Contents lists available at ScienceDirect

Computer Physics Communications

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

ELSI — An open infrastructure for electronic structure solvers[☆],[☆☆]

Victor Wen-zhe Yu[a], Carmen Campos[b], William Dawson[c], Alberto García[d], Ville Havu[e], Ben Hourahine[f], William P. Huhn[a], Mathias Jacquelin[g], Weile Jia[g,h], Murat Keçeli[i], Raul Laasner[a], Yingzhou Li[j], Lin Lin[g,h], Jianfeng Lu[j,k,l], Jonathan Moussa[m], Jose E. Roman[b], Álvaro Vázquez-Mayagoitia[i], Chao Yang[g], Volker Blum[a,l,*]

[a] Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA
[b] Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València, València, Spain
[c] RIKEN Center for Computational Science, Kobe 650-0047, Japan
[d] Institut de Ciència de Materials de Barcelona (ICMAB-CSIC), Bellaterra E-08193, Spain
[e] Department of Applied Physics, Aalto University, Aalto FI 00076, Finland
[f] SUPA, University of Strathclyde, Glasgow G4 0NG, UK
[g] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[h] Department of Mathematics, University of California, Berkeley, CA 94720, USA
[i] Computational Science Division, Argonne National Laboratory, Argonne, IL 60439, USA
[j] Department of Mathematics, Duke University, Durham, NC 27708, USA
[k] Department of Physics, Duke University, Durham, NC 27708, USA
[l] Department of Chemistry, Duke University, Durham, NC 27708, USA
[m] Molecular Sciences Software Institute, Blacksburg, VA 24060, USA

ARTICLE INFO

Article history:
Received 1 January 2020
Received in revised form 28 May 2020
Accepted 8 June 2020
Available online 24 June 2020

Keywords:
Electronic structure theory
Density-functional theory
Density-functional tight-binding
Parallel computing
Eigensolver
Density matrix

ABSTRACT

Routine applications of electronic structure theory to molecules and periodic systems need to compute the electron density from given Hamiltonian and, in case of non-orthogonal basis sets, overlap matrices. System sizes can range from few to thousands or, in some examples, millions of atoms. Different discretization schemes (basis sets) and different system geometries (finite non-periodic vs. infinite periodic boundary conditions) yield matrices with different structures. The ELectronic Structure Infrastructure (ELSI) project provides an open-source software interface to facilitate the implementation and optimal use of high-performance solver libraries covering cubic scaling eigensolvers, linear scaling density-matrix-based algorithms, and other reduced scaling methods in between. In this paper, we present recent improvements and developments inside ELSI, mainly covering (1) new solvers connected to the interface, (2) matrix layout and communication adapted for parallel calculations of periodic and/or spin-polarized systems, (3) routines for density matrix extrapolation in geometry optimization and molecular dynamics calculations, and (4) general utilities such as parallel matrix I/O and JSON output. The ELSI interface has been integrated into four electronic structure code projects (DFTB+, DGDFT, FHI-aims, SIESTA), allowing us to rigorously benchmark the performance of the solvers on an equal footing. Based on results of a systematic set of large-scale benchmarks performed with Kohn–Sham density-functional theory and density-functional tight-binding theory, we identify factors that strongly affect the efficiency of the solvers, and propose a decision layer that assists with the solver selection process. