applicable to any class of system, and the computational effort associated with them has a low prefactor, i.e., they are advantageous to use for systems comprised of up to roughly a few thousands of atoms, which account for the bulk of KS-DFT applications. In contrast, the transition to lower-scaling solution methods for larger systems is not necessarily simple. Such alternatives are typically restricted to certain classes of systems or problems. The transition is, therefore, not trivial to automate, requiring specific intervention and sometimes specialist knowledge by its users. This creates hurdles both from a user point of view (complexity of choice) and from a developer point of view (replication of often complex infrastructure to implement a particular method efficiently). The KS eigenvalue problem is thus in practice a bottleneck of KS-DFT simulations on current HPC architectures and for system sizes significantly exceeding several thousands of atoms.

We here present a software infrastructure, ELSI, that simplifies the approach to overcome the Kohn–Sham eigenproblem bottleneck as much as possible for electronic structure users and developers. ELSI provides an integrated and extendable interface to multiple strategies targeting the KS eigenproblem (referred to as Kohn–Sham electronic structure solvers throughout this paper). It presently (version: 2017.05) supports three solvers: ELPA (Eigenvalue soLvers for Petaflop-Applications) [14,15], libOMM (Orbital Minimization Method) [16], and PEXSI (Pole EXpansion and Selected Inversion) [17–19]. For the future, ELSI is expressly intended to integrate further solvers such as the linear-scaling solver CheSS (CHEbyshev Sparse Solvers) [20], the iterative solver SIPs (Shift-and-Invert Parallel Spectral transformation eigensolver) [21], and others. By design, ELSI is an open infrastructure, intended to serve a community, and it can and should be flexibly adaptable to new solvers and new electronic structure codes' needs in the future. In this paper, we describe the outline and basic principles of ELSI, as well as a comparative assessment of the three solution strategies that are already supported in ELSI as of its 2017.05 release. The software presented here is a structural foundation that is already working in several electronic structure codes, and we expect it to become a focal point for new developments and solver cross-comparisons in the future.

2. Kohn–Sham density-functional theory

In KS-DFT [1], the many-electron problem for the Born–Oppenheimer electronic ground state is reduced to a system of single particle equations known as the Kohn–Sham equations

$$\hat{h}^{\mathrm{KS}}\psi_l = \epsilon_l \psi_l, \tag{1}$$

where ψ_l and ϵ_l are Kohn–Sham orbitals and their associated eigenenergies, and $\hat{h}^{\mathrm{KS}}$ denotes the Kohn–Sham Hamiltonian:

$$\hat{h}^{\mathrm{KS}} = \hat{t}_{\mathrm{s}} + \hat{v}_{\mathrm{es}} + \hat{v}_{\mathrm{xc}} + \hat{v}_{\mathrm{ext}}, \tag{2}$$

which includes the kinetic energy $\hat{t}_{\mathrm{s}}$, the average electrostatic potential of the electron density and of the nuclei $\hat{v}_{\mathrm{es}}$ (i.e. the Hartree potential), the exchange–correlation potential $\hat{v}_{\mathrm{xc}}$, and possible additional potential terms $\hat{v}_{\mathrm{ext}}$ from external electromagnetic fields.

In almost all practical approaches, N_{basis} basis functions $\phi_i(\boldsymbol{r})$ are employed to approximately expand the Kohn–Sham orbitals:

$$\psi_l(\boldsymbol{r}) = \sum_{j=1}^{N_{\mathrm{basis}}} c_{jl}\phi_j(\boldsymbol{r}). \tag{3}$$

The choice of basis set is one of the critical decisions in the design of an electronic structure code [22]. Using non-orthogonal basis functions (e.g., Gaussian functions [22–28], Slater functions [29,30], numeric atom-centered orbitals [31–36], (linearized) augmented plane waves [37–41], finite elements [42]) in Eq. (3) converts Eq. (1) to a generalized eigenvalue problem

$$\sum_j h_{ij}c_{jl} = \epsilon_l \sum_j s_{ij}c_{jl}, \tag{4}$$

where h_{ij} and s_{ij} are the elements of the Hamiltonian matrix $\boldsymbol{H}$ and the overlap matrix $\boldsymbol{S}$, which can be computed through numerical integrations:

$$\begin{aligned} h_{ij} &= \int d^3r[\phi_i^*(\boldsymbol{r})\hat{h}^{\mathrm{KS}}\phi_j(\boldsymbol{r})], \\ s_{ij} &= \int d^3r[\phi_i^*(\boldsymbol{r})\phi_j(\boldsymbol{r})]. \end{aligned} \tag{5}$$

Eq. (4) can thus be expressed in the following matrix form:

$$\boldsymbol{Hc} = \boldsymbol{\epsilon Sc}. \tag{6}$$

Here, the matrix $\boldsymbol{c}$ and diagonal matrix $\boldsymbol{\epsilon}$ contain the eigenvectors and eigenvalues, respectively, of the eigensystem of the matrices $\boldsymbol{H}$ and $\boldsymbol{S}$.

When using orthonormal basis sets (e.g., plane waves [9,43–47], multi-resolution wavelets [48–50], adaptive local basis set [51,52], grid-discretization based approaches [53,54]), the eigenproblem described in Eq. (6) reduces to a standard form where $s_{ij} = \delta_{ij}$, or even can be circumvented completely by solving the KS equations in an integral formulation [22].

The explicit solution of Eq. (4) or (6) yields the Kohn–Sham orbitals ψ_l, from which the electron density $n(\boldsymbol{r})$ can be computed following an orbital-based method that scales as $O(N^2)$:

$$n(\boldsymbol{r}) = \sum_{j=1}^{N_{\mathrm{basis}}} f_l \psi_l^*(\boldsymbol{r})\psi_l(\boldsymbol{r}), \tag{7}$$

where f_l denotes the occupation number of each orbital. In an actual computation, it is sufficient to perform the summation only for the occupied ($f_l > 0$) orbitals. The ratio of occupied orbitals to the total number of basis functions can be below 1% for plane wave basis sets, whereas with some localized basis sets, fewer basis functions are required, leading to a larger fraction of occupied states typically between 10% and 40%.

An alternative method that scales as $O(N)$ can be employed for localized basis functions:

$$n(\boldsymbol{r}) = \sum_{i,j}^{N_{\mathrm{basis}}} \phi_i^*(\boldsymbol{r})n_{ij}\phi_j(\boldsymbol{r}), \tag{8}$$

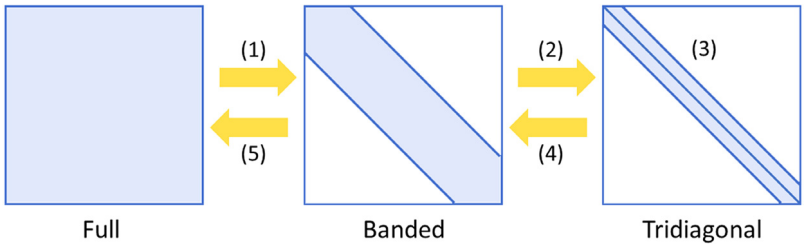

Fig. 1. Five computational steps of the ELPA eigensolver with two-stage tridiagonalization. (1) Reduction of the full matrix to a banded form. (2) Reduction of the banded matrix to a tridiagonal form. (3) Solution of the eigenvalues and eigenvectors of the tridiagonal system. (4) Back-transformation of the eigenvectors to the banded form. (5) Back-transformation of the eigenvectors to the original full form. This figure is redesigned based on Fig. 1 in Ref. [15].

with n_{ij} being the elements of the density matrix that need to be computed before the density update:

$$n_{ij} = \sum_{l=1}^{N_{\text{basis}}} f_l c_{il} c_{jl}. \tag{9}$$

Due to the dependence of $\boldsymbol{H}$ on ψ_l via the density and the potentials, Eqs. (4) and (6) are in fact non-linear eigenvalue problems, and therefore must be solved in an iterative fashion. The most commonly used method is the self-consistent field (SCF) or fixed-point iteration approach. To achieve self-consistency, the electron density needs to be updated in every iteration until converged to an acceptable level. From a viewpoint of computational complexity, almost all standard pieces of solving the Kohn–Sham equations can be formulated in a linear scaling fashion with respect to the system size. The only piece that cannot, in all cases and for all semilocal and hybrid functionals, be easily addressed in an $O(N)$ fashion is the Kohn–Sham eigenvalue problem described in Eq. (4).

3. Kohn–Sham electronic structure solvers supported by ELSI

3.1. ELPA: Eigenvalue soLvers for Petaflop-Applications

The Kohn–Sham eigenvalue problem in Eq. (4) can be explicitly solved by traditional (tri)diagonalization [55]. In ELSI, the massively parallel direct solver ELPA [14,15] facilitates the solution of symmetric or Hermitian eigenproblems on high-performance computers. It was initially designed for distributed memory architectures, then extended to exploit multi-threading parallelism, and is subject to ongoing work for GPU acceleration.

In ELPA, the generalized eigenproblem in Eq. (6) is first transformed to the standard form by Cholesky decomposition of the overlap matrix $\boldsymbol{S}$:

$$\boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}^*, \tag{10}$$

where $\boldsymbol{L}$ is a lower triangular matrix. Eq. (6) is then transformed by applying the Cholesky factor:

$$\tilde{\boldsymbol{H}}\tilde{\boldsymbol{c}} = \epsilon\tilde{\boldsymbol{c}} \tag{11}$$

with $\tilde{\boldsymbol{H}} = \boldsymbol{L}^{-1}\boldsymbol{H}(\boldsymbol{L}^*)^{-1}$ and $\tilde{\boldsymbol{c}} = \boldsymbol{L}^*\boldsymbol{c}$.

Then, the standard eigenproblem is either directly reduced to the tridiagonal form

$$\boldsymbol{T} = \boldsymbol{Q}\tilde{\boldsymbol{H}}\boldsymbol{Q}^*, \tag{12}$$

or first reduced to a banded intermediate form, then to the tridiagonal form [56]:

$$\begin{aligned} \boldsymbol{B} &= \boldsymbol{Q}_1\tilde{\boldsymbol{H}}\boldsymbol{Q}_1^*, \\ \boldsymbol{T} &= \boldsymbol{Q}_2\boldsymbol{B}\boldsymbol{Q}_2^*. \end{aligned} \tag{13}$$

In Eqs. (12) and (13), $\boldsymbol{Q}$, $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$ are transformation matrices; $\boldsymbol{T}$ is a tridiagonal matrix; $\boldsymbol{B}$ is a banded matrix.

The key steps of the two-stage tridiagonalization algorithm implemented in ELPA are reviewed in Fig. 1. Steps (1) and (2) correspond to Eq. (13), i.e. the transformations to the banded and tridiagonal forms. Step (3) corresponds to the solution of the actual eigenvalue problem by a divide-and-conquer approach [14,57], which can be restricted to compute only a fraction of the eigenvectors. Finally, the computed eigenvectors are transformed back into the representations corresponding to the banded (step (4)) and standard forms (step (5)) of the problem. Compared to the one-step tridiagonalization (Eq. (12)), the two-step algorithm introduces two additional steps (steps (1) and (5) in Fig. 1). Still, the two-step approach has been shown to enable faster computation and better parallel scalability than the one-step approach on present-day computers [15]. Specifically, the matrix–vector operations (BLAS level-2 routines) in the one-step tridiagonalization can be mostly replaced by more efficient matrix–matrix operations (BLAS level-3 routines) in the two-step version of the algorithm [58]. Since steps 2 and 4 pertain to forward and back transformations between banded and tridiagonal matrices only, the resulting transformations can be efficiently grouped to minimize computational overhead, especially for the back transformation in step (4) [14]. The computational workload associated with step (4) is further alleviated in KS-DFT calculations if only a small fraction of the eigenvectors representing the lowest eigenstates is required, and by architecture-specific linear-algebra "kernels" provided with the ELPA library [14,15].

Since ELPA employs the same 2D block-cyclic matrix distribution as does the ScaLAPACK library [59] (by way of the basic linear algebra communication subroutines (BLACS) [60]), it can easily be substituted into existing codes that already support parallel linear algebra by ScaLAPACK.

3.2. libOMM: Orbital Minimization Method

Instead of diagonalizing the $N_{\text{basis}} \times N_{\text{basis}}$ eigenproblem, the orbital minimization method (OMM) relies on efficient iterative algorithms to directly minimize an unconstrained energy functional using a set of auxiliary orbitals that are not the Kohn–Sham orbitals ϕ_i. These auxiliary orbitals are then used to obtain the density matrix of the system. Specifically, the OMM employs $N_{\text{W}} = N_{\text{electron}}/2$ non-orthogonal Wannier functions χ_k to represent the occupied subspace of a system with N_{electron} electrons:

$$\chi_k = \sum_{j=1}^{N_{\text{basis}}} W_{kj}\phi_j. \tag{14}$$

For non-spinpolarized systems, the index k runs from 1 to N_{W}. Then the matrices $\boldsymbol{H}$ and $\boldsymbol{S}$ in the occupied subspace become

$$\begin{aligned} \boldsymbol{H}_{\text{omm}} &= \boldsymbol{W}^*\boldsymbol{H}\boldsymbol{W}, \\ \boldsymbol{S}_{\text{omm}} &= \boldsymbol{W}^*\boldsymbol{S}\boldsymbol{W}, \end{aligned} \tag{15}$$

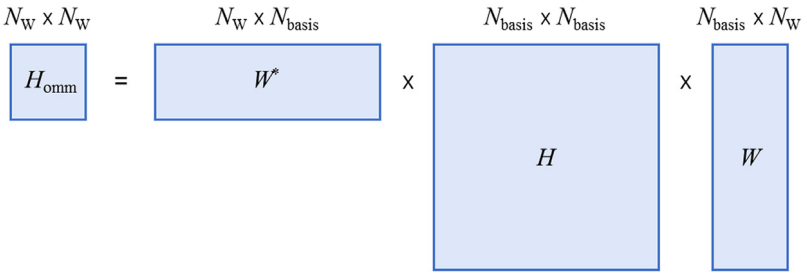

Fig. 2. Schematic representation of sizes of Hamiltonian matrix before and after applying the Wannier function transformation in the orbital minimization method. Matrix dimensions are shown above the matrices. N_W: Number of Wannier functions. N_{basis}: Number of basis functions.

where $\boldsymbol{W}$ is the coefficient matrix of the Wannier functions. The size change of the Hamiltonian matrix facilitated by Eq. (15) is illustrated in Fig. 2.

The OMM energy functional is defined as

$$E[\boldsymbol{W}] = 4Tr[\boldsymbol{H}_{\text{omm}}] - 2Tr[\boldsymbol{S}_{\text{omm}}\boldsymbol{H}_{\text{omm}}]. \tag{16}$$

This functional, when minimized with respect to the coefficients of Wannier functions $\boldsymbol{W}$, can be shown to be equal to the sum of the lowest $N_{\text{electron}}/2$ eigenvalues of the original KS eigenproblem [61–64]. Furthermore, the Wannier functions are driven towards perfect orthonormality at this minimum. The density matrix is then constructed from the $\boldsymbol{W}$ that minimizes $E[\boldsymbol{W}]$. Although this density matrix is sufficient for the electron density update following Eq. (8), compared to the density matrix in Eq. (9), it is obvious that the occupation numbers are restricted to be integers (1 for occupied; 0 for unoccupied) in this method. Without knowledge of individual eigenstates, the OMM cannot handle systems with fractional occupation numbers resulting, e.g., from a finite electronic temperature, such as is typically required for metals.

Compared to other minimization methods with the orthonormality constraint of eigenstates [47,65,66], the advantage of the OMM is that it only requires an unconstrained minimization without an explicit orthonormalization step. This makes the OMM a good candidate for linear scaling DFT; indeed, the method was originally developed in this context [61–64]. However, in order to do so, it is necessary to spatially confine the Wannier functions by imposing a certain sparsity to $\boldsymbol{W}$. This introduces a number of technical difficulties which have ultimately required the development of more involved algorithms [61,63,67]. The properties of the original OMM functional with unconstrained Wannier functions have nevertheless been found to result in an extremely efficient iterative solver with conventional cubic scaling but a smaller prefactor than diagonalization. This approach has been taken by the new implementation in libOMM [16]. It should be noted that for finite-range basis sets in which $\boldsymbol{W}$ is formally sparse, this sparsity can be taken into account to reduce the scaling of the matrix–matrix product $\boldsymbol{HW}$ from cubic to quadratic, thus effectively eliminating the most expensive matrix operation in the algorithm. The minimization of the OMM energy functional in Eq. (16) is carried out in libOMM by using the conjugate-gradient (CG) method with an efficient preconditioning using the kinetic energy matrix, as described in Ref. [16].

3.3. PEXSI: Pole EXpansion and Selected Inversion

The density matrix in Eq. (9) is associated with the Kohn–Sham orbitals and their occupation numbers f_l, which are given by the Fermi–Dirac distribution function [68]:

$$f_l = \frac{1}{1 + e^{\frac{\epsilon_l - \mu}{k_B T}}}. \tag{17}$$

Here k_B is the Boltzmann constant, T is the temperature, and μ is the chemical potential that is determined by the normalization condition

$$\sum_{l=1}^{N_{\text{basis}}} f_l = N_{\text{electron}}. \tag{18}$$

The pole expansion and selected inversion (PEXSI) method [17–19,69,70] provides an alternative way for solving the Kohn–Sham electronic structure without diagonalization. As a Fermi operator expansion (FOE) based method, PEXSI expands the density matrix in Eq. (9) using a P-term pole expansion:

$$n \approx \sum_{l=1}^{P} \text{Im}\left(\omega_l^{\rho}(\boldsymbol{H} - (z_l + \mu)\boldsymbol{S})^{-1}\right). \tag{19}$$

Here the complex shifts $\{z_l\}$ and weights $\{\omega_l^{\rho}\}$ are determined through a semi-analytic formula based on contour integration, and take only a negligible amount of time to compute. The number of terms of the pole expansion is proportional to $\log(\beta\Delta E)$, where $\beta = 1/(k_B T)$ is the inverse of the thermal energy and ΔE is the spectral radius. The logarithmic scaling makes the pole expansion a highly efficient approach to expand the Fermi operator. Typically $40 \sim 80$ poles are sufficient for the result obtained from PEXSI to be fully comparable (μeV/atom [18,19]) to that obtained from diagonalization.

At first it may seem that the entire Green's function-like object $(\boldsymbol{H} - (z_l + \mu)\boldsymbol{S})^{-1}$ needs to be computed. However, if targeting at the electron density $n(\boldsymbol{r})$, in general only the entries corresponding to the non-zero pattern of $\boldsymbol{H}$ and $\boldsymbol{S}$ are actually needed. Then a selected inversion algorithm can be used to efficiently compute these selected elements of the Green's function object, and therefore the electron density.

The computational cost of the PEXSI technique scales at most as $O(N^2)$. The actual complexity depends on the dimensionality of the system: $O(N)$ i.e. linear scaling for quasi-1D systems such as nanotubes; $O(N^{1.5})$ for quasi-2D systems such as surfaces and slabs; and $O(N^2)$ for general 3D bulk systems. This favorable scaling hinges on the sparse character of the Hamiltonian and overlap matrices, but not on any fundamental assumption about the localization properties of the single particle density matrix. This method is not only applicable to the efficient computation of the electron density, but also to other physical quantities such as the free energy, atomic forces, density of states and local density of states, all obtainable without computing any eigenvalues or eigenvectors [18]. These quantities can be given by pole expansions with the same complex shifts as those used for computing the electron density, with different weights.

PEXSI allows the usage of a hybrid scheme of density of states estimation based on Sylvester's law of inertia [71], and Newton's method to obtain the chemical potential [19], hereafter referred to as the PEXSI mu iteration. This is an efficient and relatively robust approach with respect to the initial guess of the chemical potential,

with or without the presence of gap states. A reasonable initial guess, e.g. obtained from the previous SCF step, can often converge the PEXSI mu iteration in one step.

The PEXSI method has a two-level parallelism structure and is by design highly scalable. The recently developed massively parallel PEXSI technique can make efficient use of 10,000 $\sim$ 100,000 processors on high performance machines.

4. The ELSI infrastructure

4.1. Overview of the ELSI interface

KS-DFT is implemented by a broad, diverse ecosystem of different software packages with different specialties and different numerical discretization strategies (see, e.g., Ref. [4] for a listing of 46 packages). The Kohn–Sham eigenvalue problem is unavoidable in all these packages. Since the most efficient way to solve the problem may depend on factors such as system size and character (insulating or metallic), sparsity of matrices involved, density-functional employed, etc., from a user's perspective, a library that can dynamically switch between different methods according to the features of the problem is preferred. As a first step to achieve this goal (the objective of this paper), a flexible interface to different methods should enable user codes to actively select the most effective method while imposing only a minimum of format conversions, parameter tweaking, etc. on the user code.

Although each solver library supported in ELSI maintains a limited number of well-explained Application Programming Interfaces (APIs), integrating all of them into a KS-DFT code is still a complicated, time-consuming, and error-prone task. ELSI ships a small set of APIs that are designed for rapid integration of a variety of KS electronic structure solvers into KS-DFT codes, and at the same time provides the user with hierarchical control over the interface and the solvers. There are three key steps to use ELSI, denoted by the red boxes (a), (b) and (c) in Fig. 3: (a) The ELSI interface needs to be initialized at the beginning of an SCF calculation, and potentially re-initialized if performing successive SCF cycles, e.g. for different system geometries during a molecular dynamics simulation or during a geometry optimization calculation. (b) Within the SCF cycle, ELSI serves as a bridge between the KS-DFT codes and the KS solver libraries, by taking the Hamiltonian matrix (and the overlap matrix if it exists) as input, translating the eigenproblem into a solver-specific format, invoking the solver to compute the eigenvalues and eigenvectors, or the density matrix, and finally translating the results back to the native format of the KS-DFT codes. (c) When ELSI is no longer needed, it should be finalized to deallocate any arrays internally allocated by ELSI.

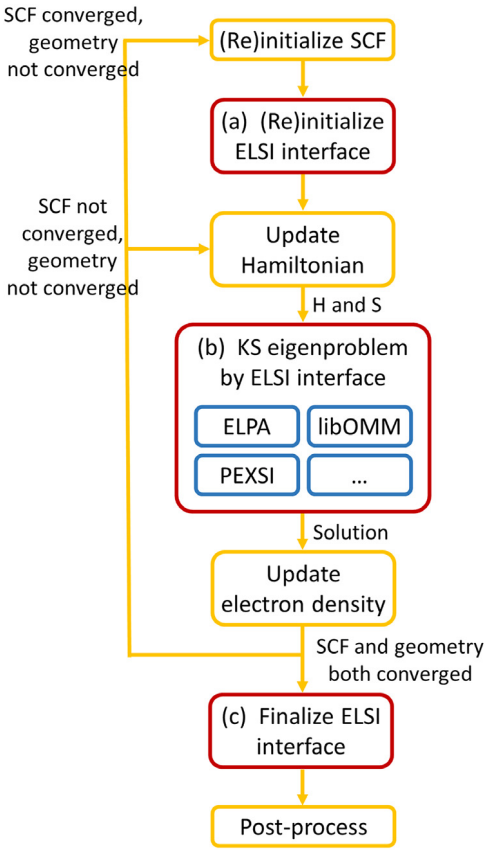

Fig. 3. Flow chart describing the key steps in a self-consistent field calculation based on Kohn–Sham Density-Functional Theory. Yellow boxes: Key steps commonly implemented in KS-DFT codes to perform a single SCF cycle or multiple successive SCF cycles with different atomic structures, e.g. for molecular dynamics or for geometry optimizations. Red boxes: Required additions to use the ELSI interface, including (a) initialization of the ELSI interface, (b) computing the eigensolution or the density matrix using the ELSI solvers, and (c) finalization of the ELSI interface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4.2. Matrix storage and distribution in ELSI

The first emerging practical consideration when developing a unified software interface is the choice of matrix storage and distribution strategy. The sparsity of matrices in KS-DFT varies dramatically from small to large systems, and from 1D to 3D systems. In general, when using localized basis functions, the sparsity of matrices increases as the simulated system becomes larger. Lower dimensional systems often generate more sparse matrices. Since the effective information is only represented by the non-zero matrix elements, storing and operating on all the matrix elements lead to unnecessary memory consumption and computational complexity for very sparse matrices.

Implementing dense linear algebra operations, ELPA and libOMM handle matrices stored densely and distributed in a 2D block-cyclic distribution, whereas PEXSI performs sparse linear algebra with matrices stored in compressed sparse column (CSC, also known as compressed column storage, CCS) format in a 1D block distribution. These two combinations, hereafter referred to as BLACS_DENSE and PEXSI_CSC formats, respectively, are chosen as the input/output matrix format of the ELSI interface to bridge the needs of the solvers and of different KS-DFT codes. The comparison between dense matrix storage and CSC sparse matrix storage is illustrated in Fig. 4, using an 8 $\times$ 8 matrix as an example. The dense storage keeps all the matrix elements including zeros and non-zeros. The CSC format, in contrast, drops the zeros and packs the remaining non-zeros into a 1D array, together with the row indices of the non-zero values and the starting points of the matrix columns. For a larger matrix with a higher sparsity, the CSC format will eventually consume less memory compared to the dense format.

To compare the two supported distributions of matrices across multiple processors in parallel computations, Fig. 5 shows how the 2D block-cyclic and the 1D block distributions are applied to the same 8 $\times$ 8 matrix. We note that shown in Fig. 5 are two mathematical matrices, the shapes of which do not represent the actual arrays in the computer. The 2D block-cyclic distribution in Fig. 5(a) divides the global matrix into several blocks, then maps the blocks to the processors in a round-robin fashion in both the row and the column directions. The 1D block distribution in Fig. 5(b) groups continuous matrix columns together, then linearly maps the groups of columns to the processors. In ELSI, when the input matrices are in a different distribution from the internally

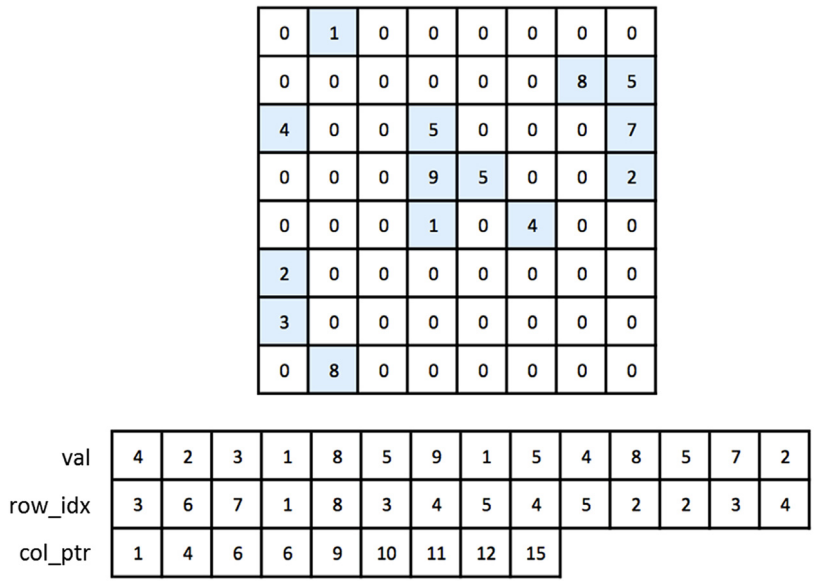

Fig. 4. An 8 × 8 matrix stored in (a) dense storage format versus in (b) compressed sparse column (CSC) storage format. In the CSC format, only the values of the non-zero elements, indicated in blue in (a), are stored in the "val" array. The row indexes of the non-zero elements are stored in the "row_idx" array. The "col_ptr" array stores the starting points of the matrix columns.

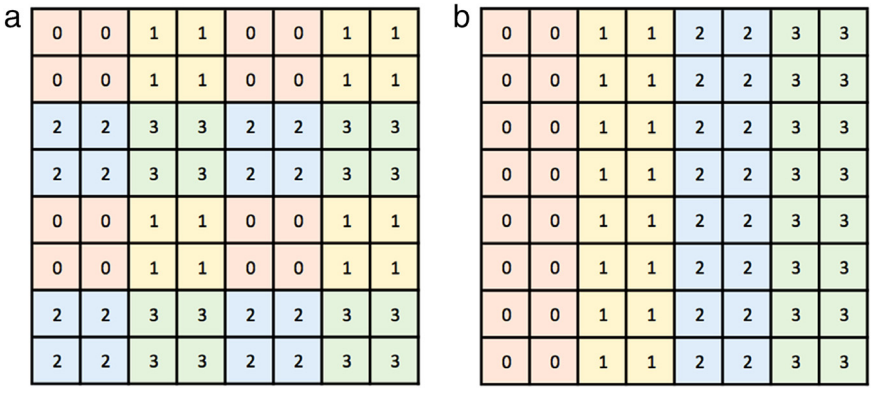

Fig. 5. Schematic visualizations of (a) two-dimensional block-cyclic distribution used in the BLACS_DENSE format, and (b) one-dimensional block distribution used in the PEXSI_CSC format, of an 8 × 8 matrix on 4 processors. Each unit square represents one matrix element. The integer inside each unit square denotes the index of the processor where the element is stored and handled. Different processors are indicated by colors. Shown in the figure are mathematical matrices, not arrays in computers. The actual matrix storage on each processor is arbitrary, e.g. dense storage used by the BLACS_DENSE format, CSC sparse storage used by the PEXSI_CSC format. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

used one, a redistribution of the non-zero matrix elements is performed internally, i.e. no unnecessary communication of the zero elements. This redistribution is managed by the all-to-all communication implemented in the Message Passing Interface (MPI) library. Once the matrix is correctly distributed, conversion to various formats is then handled concurrently on all the MPI tasks, with each task converting a local matrix of the size at most $N_{\text{basis}}^2/N_{\text{MPI}}$, where N_{MPI} is the number of MPI tasks involved.

4.3. Parallelization strategy and interaction of ELSI with an existing KS-DFT code

An important distinction in KS-DFT calculations is whether the system considered is isolated or is periodically repeated in space. In periodic systems, the full problem can be separated into subproblems defined at selected k-points in the Brillouin zone, or in a convenient unit cell in reciprocal space. The Hamilton and overlap matrices for multiple k-points are block-diagonal, such that each block on the diagonal corresponds to an eigenproblem of one k-point. These eigenproblems can therefore be solved in a embarrassingly parallel fashion side by side. For periodic systems with a small unit cell, thousands of k-points or even more can be necessary for an accurate description of the electronic structure. For a large system, in contrast, the Brillouin zone may already be well-represented by the origin of the reciprocal space known as the Γ point.

Depending on the number of k-points N_{kpt} (here defined to be 1 also for isolated, non-periodic cases) and the number of MPI tasks N_{MPI}, two different categories of possible KS-DFT calculations arise, as explained in Fig. 6. Correspondingly, ELSI supports two parallelization strategies that can be specified by a parallel_mode parameter (see also elsi_init subroutine in Section 4.4):

- MULTI_PROC mode, to be used if $N_{\text{MPI}} \geq N_{\text{kpt}}$. For instance, there are 4 k-points in example (a) in Fig. 6, handled by 16 MPI tasks. The MULTI_PROC parallelization divides the 16 MPI tasks into 4 groups. Each k-point is handled by 4 MPI tasks in the same group, and the eigenproblems of the 4 k-points are solved simultaneously by the 16 MPI tasks. Since each k-point is solved by multiple MPI tasks (processes), this parallelization mode is called MULTI_PROC. This mode

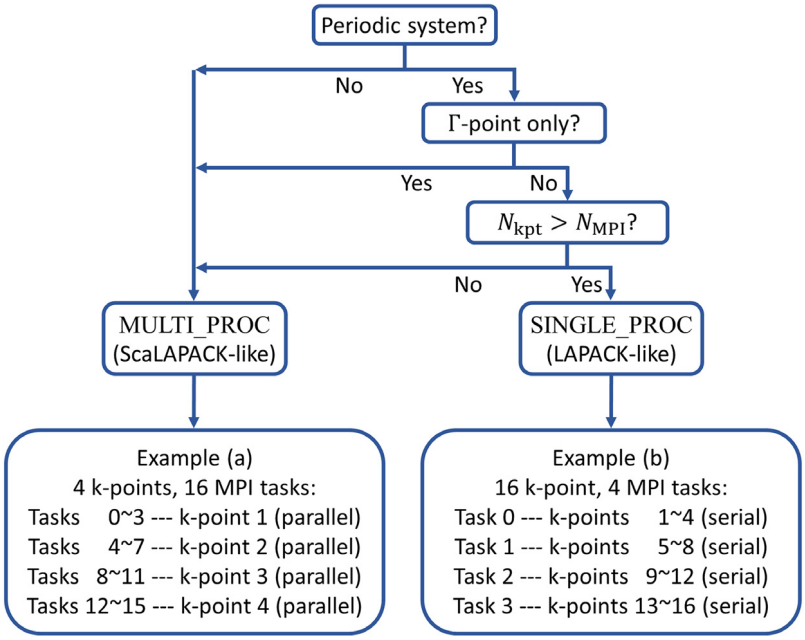

Fig. 6. Diagrammatic explanations of the two parallelization strategies supported by ELSI. N_{kpt} is the number of k-points. N_{MPI} is the number of MPI tasks.

should be chosen for isolated systems, periodic systems with only one k-point, e.g. the Γ point, and periodic systems with N_{kpt} k-points treated by $N_{MPI} \geq N_{kpt}$ MPI tasks.

- SINGLE_PROC mode, to be used if $N_{MPI} < N_{kpt}$. For instance, there are 16 k-points in example (b) in Fig. 6, handled by 4 MPI tasks. The SINGLE_PROC parallelization divides the 16 k-points into 4 groups. Each MPI task handles 4 k-points in the same group, one after another. Since each k-point is solved by a single MPI task (process), this parallelization mode is called SINGLE_PROC. This mode should be chosen for periodic systems with N_{kpt} k-points treated by $N_{MPI} < N_{kpt}$ MPI tasks.

The ELPA eigensolver supported in ELSI is available for both parallel modes, returning eigensolutions for each k-point. In this case, the KS-DFT codes can then assemble the pieces of the solutions (eigenvalues and eigenvectors) returned by the solver and construct the electron density. The density matrix solvers in the 2017.05 release of ELSI do not yet support periodic calculations with more than one k-point. The ability to return density matrices directly for problems with multiple k-points is, however, included in the ongoing ELSI development (post 2017.05) within the MULTI_PROC parallel mode. In this development version, k-point (and spin-channel) dependent density matrices can be directly returned by each of the three solvers ELPA, PEXSI and libOMM as follows: If the ELPA solver is used, the eigenproblem at each k-point can be solved separately and simultaneously. Then the eigenvalues need to be collected among k-points for the determination of the occupation numbers following Eq. (18) (the summation becomes $\sum_{i=1}^{N_{kpt}} \sum_{k=1}^{N_{basis}}$). With the correct occupation numbers, density matrices of the k-points can be independently computed by Eq. (9). If PEXSI is used, the Fermi operator expansion in Eq. (19) at each k-point will be performed in parallel with the same input chemical potential μ. The resulting number of electrons needs to be determined across all k-points in order to evaluate the quality of the input chemical potential and to find the correct chemical potential iteratively. If libOMM is used, OMM minimization for each k-point will be performed in parallel, analogously to the parallel handling of eigenproblems with ELPA and Fermi operator expansions with PEXSI. As explained in Section 3.2, libOMM in its current form is restricted to non-metallic systems, that is, systems with a gap.

Once the matrix storage format and the parallel mode are decided, the usage of ELSI in KS-DFT codes becomes straightforward. Algorithm 1 summarizes in pseudo-code all the possible use cases of the ELSI interface as of the 2017.05 release. In Algorithm 1, the main steps are denoted by subroutine names that will be systematically introduced in the following subsections. Furthermore, the initialization of the SCF calculation, updating the Hamiltonian and the electron density, checking the SCF convergence, postprocessing, and potentially further steps are all tasks that are not handled by ELSI but that are instead expected to be executed by the specific KS-DFT code calling ELSI.

Before showing detailed descriptions of the ELSI API in the next subsections, we here first introduce the concept of elsi_handle, a Fortran derived data type containing all runtime parameters, e.g. the choices of solver, matrix_storage_format, and parallel_mode (see elsi_init subroutine in Section 4.4). It is intended to avoid global variables in ELSI and to allow concurrent instances of ELSI by passing around the handle as arguments. A handle can be initialized with the elsi_init subroutine, then should be passed to all other ELSI subroutines. The ELSI interface, including the elsi_handle, is fully interoperable with C and C++ programming languages. The elsi_handle is defined in C/C++ as an "opaque" pointer, which can be seamlessly connected to a derived data type in Fortran by the iso_c_binding feature in Fortran compilers.

4.4. ELSI initialization

In this and the following subsections (Sections 4.5–4.7), we provide details of the capabilities of the ELSI interface as current in the 2017.05 release. Since these capabilities are intimately tied to the actual implementation, we here explain them grouped by individual subroutines as also shown in Algorithm 1. In all instances, elsi_h denotes the ELSI handle.

In the initialization phase, ELSI can be set up to reflect the physical quantities that usually do not change within an SCF calculation (i.e. fixed atomic structure), such as the number of basis functions and the number of electrons in the system. Implementations of SCF typically initialize these quantities before the SCF cycle begins,

Algorithm 1 Usage of ELSI interface in KS-DFT codes. Pseudo-code in line 3–11, line 14–27, and line 31 corresponds to Fig. 3(a), (b), and (c), respectively.

```
1:  procedure ELSI
2:      initialize SCF calculation
3:      call elsi_init
4:      if (parallem_mode = MULTI_PROC) then
5:          call elsi_set_mpi
6:          if (matrix_storage_format = BLACS_DENSE) then
7:              call elsi_set_blacs
8:          else if (matrix_storage_format = PEXSI_CSC) then
9:              call elsi_set_csc
10:         end if
11:     end if
12:     while (SCF not converged) do
13:         update Hamiltonian
14:         call elsi_customize
15:         if (desired output: eigensolution) then
16:             if (matrix_storage_format = BLACS_DENSE) then
17:                 call elsi_ev_{real|complex}
18:             else if (matrix_storage_format = PEXSI_CSC) then
19:                 call elsi_ev_real_sparse
20:             end if
21:         else if (desired output: density matrix) then
22:             if (matrix_storage_format = BLACS_DENSE) then
23:                 call elsi_dm_real
24:             else if (matrix_storage_format = PEXSI_CSC) then
25:                 call elsi_dm_real_sparse
26:             end if
27:         end if
28:         update electron density
29:         check SCF convergence
30:     end while
31:     call elsi_finalize
32:     post-process
33: end procedure
```

then keep reusing them within the cycle to repeatedly solve KS problems with an updated Hamiltonian matrix and a fixed overlap matrix. Similarly, the ELSI interface only needs to be (re-)initialized whenever the SCF cycle is itself (re-)initialized. The subroutines that are used to initialize ELSI include:

- **elsi_init** (elsi_h, solver, matrix_storage_format, parallel_mode, n_basis, n_state, n_electron)
  (line 3 in Algorithm 1) – Initializes an ELSI handle with user's choices of the solver, the matrix format and distribution, the parallelization strategy, and system information including the number of basis functions, the number of eigenstates to compute, and the number of electrons.
  - **elsi_h** (type(elsi_handle), output): An ELSI handle (see Section 4.3) returned by elsi_init subroutine. The same handle must be passed to other ELSI subroutines and be finalized when no longer needed. Multiple handles can be initialized if needed.
  - **solver** (integer, input): The choice of solver. Accepted options are 1 (ELPA), 2 (libOMM), and 3 (PEXSI).
  - **matrix_storage_format** (integer, input): Matrix storage and distribution of the Hamiltonian matrix, the overlap matrix, and the density matrix or the eigenvectors. Accepted options are 1 (BLACS_DENSE) and 2 (PEXSI_CSC) (see Section 4.2). The BLACS_DENSE format is compatible with ELPA, libOMM, and PEXSI. If the chosen solver is PEXSI, the input matrices in the BLACS_DENSE format are converted to PEXSI_CSC internally, and the results in the PEXSI_CSC format are back-converted to BLACS_DENSE. The PEXSI_CSC format is compatible with ELPA and PEXSI in the current release. Supporting the PEXSI_CSC format with libOMM is on the list of features to be added in the near future.
  - **parallel_mode** (integer, input): The choice of parallelization strategy. Accepted options are 1 (MULTI_PROC) and 2 (SINGLE_PROC) (see Section 4.1). In the current release of ELSI, the SINGLE_PROC mode is only compatible with ELPA, while the MULTI_PROC mode supports all three solvers.
  - **n_basis** (integer, input): Number of basis functions. This is equal to the global size of the Hamiltonian matrix, the overlap matrix, the density matrix, etc.
  - **n_state** (integer, input): Number of states. For ELPA this is the number of eigenstates to be solved. For libOMM this must be the number of occupied states, without any fractional occupation numbers. PEXSI does not use this information.
  - **n_electron** (integer, input): Number of electrons.
- **elsi_set_mpi** (elsi_h, mpi_comm)
  (line 5 in Algorithm 1) – Sets the MPI communicator to be used in the ELSI instance indicated by the handle.
  - **mpi_comm** (integer, input): An MPI communicator, containing an ordered group of MPI tasks, is required to use the functionalities implemented in the MPI library. The communicator assigned to an ELSI calculation can be the default global communicator of MPI, or a communicator created by the user (e.g. by calling the MPI subroutine MPI_Comm_Split), as long as it is compatible with the distribution of matrices.
- **elsi_set_blacs** (elsi_h, blacs_ctxt, block_size)
  (line 7 in Algorithm 1) – Sets the BLACS context and the block size of the 2D block-cyclic distribution to be used in the ELSI instance indicated by the handle. Required before calling elsi_ev_real, elsi_ev_complex, and elsi_dm_real (see Section 4.5).
  - **blacs_ctxt** (integer, input): A BLACS context encloses a group of processes and arranges them in a particular grid. Processes in the same context can safely communicate with each other, without worrying if the operations in one context interfere with operations in another context [60]. The ELSI interface requires the KS-DFT code to set up BLACS context(s), by calling BLACS subroutine BLACS_Gridinit or BLACS_Gridmap.
  - **block_size** (integer, input): The block size parameter of the 2D block-cyclic distribution. The matrix operations inside ELSI interface, ELPA, and libOMM restrict the block sizes in the row and column directions to be the same.
- **elsi_set_csc** (elsi_h, nnz_g, nnz_l, n_l_cols, row_idx, col_ptr)
  (line 9 in Algorithm 1) – Set the parameters of 1D block distributed CSC matrix storage (PEXSI_CSC) to be used in the ELSI instance indicated by the handle. Required before calling elsi_ev_real_sparse and elsi_dm_real_sparse (see Section 4.5).
  - **nnz_g** (integer, input): The global number of non-zero elements in the sparsity pattern.
  - **nnz_l** (integer, input): The local number of non-zero elements in the sparsity pattern held by an MPI task.

- **n_l_cols** (integer, input): The local number of matrix columns held by an MPI task.
- **row_idx** (integer, 1D array, input): The row index array of the CSC matrix storage format, containing the row index of each non-zero matrix element. An example is given in Fig. 4.
- **col_ptr** (integer, 1D array, input): The column pointer array of the CSC matrix storage format, containing the starting point of each matrix column. An example is given in Fig. 4.

The matrices that arise in KS-DFT can be either real or complex-valued. ELSI must account for these two possibilities as well. Since the real and complex arithmetic cases only differ in the data type of input/output matrices, they are not distinguishable at the initialization stage.

4.5. Tasks during SCF

During the SCF cycle, the following tasks may be executed by ELSI solver subroutines to compute either the eigensolutions or the density matrix from the input Hamiltonian matrix (and overlap matrix, if it is not unity).

- **elsi_ev_real** (elsi_h, ham, ovlp, eval, evec) (line 17 in Algorithm 1) – Computes the eigenvalues and n_state eigenvectors. Compatible solver: ELPA.
  - **ham** (double precision real, 2D array, input & output): The real-valued Hamiltonian matrix in the BLACS_DENSE format set by subroutine elsi_set_blacs. This array is used for internal storage when solving the eigenproblem, and thus is destroyed on exit.
  - **ovlp** (double precision real, 2D array, input & output): The real-valued overlap matrix in the BLACS_DENSE format set by subroutine elsi_set_blacs. A singularity check of the overlap matrix $\boldsymbol{S}$ is performed the first time elsi_ev_real is called. This is because the Cholesky factorization in Eq. (10) requires $\boldsymbol{S}$ to be Hermitian positive-definite. While $\boldsymbol{S}$ in KS-DFT is guaranteed to be Hermitian by Eq. (5), the positive-definite condition can be numerically violated if the chosen basis set is large and (near-)singular, i.e. the lowest eigenvalues of $\boldsymbol{S}$ are too close to 0 (although still greater than 0). Using a near-singular basis set can lead to completely wrong and unpredictable numerical results, and thus should be avoided in general. In ELSI, this is done by computing all the eigenvalues of $\boldsymbol{S}$ and comparing them with a user-defined singularity tolerance τ. The matrix is considered to be singular if it has one or more eigenvalues smaller than τ. For a singular overlap matrix, the Cholesky decomposition is replaced by an eigendecomposition:

    $$\boldsymbol{S} = (\sqrt{\boldsymbol{\lambda}}\boldsymbol{x})(\sqrt{\boldsymbol{\lambda}}\boldsymbol{x})^* = \boldsymbol{X}\boldsymbol{X}^*, \tag{20}$$

    where the matrix $\boldsymbol{x}$ and the diagonal matrix $\boldsymbol{\lambda}$ contain the eigenvectors and eigenvalues of $\boldsymbol{S}$, and the matrix $\boldsymbol{X}$ is simply $\sqrt{\boldsymbol{\lambda}}\boldsymbol{x}$. By using eigendecomposition, the generalized eigenproblem is again transformed to the standard form in Eq. (11), with $\tilde{\boldsymbol{H}} = \boldsymbol{X}^{-1}\boldsymbol{H}(\boldsymbol{X}^*)^{-1}$ and $\tilde{\boldsymbol{c}} = \boldsymbol{X}^*\boldsymbol{c}$. In case that only the first N_{nonsing} eigenvalues of $\boldsymbol{S}$ are greater than the threshold τ, $\boldsymbol{X}$ correspondingly contains only the first N_{nonsing} eigenvectors by dropping the $N_{\text{basis}} - N_{\text{nonsing}}$ eigenvectors associated with small eigenvalues. The eigenproblem transformation is still valid. However, it yields a smaller transformed $\tilde{\boldsymbol{H}}$ ($N_{\text{nonsing}} \times N_{\text{nonsing}}$). The solution of the transformed standard eigenproblem must be back-transformed accordingly.
    On exit, ovlp is overwritten by either $\boldsymbol{L}$ in Eq. (10) or $\boldsymbol{X}$ in Eq. (20), depending on which transformation is used. If in the MULTI_PROC mode, i.e. no MPI task handles more than one k-point, $\boldsymbol{L}$ or $\boldsymbol{X}$ can be stored in ovlp and efficiently reused throughout the SCF cycle. The Cholesky factorization or the eigendecomposition then only needs to be performed once. However, in the SINGLE_PROC mode, since each MPI task handles a group of k-points in serial, memory constraints make it more difficult to reuse the matrices $\boldsymbol{L}$ or $\boldsymbol{X}$. In this case, the decision to either store $\boldsymbol{L}$ or $\boldsymbol{X}$, or to redo the decomposition in every SCF iteration, is up to the KS-DFT code that calls ELSI.
  - **eval** (double precision real, 1D array, output): The eigenvalues in ascending order.
  - **evec** (double precision real, 2D array, output): The real-valued eigenvectors in a matrix form in the BLACS_DENSE format set by subroutine elsi_set_blacs.
- **elsi_ev_complex** (elsi_h, ham, ovlp, eval, evec) (line 17 in Algorithm 1) – Same as elsi_ev_real, except that the Hamiltonian matrix, overlap matrix and eigenvectors are complex-valued.
- **elsi_ev_real_sparse** (elsi_h, ham, ovlp, eval, evec) (line 19 in Algorithm 1) – Computes the eigenvalues and n_state eigenvectors. Compatible solver: ELPA.
  - **ham** (double precision real, 1D array, input): The non-zero elements of the real-valued Hamiltonian matrix in the PEXSI_CSC format set by subroutine elsi_set_csc. Inside ELSI, the input Hamiltonian matrix is converted to the BLACS_DENSE format in every SCF iteration.
  - **ovlp** (double precision real, 1D array, input): The non-zero elements of the real-valued overlap matrix in the PEXSI_CSC format set by subroutine elsi_set_csc. Inside ELSI, the input overlap matrix is converted to the BLACS_DENSE format in the first SCF iteration. The singularity check of the overlap matrix is performed as in the elsi_ev_real case. Since the sparsity of the eigenproblem transformation matrix $\boldsymbol{L}$ or $\boldsymbol{X}$ is not guaranteed, the matrix $\boldsymbol{L}$ or $\boldsymbol{X}$ is stored internally in the BLACS_DENSE format for further reuse throughout the SCF cycle.
  - **eval** (double precision real, 1D array, output): The eigenvalues in ascending order.
  - **evec** (double precision real, 2D array, output): The real-valued eigenvectors in a matrix form in the BLACS_DENSE format. Note that the computed eigenvectors are returned in a dense format, for the reason that they are not in the same sparsity pattern of $\boldsymbol{H}$ and $\boldsymbol{S}$, or even not sparse at all.
- **elsi_dm_real** (elsi_h, ham, ovlp, den_mat, energy) (line 23 in Algorithm 1) – Computes the density matrix. Compatible solvers: ELPA, libOMM, PEXSI.
  - **ham** (double precision real, 2D array, input & output): The real-valued Hamiltonian matrix in the BLACS_DENSE format set by subroutine elsi_set_blacs. This array is used for internal storage when computing the density matrix, and thus is destroyed on exit. If the chosen solver is PEXSI, the input Hamiltonian matrix is converted to the PEXSI_CSC format in every SCF iteration.

- **ovlp** (double precision real, 2D array, input & output): The real-valued overlap matrix in the BLACS_DENSE format set by subroutine elsi_set_blacs. If the chosen solver is PEXSI, the input overlap matrix is converted to the PEXSI_CSC format in the first SCF iteration and reused throughout the SCF cycle. If the chosen solver is ELPA or libOMM, the singularity check of the overlap matrix is performed as in the elsi_ev_real case. The singularity check is not yet implemented for PEXSI.
- **den_mat** (double precision real, 2D array, output): The density matrix in the BLACS_DENSE format set by subroutine elsi_set_blacs. The chemical potential and occupation numbers must be known when ELPA is chosen to compute the density matrix following Eq. (9). In ELSI, the chemical potential is found using a bisection algorithm that starts from an energy interval that includes the actual solution of the chemical potential. This is often guaranteed by using the lowest and highest eigenvalues of the system as the lower and upper bounds of the interval, and expanding the interval towards both ends if necessary. In each bisection step the number of electrons on both bounds and at the middle point of the interval is computed by Eq. (18) (the summation becomes $\sum_{i=1}^{N_{\text{kpt}}}\sum_{j=1}^{N_{\text{spin}}}\sum_{k=1}^{N_{\text{basis}}}$ if including k-points and spin channels), to determine which subinterval the solution lies in. Then the interval can be repeatedly bisected until the computed number of electrons on either bound or at the middle point is sufficiently close to the actual number. During this process, the computation of occupation numbers f_l requires a specific broadening scheme, which can be the Fermi broadening in Eq. (17), or the Gaussian broadening [72]

$$f_l = 0.5 \cdot [1 - \text{erf}\left(\frac{\epsilon_l - \mu}{k_B T}\right)], \tag{21}$$

where erf is the Gauss error function:

$$\text{erf} = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \tag{22}$$

Although the error function is implemented as an intrinsic function in most programming languages, the error of each single evaluation can accumulate as a consequence of the summation in Eq. (18). This accumulation leads to an error on the order of 10^{-10} in terms of the number of electrons, which is small but not negligible if the desired accuracy is on the same order. During the convergence of the SCF cycle, this small error can become more noticeable, since fluctuations of the norm of the density matrix (i.e. the system charge) will have a relatively large electrostatic effect, and can thus disturb the solution of the nonlinear fixed-point iteration scheme (e.g. Pulay mixing [73]) that is used to converge an SCF cycle. Therefore, it is useful and sometimes necessary to avoid charge fluctuations whenever possible, by ensuring an exact charge norm after the fact. In ELSI, when the accuracy of electron count can no longer be improved by bisection, then the remaining discrepancy (surplus of electrons in case of the upper bisection bound) is successively removed starting from the highest occupied KS states and proceeding to lower-lying states until the norm in Eq. (18) is numerically exactly fulfilled.
- **energy** (double precision real, output): The energy corresponding to the occupied eigenstates.

- **elsi_dm_real_sparse** (elsi_h, ham, ovlp, den_mat, energy) (line 25 in Algorithm 1) – Computes the density matrix. Compatible solver: PEXSI.
  - **ham** (double precision real, 1D array, input & output): The non-zero elements of the real-valued Hamiltonian matrix in the PEXSI_CSC format set by subroutine elsi_set_csc. This array is used for internal storage when computing the density matrix, and thus is destroyed on exit.
  - **ovlp** (double precision real, 1D array, input & output): The non-zero elements of the real-valued overlap matrix in the PEXSI_CSC format set by subroutine elsi_set_csc.
  - **den_mat** (double precision real, 1D array, output): The non-zero elements of the density matrix in the PEXSI_CSC format set by subroutine elsi_set_csc.
  - **energy** (double precision real, output): The energy corresponding to the occupied eigenstates.
- **elsi_collect_pexsi** (elsi_h, mu, e_den_mat, f_den_mat) – Collects additional results computed by PEXSI. Compatible solver: PEXSI.
  - **mu** (double precision real, output): The chemical potential computed by PEXSI.
  - **e_den_mat** (double precision real, 1D array, output): The non-zero elements of the energy density matrix in the PEXSI_CSC format set by subroutine elsi_set_csc.
  - **f_den_mat** (double precision real, 1D array, output): The non-zero elements of the free energy density matrix in the PEXSI_CSC format set by subroutine elsi_set_csc.

4.6. ELSI customization options

Although ELSI sets reasonable default runtime parameters for each solver whenever possible, no set of parameters can adequately cover all use cases. The elsi_customize subroutines allow a user to determine runtime parameters explicitly, thus providing maximum flexibility to control the particulars of ELSI. Designed with the feature of optional arguments in Fortran, the elsi_customize subroutines have a general calling syntax:

call elsi_customize(elsi_h, keyword=choice),

where elsi_h is the ELSI handle to be customized, "keyword" is the parameter to be customized, and "choice" is the value to overwrite the default value of "keyword". Calling elsi_customize (line 14 in Algorithm 1) only modifies the parameter associated with elsi_h, instead of changing the behavior of all handles.

- **elsi_customize** (elsi_h, keyword=choice) The following customizable keywords are particularly important:
  - **overlap_is_unit** (logical, input): ELSI by default assumes that the KS eigenproblem is a generalized problem (Eq. (6)). Setting the keyword overlap_is_unit to true allows the usage of ELSI for a standard eigenproblem, e.g. when using orthonormal basis sets, or the generalized eigenproblem has been transformed to the standard form by the calling code itself. If overlap_is_unit is true, the singularity check for the overlap matrix described in Section 4.5 will be completely ignored.
  - **zero_threshold** (double precision real, input): Threshold to define "zero" in ELSI matrix format conversions. When converting a dense matrix into a sparse format, any double precision number smaller than this threshold is overwritten by 0.

  - **no_singularity_check** (logical, input): The singularity check of the overlap matrix can be skipped here.
  - **singularity_tolerance** (double precision real, input): The tolerance of basis singularity τ in the singularity check.

- **elsi_customize_mu** (elsi_h, keyword=choice)
Customizes the chemical potential and occupation number computation in ELSI. Customizable keywords include:

  - **broadening_scheme** (integer, input): The broadening scheme to be used in the determination of occupation numbers and chemical potential. Accepted options are 1 (Gaussian broadening), 2 (Fermi broadening), 3 (0th order Methfessel–Paxton broadening), and 4 (1st order Methfessel–Paxton broadening).
  - **broadening_width** (double precision real, input): The broadening width parameter (k_BT in Eqs. (17) and (21)).
  - **occ_accuracy** (double precision real, input): Desired accuracy in terms of the sum of occupation numbers, i.e. the number of electrons, in the determination of occupation numbers and chemical potential.
  - **mu_max_steps** (integer, input): Maximum steps of the bisection algorithm (described as a part of subroutine elsi_dm_real) to compute the occupation numbers and chemical potential.

- **elsi_customize_elpa** (elsi_h, keyword=choice)
Customizes the ELPA solver. Customizable keywords include:

  - **elpa_solver** (integer, input). The choice of ELPA solvers. Accepted options are 1 (ELPA 1-stage solver) and 2 (ELPA 2-stage solver).

- **elsi_customize_omm** (elsi_h, keyword=choice)
Customizes the libOMM solver. Customizable keywords include:

  - **omm_flavor** (integer, input): The choice of method to perform OMM minimization. Accepted options are 0 (the basic flavor that follows Eq. (16) exactly) and 2 (the Cholesky flavor that transforms the generalized eigenproblem to the standard form using Cholesky factorization before minimization).
  - **n_elpa_steps** (integer, input): ELPA can be employed in the first n_elpa_steps SCF iterations, as these take the longest time to converge with iterative methods. Starting from the (n_elpa_steps + 1)th SCF step, the libOMM solver will be used with the eigenvectors computed by ELPA in the (n_elpa_steps)th SCF step as the initial guess for the coefficients of Wannier functions.
  - **omm_tolerance** (double precision real, input): The stop criterion of the OMM energy functional minimization in Eq. (16). This minimization is considered to be converged when the relative energy difference between subsequent line searches given by $2(E[\boldsymbol{W}_1] - E[\boldsymbol{W}_0])/(E[\boldsymbol{W}_1] + E[\boldsymbol{W}_0])$ is smaller than or equal to this dimensionless value.

  The convergence rate of the OMM energy functional minimization depends heavily on the minimization method and the initial guess of the coefficients of the Wannier functions. The effects of omm_flavor and n_elpa_steps on the performance of OMM are investigated and reported in Section 5.5.
- **elsi_customize_pexsi** (elsi_h, keyword=choice) Customizes the PEXSI solver.
Customizable keywords include:

  - **n_poles** (integer, input): The number of poles in the Fermi operator expansion, i.e. P in Eq. (19). The pole expansion is an exact algorithm if the number of poles is infinitely large. In practice, 40 $\sim$ 80 poles are usually sufficient for the result obtained from PEXSI to be fully comparable to that obtained from diagonalization. Performing a convergence test with increasing number of poles is a practical approach to estimate the optimal number of poles for a KS-DFT code.
  - **n_electron_accuracy** (double precision real, input): The desired accuracy in terms of the number of electrons out of the density matrix approximated by Eq. (19).
  - **temperature** (double precision real, input): The physical meaning of the temperature here is the energy $\beta = K_BT$ in Eq. (17), i.e. the broadening width.
  - **delta_e** (double precision real, input): The upper bound for the spectral radius ΔE of $S^{-1}H$. This parameter and the β parameter affect the number of terms of the pole expansion.
  - **max_iteration** (integer, input): The maximum number of PEXSI mu iterations to determine the chemical potential.
  - **mu_0, mu_min, mu_max** (double precision real, input): The initial guess, lower bound, and upper bound for the chemical potential. A good initial guess significantly accelerates the convergence of the PEXSI mu iteration. An estimate of the chemical potential is available in PEXSI via the inertia counting procedure based on Sylvester's law of inertia. Starting from the second SCF iteration, if the change in chemical potential from the previous SCF step to the current step is small, ELSI will automatically skip the inertia counting and use the chemical potential from the previous step as the initial guess for the current step.
  - **mu_safeguard** (double precision real, input): A failsafe approach designed for the PEXSI mu iteration. If the error in the chemical potential computed by PEXSI is larger than this safeguard, the code will exit the mu iteration and re-invoke the inertia counting to estimate the chemical potential.

4.7. ELSI finalization

- **elsi_finalize** (elsi_h)
(line 31 in Algorithm 1) – Terminates the ELSI instance associated with the handle. This deallocates any arrays internally allocated by ELSI.

  - **elsi_h** (type(elsi_handle), input & output): On exit, all the parameters of this handle are reset to "UNSET" or their default values. To become valid again, the handle must be re-initialized by elsi_init.

4.8. ELSI software in practice

The 2017.05 release of the ELSI software package, available on the "ELSI Interchange" website (http://elsi-interchange.org), contains the ELSI interface described in Section 4, as well as redistributed source code of the three solver libraries ELPA (version 2016.11.001.pre, http://elpa.mpcdf.mpg.de), libOMM (version 0.0.1, http://esl.cecam.org/LibOMM), and PEXSI (version 0.10.2, http://pexsi.org). They are redistributed with ELSI for an optional integrated installation managed by a unified make-based build system with specific keywords set by the users in "make.sys" files.

Table 1
Supercell size, number of atoms N_{atom}, number of basis functions N_{basis}, and sparsity factor $N_{\text{zero}}/N_{\text{basis}}^2$ of the graphene systems used in this work. N_{zero} is the number of zero elements in the Hamiltonian matrices. FHI-aims models contain 2 carbon atoms in each unit cell, and results are shown in Figs. 7–12 and A.14 and. DGDFT models contain 10 graphene layers (20 carbon atoms) in each unit cell, and results are shown in Fig. 13.

Code	Model	Supercell	N_{atom}	N_{basis}	$N_{\text{zero}}/N_{\text{basis}}^2$
FHI-aims	Graphene	$30 \times 30 \times 1$	1,800	25,200	97.50%
FHI-aims	Graphene	$35 \times 35 \times 1$	2,450	34,300	98.16%
FHI-aims	Graphene	$40 \times 40 \times 1$	3,200	44,800	98.58%
FHI-aims	Graphene	$45 \times 45 \times 1$	4,050	56,700	98.88%
FHI-aims	Graphene	$50 \times 50 \times 1$	5,000	70,000	99.09%
FHI-aims	Graphene	$55 \times 55 \times 1$	6,050	84,700	99.25%
FHI-aims	Graphene	$60 \times 60 \times 1$	7,200	100,800	99.41%
DGDFT	Graphene	$18 \times 18 \times 1$	6,480	97,200	99.98%
DGDFT	Graphene	$24 \times 24 \times 1$	11,520	172,800	99.99%

While we focus more on the development of a unified interface to connect the KS solvers and the KS-DFT codes, the ELPA, libOMM, and PEXSI solvers themselves are being actively developed by their own communities. The three solvers linked into ELSI can be either the built-in versions shipped with ELSI, or independently built versions, e.g. pre-installed and optimized versions available on a given supercomputer. There are two external dependencies that must be downloaded and installed separately: the ParMETIS library (http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview) and the SuperLU_DIST library (http://crd-legacy.lbl.gov/~xiaoye/SuperLU).

ELSI can be integrated directly into relevant pieces of KS-DFT codes written in Fortran, C, or C++. So far, ELSI has been tested in the DGDFT [52], FHI-aims [33], NWChem [26] (via Global Arrays Toolkit [74]), and SIESTA [36] software packages. Detailed instructions on how to obtain, install, and use the ELSI software are documented in the ELSI User's Guide [75].

5. Benchmarks and discussions

In the final part of this work, we present a comparative study of the three KS electronic structure solvers ELPA, libOMM, and PEXSI, as currently supported by ELSI. This study employs a consistent set of systems and settings, and illustrates the optimal choice of solver strategies in different scenarios and system size ranges. The Hamiltonian and overlap matrices are constructed from actual DFT-PBE [76] calculations using the all-electron, full-potential electronic structure code FHI-aims (Fortran) with a "tier 1" numeric atom-centered orbital (NAO) basis set [33,77], and the pseudopotential code DGDFT (C++) with an adaptive local basis (ALB) [51,52]. Both packages have been demonstrated to perform large-scale DFT calculations with at least thousands of atoms [15,52,78]. Details of the KS-DFT code specific settings are given in Appendices A and B, respectively. As the benchmark systems, we selected 2D graphene supercell models with sizes ranging from 1,800 to 11,520 atoms. All calculations reported here are Γ-point-only (the ELSI interface is thus in MULTI_PROC mode) and real arithmetic. Among the benchmark problems, the graphene $30 \times 30 \times 1$, $45 \times 45 \times 1$, and $60 \times 60 \times 1$ supercell models have a small band gap of about 0.002 meV, since the Dirac cone of graphene, whose coordinates in the reciprocal space are (1/3, 1/3, 0), is included in the folded images of the Γ point. The other graphene models have a band gap of $0.34 \sim 0.51$ eV. The dimensions of the models, the number of employed basis functions, and the sparsity factor of the corresponding matrices are reported in Table 1. The maximum differences in the converged total energies are 6.3 μeV/atom between the results obtained with ELPA and libOMM, and 0.8 μeV/atom between ELPA and PEXSI. We note that separate benchmarks of ELPA, libOMM, and PEXSI applied to insulating/semiconducting, 1D/3D systems have been reported in earlier publications [14–18].

We here report, to our knowledge, the first directly comparable benchmark of all three approaches for the same system and using exactly the same hardware and software environment. All computations were performed on the Cray XC30 supercomputer Edison at National Energy Research Scientific Computing Center (NERSC). Each node of Edison is equipped with two 12-core Intel Ivy Bridge processors. The nodes were fully exploited by launching 24 MPI tasks on each node. No multi-threading parallelization was employed.

5.1. Performance of the ELPA, libOMM, PEXSI solvers

We first compare the performance of the key computational steps of the ELSI solvers that are repeated in every SCF iteration. These repeated steps are: transforming the eigenproblem (Eq. (11)), solving the standard eigenproblem (Fig. 1), and back-transforming the eigenvectors in ELPA; the minimization (CG line search) of OMM energy functional (Eqs. (15) and (16)), and the construction of the density matrix from the final Wannier functions in libOMM; the numerical factorization and the selected inversion of the object $\boldsymbol{H}-(z_l+\mu)\boldsymbol{S}$ (Eq. (19)), and the construction of the density matrix from the poles in PEXSI. There are other computationally expensive steps that only occur in the first SCF iteration and have less significant effects on the total time of an SCF cycle. The performance of those steps is discussed separately in Sections 5.2 and 5.3. For reference, the performance of the remaining computational steps (in addition to the KS eigenproblem) of standard DFT-PBE calculations using FHI-aims code is shown in Appendix A.

Fig. 7 shows the wall clock time of the above-mentioned repeated steps of the solvers. It is worth noting that, when using the same computational resources, the time used by ELPA is theoretically constant during an SCF cycle, as the performance of a dense direct eigensolver only depends on the size of the matrix to solve. In contrast, the time used by libOMM and PEXSI depends on the number of CG line searches and the number of PEXSI mu iterations, respectively. Since both the number of CG line searches and the number of PEXSI mu iterations can be quickly reduced to 1 as the SCF cycle proceeds, shown in Fig. 7 are the timings corresponding to 1 CG line search in libOMM using the basic flavor (see Section 5.5 for the effect of the flavor on the performance of libOMM), and 1 PEXSI mu iteration in PEXSI. In future versions of PEXSI, a newly designed algorithm will be used to update the chemical potential as the SCF cycle converges, and the number of mu iterations will always be 1 in each SCF iteration.

In Fig. 7(a), the scaling of solvers with respect to the basis size is shown for DFT-PBE calculations of graphene models consisting of 1,800 atoms (25,200 basis functions) to 7,200 atoms (100,800 basis functions) using 1,920 MPI tasks. Both ELPA and libOMM exhibit scalings close to $O(N^3)$, as expected. In this particular setup, libOMM is consistently faster than ELPA by a factor of 2. PEXSI, with a lower computational complexity (theoretically $O(N^{1.5})$ for 2D systems), begins to outperform ELPA and libOMM at around

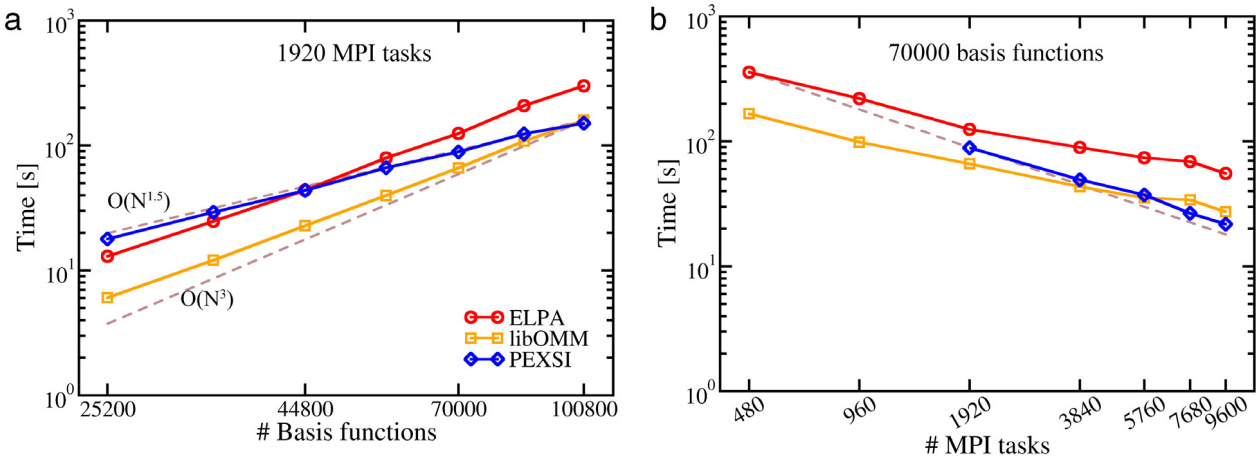

Fig. 7. Scaling of the repeated steps in ELPA, libOMM, and PEXSI solvers with respect to (a) the number of basis functions and (b) the number of MPI tasks. The number of MPI tasks in (a) is 1,920. The number of basis functions in (b) is 70,000. The repeated steps are: transforming the eigenproblem (Eq. (11)), solving the standard eigenproblem (Fig. 1), and back-transforming the eigenvectors in ELPA; the minimization of OMM energy functional (Eqs. (15) and (16)), and the construction of density matrix from the final Wannier functions in libOMM (the CG line search converges in one step); the numerical factorization and the selected inversion of the object $\boldsymbol{H} - (z_l + \mu)\boldsymbol{S}$ (Eq. (19)), and the construction of the density matrix from the poles in PEXSI (the PEXSI iteration converges in one step). Ideal scaling is indicated by the dashed lines. PEXSI cannot solve the problem of 70,000 basis functions (5,000 carbon atoms) with 480 or 960 MPI tasks, due to the limited amount of memory assigned to each pole. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3,000 atoms and 7,000 atoms, respectively. The benefit of using PEXSI should become more significant as we further increase the system size.

The strong scaling shown in Fig. 7(b) demonstrates the scalability of the solvers when they are applied to the graphene 5,000-atom model (70,000 basis functions) using 480 to 9,600 MPI tasks. All three solvers exhibit good scalability to 9,600 MPI tasks. In particular, the PEXSI solver scales almost ideally up to thousands of MPI tasks. This is attributed to the 2-level parallelism employed in PEXSI (Section 3.3). The perfect strong scaling of PEXSI can be further extended to at least tens of thousands of MPI tasks (this is demonstrated in Section 5.6). However, PEXSI fails to solve the problem with 480 or 960 MPI tasks, owing to the limited memory assigned to each pole.

5.2. Matrix redistribution

When using the elsi_dm_real subroutine (Section 4.5) to compute the density matrix with the BLACS_DENSE format and the PEXSI solver, the input Hamiltonian and overlap matrices are not in the correct format for PEXSI. The elsi_dm_real subroutine internally converts the input Hamiltonian matrix to the PEXSI_CSC format, and converts the density matrix computed by PEXSI back to the original format. The overlap matrix is converted as well, albeit only in the first iteration of an SCF cycle. The performance of the Hamiltonian matrix conversion from BLACS_DENSE to PEXSI_CSC and the density matrix conversion from PEXSI_CSC to BLACS_DENSE are shown and compared to the PEXSI computation time in Fig. 8. For matrix sizes ranging from 25,200 (1,800 atoms) to 100,800 (7,200 atoms), Fig. 8(a) shows that the wall clock time for both conversions with 1,920 MPI tasks is always below 10% of the PEXSI computation time (red lines in Fig. 8). Fig. 8(b) shows that the data redistribution time is consistently below 10% of the computation time, when using 1,920 to 9,600 MPI tasks for a problem of dimension 70,000. The BLACS_DENSE to PEXSI_CSC conversion stops scaling at 9,600 MPI tasks. Further optimization of the conversion using MPI point-to-point communications is planned as a future work direction.

5.3. SCF initialization

Computational steps that are only required in the first one or few SCF iterations have some impact on the overall performance of an SCF cycle. Here we discuss three such steps: (1) Cholesky factorization of the overlap matrix in Eq. (10), which is used to transform the generalized eigenvalue problem to the standard form. This is a mandatory step for ELPA and an optional step for libOMM. The Cholesky factorization of a dense matrix in ELSI is performed using subroutines provided in ELPA. (2) Symbolic factorization that provides PEXSI with necessary information of the sparsity pattern of the Hamiltonian and overlap matrices before numerical factorization and selected inversion are carried out. The symbolic factorization of a sparse matrix is performed using subroutines provided in the SuperLU_DIST library [79,80]. (3) Inertia counting that quickly estimates the chemical potential of the system according to Sylvester's Inertia Law theorem [71]. This reasonable initial guess of the chemical potential is essential to the fast convergence of PEXSI.

Fig. 9(a) shows the wall clock time of the three initialization steps as a function of the system size. The Cholesky factorization of a dense matrix using ELPA subroutines scales cubically with the system size, whereas the symbolic factorization and inertia counting scale linearly. The scaling difference among these preprocessing steps helps explain why PEXSI is more favorable for large systems. In the strong scaling plot shown in Fig. 9(b), the dense Cholesky factorization is shown to scale up to 9,600 MPI tasks. Because the symbolic factorization implemented in SuperLU_DIST is not stable when executed on multiple processors, we used a sequential version of the symbolic factorization in the experiment, which obviously does not scale. We are in the process of developing a more robust and scalable implementation of the symbolic factorization procedure as part of the development of a new parallel sparse Cholesky (and LDLT) factorization library called symPACK [81].

5.4. ELPA

To analyze the performance of the ELPA eigensolver for the graphene problem solved here, the solution of a generalized eigenproblem (red lines in Fig. 7) is divided into three steps: the transformation of the generalized eigenproblem to the standard form (Eq. (11)), the solution of the standard eigenproblem (Fig. 1), and the back-transformation of the eigenvectors. Fig. 10(a) and (b) show the scaling of the three steps with respect to the number of basis functions and the number of MPI tasks, respectively. All these

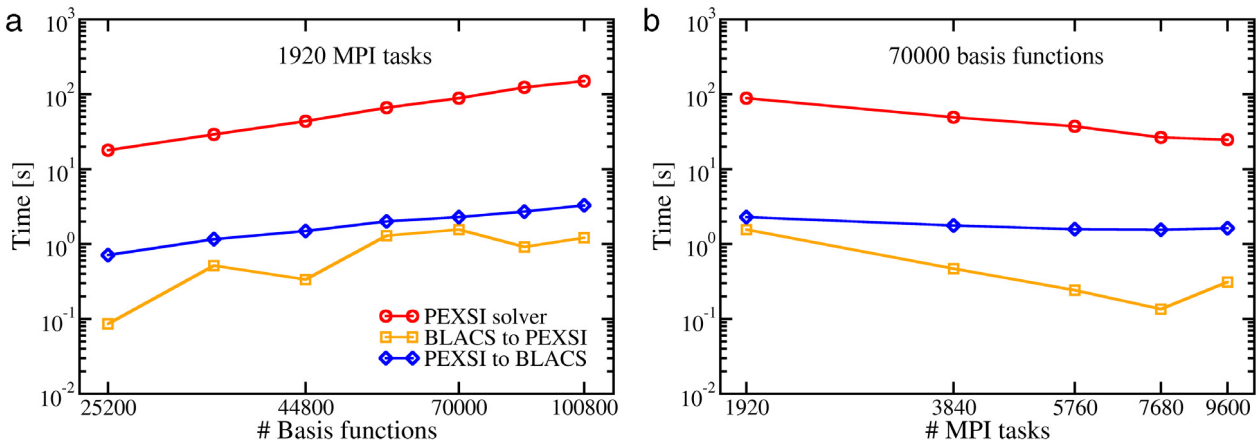

Fig. 8. Scaling of matrix redistribution with respect to (a) the number of basis functions and (b) the number of MPI tasks. The number of MPI tasks in (a) is 1,920. The number of basis functions in (b) is 70,000. BLACS to PEXSI: redistribution of the Hamiltonian matrix from 2D block-cyclic dense storage (BLACS_DENSE) to 1D block CSC sparse storage (PEXSI_CSC). PEXSI to BLACS: redistribution of the density matrix from 1D block CSC sparse storage (PEXSI_CSC) to 2D block-cyclic dense storage (BLACS_DENSE). The overlap matrix is redistributed only once per SCF cycle, hence its absence here. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

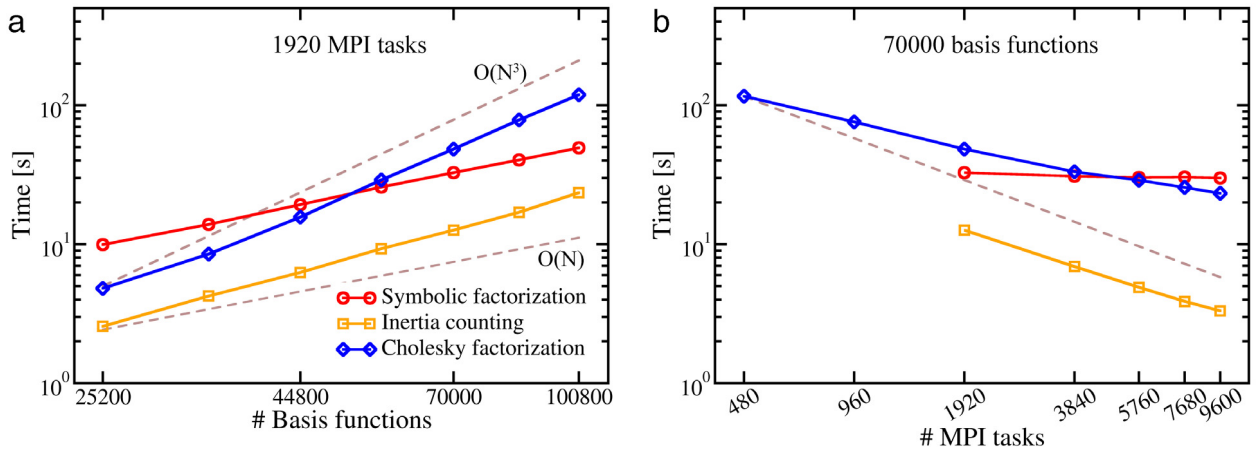

Fig. 9. Scaling of symbolic factorization using SuperLU_DIST, inertia counting using PEXSI, and Cholesky factorization using ELPA, with respect to (a) the number of basis functions and (b) the number of MPI tasks. The number of MPI tasks in (a) is 1,920. The number of basis functions in (b) is 70,000. Symbolic factorization is performed in serial. Ideal scaling is indicated by the dashed lines.

steps scale cubically with respect to the system size. Solving the standard eigenproblem is more expensive than the transformation steps. While the three steps show similar strong scaling up to 9,600 MPI tasks, the solution of a standard eigenproblem dominates the total computation time. In Fig. 10(c) and (d), the solution time of a standard eigenproblem using ELPA 2-stage solver is further decomposed into five steps illustrated in Fig. 1. These plots show that the current bottlenecks in terms of both computation time and parallel efficiency are the first step, i.e. the transformation of a full matrix to a banded form, and the fifth step, i.e. the back-transformation of the eigenvectors from a banded form to a full form. The fourth step, back-transformation of the eigenvectors to the banded form, is not the most time consuming step of the computation. In fact, the computational complexity of the third, fourth, and fifth steps is roughly proportional to the number of eigenvectors to compute, as only these eigenvectors need to be calculated in the third step and transformed in the fourth and fifth steps.

5.5. libOMM

The performance of the iterative OMM method depends significantly on the convergence rate of the CG minimization. The prototype OMM implementation in libOMM generates random numbers as the initial guess for the coefficients of Wannier functions used in the first SCF iteration, consequently leading to a large and unpredictable number of iterations in the CG line search scheme. Then, the convergence of line search is dramatically accelerated as the SCF cycle proceeds, as the Wannier functions coefficients calculated in the current iteration are reused as the initial guess in the next iteration. Inspired by the connection between the Wannier functions and the basis functions in Eq. (14), a better idea is to use the eigenfunctions corresponding to the occupied space computed by ELPA as the initial guess for OMM. In ELSI, this is achieved automatically, controlled by the n_elpa_steps parameter (see Section 4.6). Table 2 reflects how n_elpa_steps affects the CG convergence of OMM in the (n_elpa_steps + 1)th SCF iteration, by showing the number of CG line searches in the basic and Cholesky flavors of OMM as a function of the number of ELPA steps. In general, more ELPA steps lead to faster CG convergence. In this particular test case with 5,000 carbon atoms and 70,000 basis functions, 6 ELPA steps are sufficient to reduce the number of CG line searches in libOMM to 1 for both tested flavors.

Compared in the second and third rows of Table 2 is another factor that has an impact on the number of CG line searches in libOMM, i.e., the method used to minimize the OMM functional. The basic algorithm directly follows the recipe in Eq. (16), but

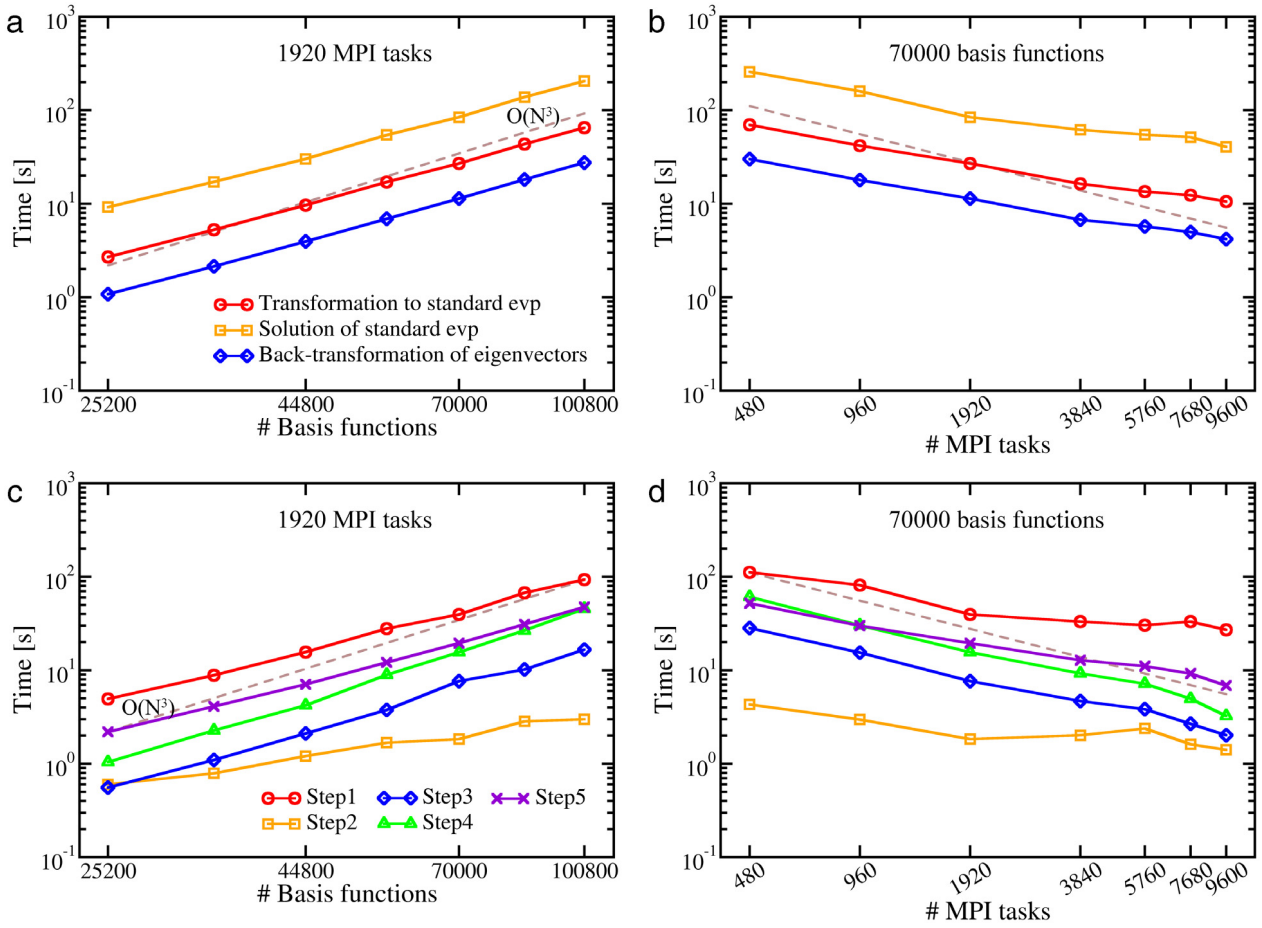

Fig. 10. Scaling of the key computational steps of the ELPA eigensolver with respect to (a, c) the number of basis functions and (b, d) the number of MPI tasks. The number of MPI tasks in (a) is 1,920. The number of basis functions in (b) is 70,000. Ideal scaling is indicated by the dashed lines. The upper panel (a,b) focuses on the transformation from a generalized eigenproblem to its standard form, the solution of a standard problem, and the back-transformation of the eigenvectors to the original generalized problem. The lower panel (c, d) further decomposes the solution of a standard eigenproblem using the ELPA 2-stage solver into 5 substeps, as illustrated in Fig. 1.

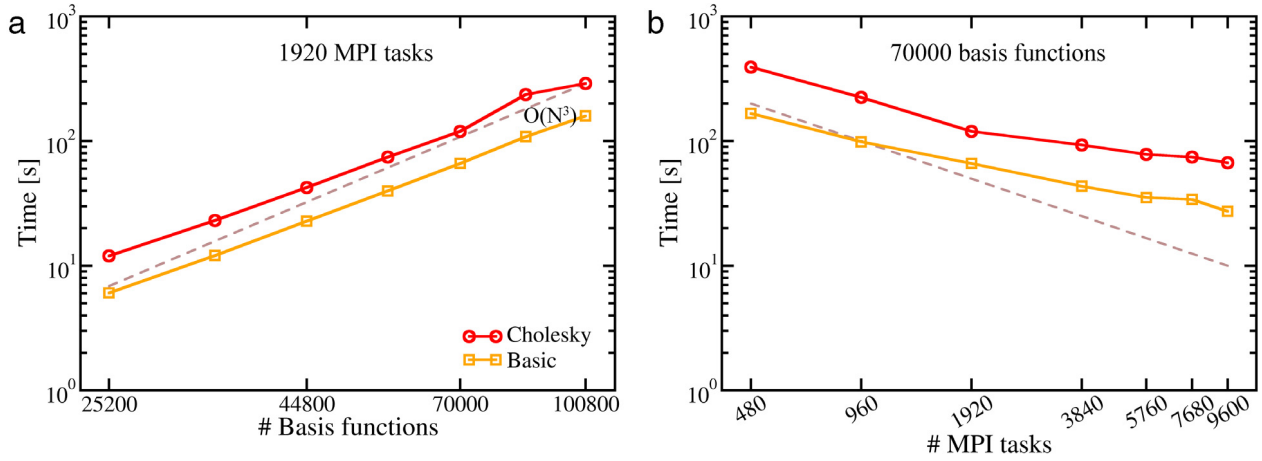

Fig. 11. Scaling of the computation of the density matrix using orbital minimization method, with respect to (a) the number of basis functions and (b) the number of MPI tasks. The number of MPI tasks in (a) is 1,920. The number of basis functions in (b) is 70,000. Shown here is the ideal case of OMM, where the CG line search of the OMM energy functional minimum requires only one step to converge. In practical SCF calculations, the number of line searches in OMM can only be reduced to one after several SCF steps. "Basic" refers to the method that directly handles the generalized eigenproblem. "Cholesky" refers to the method that applies Cholesky factorization to transform the generalized problem to the standard form before minimization. Ideal scaling is indicated by the dashed line.

Eq. (16) can also be minimized by first transforming the generalized eigenproblem to a standard problem based on Cholesky factorization. As shown in Table 2, minimizing the OMM functional in the context of a standard eigenproblem (Cholesky, the third row in the table) contributes to a decrease in the number of line searches. This acceleration of the CG line search, however, comes at the price of the additional complexity required by the eigenproblem transformation. Fig. 11 shows the comparison of the

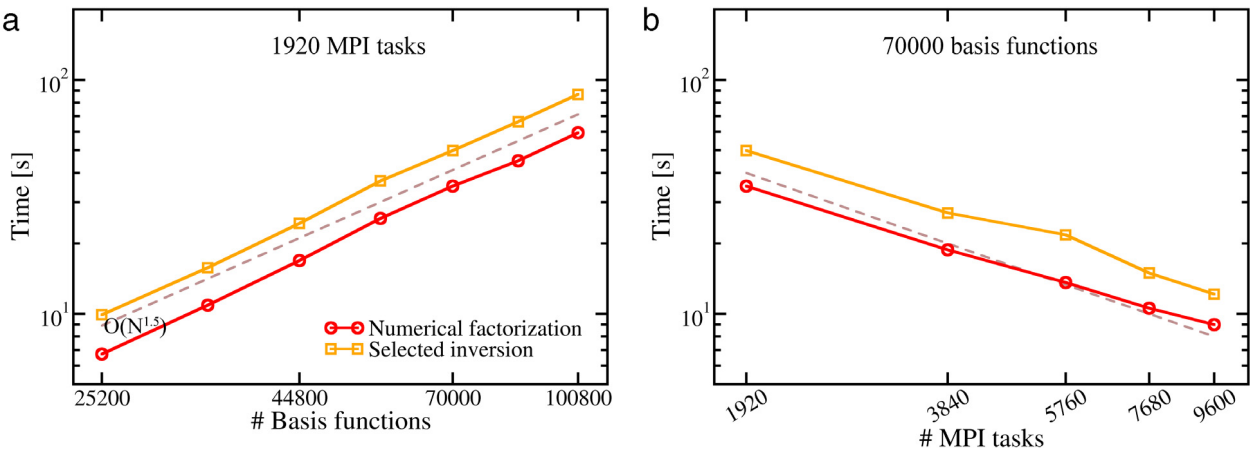

Fig. 12. Scaling of the two key computational steps of the PEXSI DFT driver, namely the numerical factorization and the selected inversion, with respect to (a) the number of basis functions and (b) the number of MPI tasks. The number of MPI tasks in (a) is 1,920. The number of basis functions in (b) is 70,000. Ideal scaling is indicated by the dashed lines. The 80 poles employed for the pole expansion in Eq. (19) are independently evaluated in parallel. The numerical factorization and selected inversion of each pole are carried out using 1,920/80 = 24 MPI tasks in (a), and # MPI tasks/80 in (b).

Table 2
Number of conjugate gradient (CG) line search steps required by libOMM to minimize the OMM energy functional. The benchmark system here is the graphene $50 \times 50 \times 1$ supercell model containing 5,000 atoms and 70,000 basis functions. In the table, "Basic" refers to the method that directly operates on the generalized eigenproblem; "Cholesky" refers to the method that applies Cholesky factorization to transform the generalized problem to the standard form. "x" in the second column means that the minimization cannot converge within the maximum allowed number of CG iterations (5,000).

# ELPA steps	0	1	2	3	4	5	6	7
# CG (Basic)	x	185	255	153	72	7	1	1
# CG (Cholesky)	254	27	36	24	13	5	1	1

computational time of one CG line search in libOMM with the basic flavor versus the Cholesky flavor. The two flavors scale similarly, with respect to both the number of basis functions (Fig. 11(a), from 25,200 to 100,800 basis functions) and the number of MPI tasks (Fig. 11(b), from 480 to 9,600 MPI tasks). The Cholesky flavor is consistently slower than the basic flavor by a factor of $2 \sim 4$, due to the eigenproblem transformation and the corresponding back-transformation of Wannier function coefficients. Also reflected in Fig. 11 is the shortest time to compute the density matrix using OMM, which is the basic flavor that converges in one CG line search. Indicated by Table 2 and Fig. 11, the most promising approach that could be used in practical calculations is the combination of a few ELPA steps followed by the basic flavor of OMM, whose convergence is guaranteed within one CG iteration. To further improve the performance of this solver, future work will include the inclusion in the ELSI interface of a preconditioned libOMM flavor, which has already proven to efficiently speed up the line search convergence [16,82]; a spectral slicing method to separately evaluate the eigenstates near the Fermi level and thus to enable the proper handling of fractional occupation numbers; the sparse linear algebra via routines implemented in the PSPBLAS (Parallel SParse BLAS) library [83]; and ultimately the extension of OMM to a linear scaling solver as originally proposed [61–64].

5.6. PEXSI

As noted in Section 3.3, PEXSI exploits two levels of parallelization: the first level is the parallel evaluation of each pole in the pole expansion (Eq. (19)), and the second level is the parallel numerical factorization and selected inversion at each pole. MPI tasks are divided into several groups with one pole assigned to each group. Fig. 12(a) shows that both steps scale as $O(N^{1.5})$ for the graphene model, which is in agreement with the theoretical prediction for quasi-2D systems. The selected inversion step is slightly more expensive than the numerical factorization step. As shown in the strong scaling plot in Fig. 12(b), both the numerical factorization and the selected inversion scale almost ideally to at least 9,600 MPI tasks. The number of MPI tasks shown in Fig. 12(b) should be divided by the number of poles, 80, to reflect the scaling of numerical factorization and selected inversion at each pole. Since PEXSI has been shown to scale to several thousands of MPI tasks [70], the performance reported in Fig. 12(b), which measures scalability up to 120 tasks per pole, is still far from the scalability limit. To further demonstrate the strong scaling of the PEXSI solver, Fig. 13 shows the wall clock time used by PEXSI for a graphene model consisting of 6,480 atoms (97,200 basis functions) using 2,592 to 31,104 MPI tasks (Fig. 13(a)) and a graphene model consisting of 11,520 atoms (172,800 basis functions) using 2,304 to 110,592 MPI tasks (Fig. 13(b)). These tests are performed using the ELSI interface as implemented in the DGDFT software package. The ELPA eigensolver is also included as a reference. For both models, PEXSI exhibits nearly ideal strong scaling and eventually outperforms ELPA as the number of MPI tasks becomes sufficiently large. The ELPA solver ceases to scale beyond 18,432 MPI tasks for the 172,800-atom system. We note that we have not investigated the origin of the increase of ELPA timings for 55,296 and 110,592 MPI tasks, respectively. Thus, these data points are shown in Fig. 13 for comparison only, reflecting the observed behavior on the particular supercomputer (Edison) used at the time of writing. In particular, we do not know whether this behavior of ELPA is specific to this supercomputer and/or to the specific setup used by us.

6. Conclusions

Materials simulations based on Kohn–Sham density-functional theory require solving an eigenvalue problem repeatedly in an iterative procedure designed to obtain the ground state electron density of a poly-atomic system. Although this is a well studied subject in numerical linear algebra, it constitutes the bottleneck in large-scale calculations. A number of new approaches have emerged in the last few years. These approaches have different features and performance characteristics. Proper use of these approaches requires a good understanding of the pros and cons of each approach, and the input and output of specific algorithms. ELSI is designed to provide a common interface that allows users to easily choose an appropriate solver. Although the choice of the best

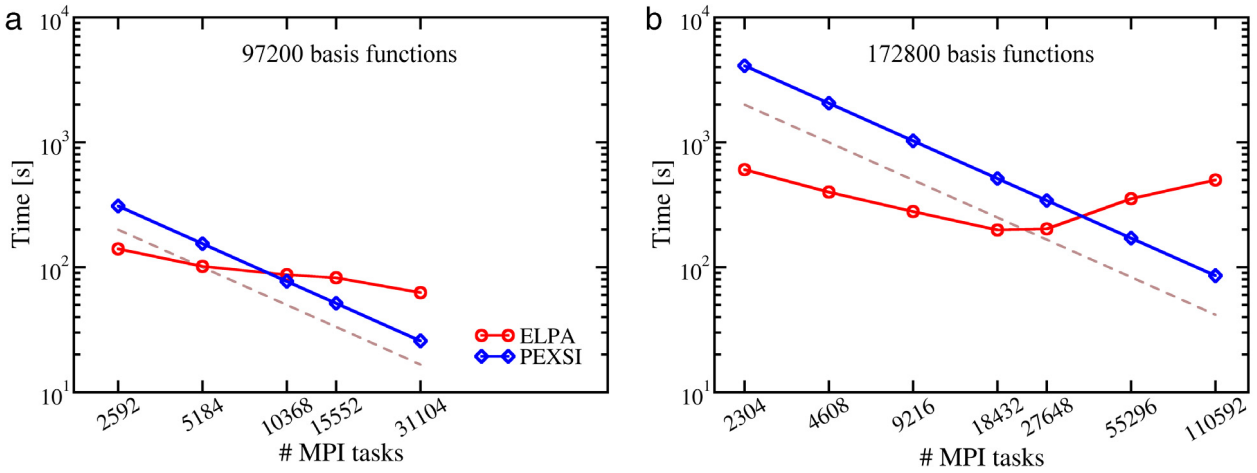

Fig. 13. Comparison of the strong scaling of the ELPA and PEXSI solvers. The number of basis functions is 97,200 in (a) and 172,800 in (b). The matrices are from DGDFT code. There are 48 poles employed in the PEXSI pole expansion. Ideal scaling is indicated by the dashed lines.

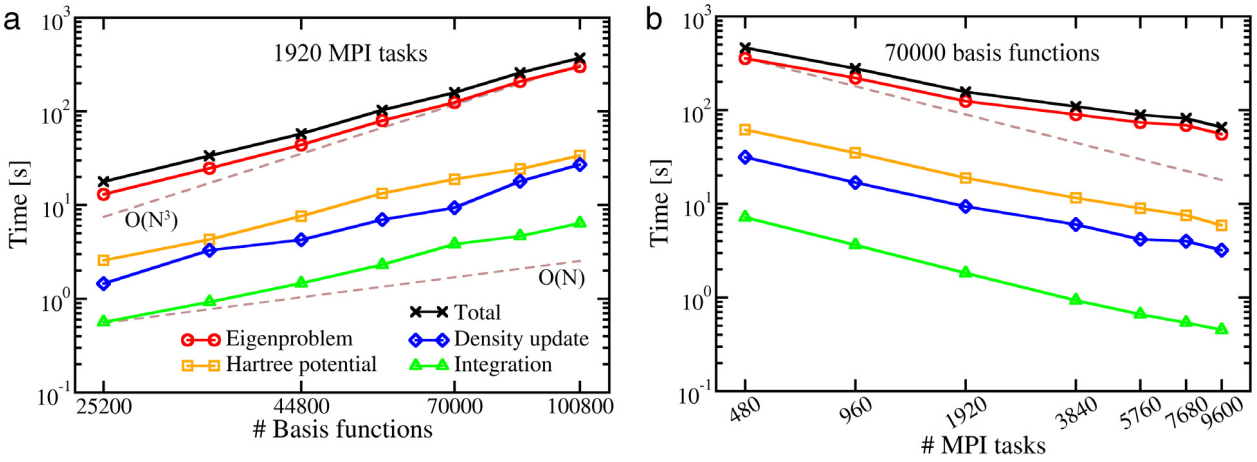

Fig. A.14. Scaling of the key computational steps of the DFT-PBE calculations in FHI-aims, with respect to (a) the number of basis functions and (b) the number of MPI tasks. The number of MPI tasks in (a) is 1,920. The number of basis functions in (b) is 70,000. Ideal scaling is indicated by the dashed lines. The key steps are the evaluation of the Hartree potential, the numerical integrations of the Hamiltonian matrix elements, the update of the electron density and its gradient, and solving the Kohn–Sham eigenproblem using the ELPA eigensolver library. Shown here are timings corresponding to one SCF iteration, not accumulated timings in a complete SCF cycle.

solver often depends on a number of factors such as the problem size and the available computational resources, the benchmark results presented in this paper provide some general guidance on how to make these choices. In particular, we have shown different regimes in which one approach outperforms others and the crossover points between these regimes.

Finally, we demonstrated how different solvers can be organized in a common framework to enable easy integration with a vast number of electronic structure software packages. We anticipate that the number of new approaches to solving eigenvalue problems related to KS-DFT will continue to increase. We hope that ELSI will become a focal point for the community to integrate, comparatively assess and, ultimately, adopt this diverse ecosystem in a simple, effective fashion.

Acknowledgments

This work was supported by the National Science Foundation under grant number 1450280. We thank the National Energy Research Scientific Computing Center (NERSC) for the computational resources. This work also used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357. Regarding the establishment and improvement of the ELSI API, we especially appreciate the fruitful discussions with and the feedback from many fellow researchers in the electronic structure community, including developers of the BigDFT, CP2K, DGDFT, FHI-aims, SIESTA code projects and of CECAM's Electronic Structure Library (http://esl.cecam.org). AG thanks EU H2020 grant 676598 ('MaX: Materials at the eXascale' CoE), Spain's MINECO (FIS2012-37549-C05-05, FIS2015-64886-C5-4-P and the 'Severo Ochoa' program grant SEV-2015-0496), and GenCat (2014 SGR 301). The work of JL was partially supported by the National Science Foundation under grant number DMS-1127914 to the Statistical and Applied Mathematical Sciences Institute. VB particularly acknowledges the experiences shared by many of the co-authors of the FHI-aims code over many years, for instance, regarding parallelization strategies or the handling of ill-conditioned overlap matrices, from which we learned during the development of the ELSI interface — especially Dr. Ville Havu (Aalto University) and Dr. Rainer Johanni (Munich; now deceased).

Appendix A. Technical settings in FHI-aims calculations

The benchmark calculations reported in Figs. 7–12 in Section 5 are KS-DFT calculations performed with the FHI-aims code [33,77],

PBE exchange–correlation functional [76], "tier1" numeric atom-centered orbital (NAO) basis set (see Table 1 in Ref. [33], "light" numerical settings, and a $1 \times 1 \times 1$ k-grid (Γ point). In order to place the timings reported in Fig. 7 into perspective with respect to the other parts of a KS-DFT calculation, Fig. A.14 shows timings for all other important computational steps in the corresponding FHI-aims calculations, obtained on the same hardware and in the same runs as the results shown in Fig. 7. The main additional steps are executed on a real-space grid and include the Hartree potential evaluation, the numerical integrations of the Hamiltonian matrix elements, and the update of the electron density and its gradients, all implemented in a near $O(N)$ fashion and efficiently parallelized in FHI-aims. Refs. [33,77] provide a more detailed account of the algorithms involved.

Appendix B. Technical settings in DGDFT calculations

The benchmark calculations reported in Fig. 13 in Section 5 are KS-DFT calculations performed with DGDFT [51,52] using the PBE exchange–correlation functional [76]. The global system is partitioned into 36×36 and 48×48 elements for the system containing 6,480 and 11,520 atoms, respectively. The number of adaptive local basis functions (ALB) per atom is 15, which is sufficient for the error of the total energy per atom and the maximum error of the force to be below 10^{-3} Hartree and 10^{-3} Hartree/Bohr, respectively. The DG penalty parameter is chosen to be 5.0, and the kinetic energy cutoff to generate the ALBs is set to 40 Hartree. The number of poles used by PEXSI is 48.

References

[1] W. Kohn, L.J. Sham, Phys. Rev. 140 (4A) (1965) 1133–1138.
[2] A.D. Becke, J. Chem. Phys. 98 (2) (1993) 1372–1377.
[3] A. Seidl, A. Görling, P. Vogl, J.A. Majewski, M. Levy, Phys. Rev. B 53 (1996) 3764–3774.
[4] L.M. Ghiringhelli, C. Carbogno, S. Levchenko, F. Mohamed, G. Huhs, M. Lueders, M. Oliveira, M. Scheffler, Psi-K Scientific Highlight of the Month 131 (July) (2016) 1.
[5] R. Haunschild, A. Barth, W. Marx, J. Cheminform. 8 (1) (2016) 52.
[6] P. Mavropoulos, P. Dederichs, Psi-K Scientific Highlight of the Month 135 (April) (2017) 1.
[7] S. Goedecker, Rev. Modern Phys. 71 (1999) 1085–1123.
[8] D.R. Bowler, T. Miyazaki, Rep. Progr. Phys. 75 (3) (2012) 036503.
[9] C.-K. Skylaris, P.D. Haynes, A.A. Mostofi, M.C. Payne, J. Chem. Phys. 122 (8) (2005) 084119.
[10] D.R. Bowler, T. Miyazaki, J. Phys.: Condens. Matter 22 (7) (2010) 074207.
[11] J. VandeVondele, U. Borštnik, J. Hutter, J. Chem. Theory Comput. 8 (10) (2012) 3565–3573.
[12] H.F. Schurkus, C. Ochsenfeld, J. Chem. Phys. 144 (3) (2016) 031101.
[13] A. Luenser, H.F. Schurkus, C. Ochsenfeld, J. Chem. Theory Comput. 13 (4) (2017) 1647–1655.
[14] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, P.-R. Willems, Parallel Comput. 37 (12) (2011) 783–794.
[15] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, H. Lederer, J. Phys.: Condens. Matter 26 (21) (2014) 213201.
[16] F. Corsetti, Comput. Phys. Comm. 185 (3) (2014) 873–883.
[17] L. Lin, J. Lu, L. Ying, R. Car, W. E, Commun. Math. Sci. 7 (3) (2009) 755–777.
[18] L. Lin, M. Chen, C. Yang, L. He, J. Phys.: Condens. Matter 25 (29) (2013) 295501.
[19] L. Lin, A. García, G. Huhs, C. Yang, J. Phys.: Condens. Matter 26 (2014) 305503.
[20] S. Mohr, D. Caliste, M. Wagner, L. Genovese, Efficient computation of sparse matrix functions for large scale electronic structure calculations: The CheSS library arXiv:1704.00512, 2017.
[21] M. Keçeli, H. Zhang, P. Zapol, D.A. Dixon, A.F. Wagner, J. Comput. Chem. 37 (4) (2016) 448–459.
[22] S.R. Jensen, S. Saha, J.A. Flores-Livas, W. Huhn, V. Blum, S. Goedecker, L. Frediani, J. Phys. Chem. Lett. 8 (7) (2017) 1449–1457.
[23] A. Szabo, N.S. Ostlund, Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory, Dover Publications, 1989.
[24] J. Hutter, M. Iannuzzi, F. Schiffmann, J. VandeVondele, Wiley Interdisciplinary Rev.: Comput. Mol. Sci. 4 (1) (2014) 15–25.
[25] M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, G. Scalmani, V. Barone, G.A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. Marenich, J. Bloino, B.G. Janesko, R. Gomperts, B. Mennucci, H.P. Hratchian, J.V. Ortiz, A.F. Izmaylov, J.L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V.G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J.A. Montgomery Jr., J.E. Peralta, F. Ogliaro, M. Bearpark, J.J. Heyd, E. Brothers, K.N. Kudin, V.N. Staroverov, T. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J.C. Burant, S.S. Iyengar, J. Tomasi, M. Cossi, J.M. Millam, M. Klene, C. Adamo, R. Cammi, J.W. Ochterski, R.L. Martin, K. Morokuma, O. Farkas, J.B. Foresman, D.J. Fox, Gaussian09 Revision A.02, Gaussian Inc., Wallingford, 2016.
[26] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, W.A. de Jong, Comput. Phys. Comm. 181 (9) (2010) 1477–1489.
[27] Y. Shao, Z. Gan, E. Epifanovsky, A.T.B. Gilbert, M. Wormit, J. Kussmann, A.W. Lange, A. Behn, J. Deng, X. Feng, D. Ghosh, M. Goldey, P.R. Horn, L.D. Jacobson, I. Kaliman, R.Z. Khaliullin, T. Kuś, A. Landau, J. Liu, E.I. Proynov, Y.M. Rhee, R.M. Richard, M.A. Rohrdanz, R.P. Steele, E.J. Sundstrom, H.L. Woodcock III, P.M. Zimmerman, D. Zuev, B. Albrecht, E. Alguire, B. Austin, G.J.O. Beran, Y.A. Bernard, E. Berquist, K. Brandhorst, K.B. Bravaya, S.T. Brown, D. Casanova, C.-M. Chang, Y. Chen, S.H. Chien, K.D. Closser, D.L. Crittenden, M. Diedenhofen, R.A. DiStasio Jr., H. Do, A.D. Dutoi, R.G. Edgar, S. Fatehi, L. Fusti-Molnar, A. Ghysels, A. Golubeva-Zadorozhnaya, J. Gomes, M.W.D. Hanson-Heine, P.H.P. Harbach, A.W. Hauser, E.G. Hohenstein, Z.C. Holden, T.-C. Jagau, H. Ji, B. Kaduk, K. Khistyaev, J. Kim, J. Kim, R.A. King, P. Klunzinger, D. Kosenkov, T. Kowalczyk, C.M. Krauter, K.U. Lao, A.D. Laurent, K.V. Lawler, S.V. Levchenko, C.Y. Lin, F. Liu, E. Livshits, R.C. Lochan, A. Luenser, P. Manohar, S.F. Manzer, S.-P. Mao, N. Mardirossian, A.V. Marenich, S.A. Maurer, N.J. Mayhall, E. Neuscamman, C.M. Oana, R. Olivares-Amaya, D.P. O'Neill, J.A. Parkhill, T.M. Perrine, R. Peverati, A. Prociuk, D.R. Rehn, E. Rosta, N.J. Russ, S.M. Sharada, S. Sharma, D.W. Small, A. Sodt, T. Stein, D. Stück, Y.-C. Su, A.J.W. Thom, T. Tsuchimochi, V. Vanovschi, L. Vogt, O. Vydrov, T. Wang, M.A. Watson, J. Wenzel, A. White, C.F. Williams, J. Yang, S. Yeganeh, S.R. Yost, Z.-Q. You, I.Y. Zhang, X. Zhang, Y. Zhao, B.R. Brooks, G.K.L. Chan, D.M. Chipman, C.J. Cramer, W.A. Goddard III, M.S. Gordon, W.J. Hehre, A. Klamt, H.F. Schaefer III, M.W. Schmidt, C.D. Sherrill, D.G. Truhlar, A. Warshel, X. Xu, A. Aspuru-Guzik, R. Baer, A.T. Bell, N.A. Besley, J.-D. Chai, A. Dreuw, B.D. Dunietz, T.R. Furlani, S.R. Gwaltney, C.-P. Hsu, Y. Jung, J. Kong, D.S. Lambrecht, W. Liang, C. Ochsenfeld, V.A. Rassolov, L.V. Slipchenko, J.E. Subotnik, T.V. Voorhis, J.M. Herbert, A.I. Krylov, P.M.W. Gill, M. Head-Gordon, Mol. Phys. 113 (2) (2015) 184–215.
[28] F. Furche, R. Ahlrichs, C. Hättig, W. Klopper, M. Sierka, F. Weigend, Wiley Interdisciplinary Rev.: Comput. Mol. Sci. 4 (2) (2014) 91–100.
[29] J.C. Slater, Phys. Rev. 36 (1930) 57–64.
[30] G. te Velde, F.M. Bickelhaupt, E.J. Baerends, C.F. Guerra, S.J.A. van Gisbergen, J.G. Snijders, T. Ziegler, J. Comput. Chem. 22 (9) (2001) 931–967.
[31] http://www.quantumwise.com, (Accessed: 2017-04-27).
[32] B. Delley, J. Chem. Phys. 92 (1) (1990) 508.
[33] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, M. Scheffler, Comput. Phys. Comm. 180 (2009) 2175–2196.
[34] K. Koepernik, H. Eschrig, Phys. Rev. B 59 (3) (1999) 1743.
[35] T. Ozaki, Phys. Rev. B 67 (15) (2003) 155108.
[36] J.M. Soler, E. Artacho, J.D. Gale, A. García, J. Junquera, P. Ordejón, D. Sánchez-Portal, J. Phys.: Condens. Matter 14 (11) (2002) 2745–2779.
[37] D.J. Singh, Planewaves, Pseudopotentials and the LAPW Method, Springer, US, 1994.
[38] http://elk.sourceforge.net, (Accessed: 2017-04-27).
[39] A. Gulans, S. Kontur, C. Meisenbichler, D. Nabok, P. Pavone, S. Rigamonti, S. Sagmeister, U. Werner, C. Draxl, J. Phys.: Condens. Matter 26 (36) (2014) 363202.
[40] S. Blügel, G. Bihlmayer, D. Wortmann, C. Friedrich, M. Heide, M. Lezaic, F. Freimuth, M. Betzinger, The Jülich FLEUR project, Jülich Research Center, 1987.
[41] P. Blaha, K. Schwarz, G.K.H. Madsen, D. Kvasnicka, J. Luitz, WIEN2k: An augmented plane wave + local orbitals program for calculating crystal properties, 2001.
[42] J.E. Pask, B.M. Klein, P.A. Sterne, C.Y. Fong, Comput. Phys. Comm. 135 (1) (2001) 1–34.
[43] R.M. Martin, Electronic Structure: Basic Theory and Practical Methods, Cambridge University Press, 2004.
[44] X. Gonze, B. Amadon, P.-M. Anglade, J.-M. Beuken, F. Bottin, P. Boulanger, F. Bruneval, D. Caliste, R. Caracas, M. Côté, T. Deutsch, L. Genovese, P. Ghosez, M. Giantomassi, S. Goedecker, D.R. Hamann, P. Hermet, F. Jollet, G. Jomard, S. Leroux, M. Mancini, S. Mazevet, M.J.T. Oliveira, G. Onida, Y. Pouillon, T. Rangel, G.-M. Rignanese, D. Sangalli, R. Shaltaf, M. Torrent, M.J. Verstraete, G. Zerah, J.W. Zwanziger, Comput. Phys. Comm. 180 (12) (2009) 2582–2615.
[45] S.J. Clark, M.D. Segall, C.J. Pickard, P.J. Hasnip, M.I.J. Probert, K. Refson, M.C. Payne, Z. Kristallogr. 220 (2005) 567–570.
[46] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G.L. Chiarotti, M. Cococcioni, I. Dabo, A.D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A.P. Seitsonen, A. Smogunov, P. Umari, R.M. Wentzcovitch, J. Phys.: Condens. Matter 21 (39) (2009) 395502.

[47] G. Kresse, J. Furthmüller, Phys. Rev. B 54 (1996) 11169–11186.
[48] S. Mohr, L.E. Ratcliff, L. Genovese, D. Caliste, P. Boulanger, S. Goedecker, T. Deutscha, Phys. Chem. Chem. Phys. 17 (47) (2015) 31360–31370.
[49] R.J. Harrison, G. Beylkin, F.A. Bischoff, J.A. Calvin, G.I. Fann, J. Fosso-Tande, D. Galindo, J.R. Hammond, R. Hartman-Baker, J.C. Hill, J. Jia, J.S. Kottmann, M.-J.Y. Ou, J. Pei, L.E. Ratcliff, M.G. Reuter, A.C. Richie-Halford, N.A. Romero, H. Sekino, W.A. Shelton, B.E. Sundahl, W.S. Thornton, E.F. Valeev, A. Vázquez-Mayagoitia, N. Vence, T. Yanai, Y. Yokoi, SIAM J. Sci. Comput. 38 (5) (2016) S123–S142.
[50] http://mrchemdoc.readthedocs.org, (Accessed: 2017-05-21).
[51] L. Lin, J. Lu, L. Ying, W. E, J. Comput. Phys. 231 (4) (2012) 2140–2154.
[52] W. Hu, L. Lin, C. Yang, J. Chem. Phys. 143 (2015) 124110.
[53] J. Bernholc, M. Hodak, W. Lu, J. Phys.: Condens. Matter 20 (29) (2008) 294205.
[54] L. Kronik, A. Makmal, M.L. Tiago, M.M.G. Alemany, M. Jain, X. Huang, Y. Saad, J.R. Chelikowsky, Phys. Status Solidi (B) 243 (5) (2006) 1063–1079.
[55] G.H. Golub, C.F.V. Loan, Matrix Computations, 3rd edition, Johns Hopkins University Press, 2013.
[56] C.H. Bischof, B. Lang, X. Sun, ACM Trans. Math. Software 26 (4) (2000) 602–616.
[57] J.J.M. Cuppen, Numer. Math. 36 (2) (1980) 177–195.
[58] C.H. Bischof, X. Sun, B. Lang, (1994) Parallel tridiagonalization through two-step band reduction, in: Proceedings of IEEE Scalable High Performance Computing Conference, pp. 23–27.
[59] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R.C. Whaley, ScaLAPACK users' guide, SIAM, 1997.
[60] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, R.A. van de Geijn, Distributed Memory Computing Conference, 1991. Proceedings., the Sixth, IEEE, 1991, pp. 287–290.
[61] F. Mauri, G. Galli, R. Car, Phys. Rev. B 47 (15) (1993) 9973.
[62] P. Ordejón, D.A. Drabold, M.P. Grumbach, R.M. Martin, Phys. Rev. B 48 (1993) 14646–14649.
[63] F. Mauri, G. Galli, Phys. Rev. B 50 (1994) 4316–4326.
[64] P. Ordejón, D.A. Drabold, R.M. Martin, M.P. Grumbach, Phys. Rev. B 51 (3) (1995) 1456.
[65] M.P. Teter, M.C. Payne, D.C. Allan, Phys. Rev. B 40 (1989) 12255.
[66] M.C. Payne, M.P. Teter, D.C. Allan, T.A. Arias, J.D. Joannopoulos, Rev. Modern Phys. 64 (1992) 1045–1097.
[67] J. Kim, F. Mauri, G. Galli, Phys. Rev. B 52 (1995) 1640–1648.
[68] N.D. Mermin, E. Canel, Ann. Phys. 26 (2) (1964) 247–273.
[69] L. Lin, C. Yang, J. Meza, J. Lu, L. Ying, W. E, ACM Trans. Math. Software 37 (2011) 40.
[70] M. Jacquelin, L. Lin, C. Yang, ACM Trans. Math. Software 43 (3) (2016) 21.
[71] J.J. Sylvester, Phil. Mag. 4 (1852) 138–142.
[72] C.-L. Fu, K.-M. Ho, Phys. Rev. B 28 (1983) 5480–5486.
[73] P. Pulay, Chem. Phys. Lett. 73 (2) (1980) 393–398.
[74] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, E. Aprà, Int. J. High Perform. Comput. Appl. 20 (2) (2006) 203–231.
[75] The ELSI team, 2017, ELSI Users' Guide, http://elsi-interchange.org.
[76] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865–3868.
[77] V. Havu, V. Blum, P. Havu, M. Scheffler, J. Comput. Phys. 228 (22) (2009) 8367–8379.
[78] L. Nemec, V. Blum, P. Rinke, M. Scheffler, Phys. Rev. Lett. 111 (6) (2013) 065502.
[79] X.S. Li, J.W. Demmel, ACM Trans. Math. Software 29 (2) (2003) 110–140.
[80] L. Grigori, J.W. Demmel, X.S. Li, SIAM J. Sci. Comput. 29 (3) (2007) 1289–1314.
[81] M. Jacquelin, Y. Zheng, E. Ng, K. Yelick, An asynchronous task-based fan-both sparse Cholesky solver, arXiv:1608.00044, 2016.
[82] J. Lu, H. Yang, Multiscale Model. Simul. 15 (1) (2017) 254–273.
[83] H. Yang, J. Lu, PSPBLAS: A new framework for distributed memory sparse BLAS, 2017, in preparation.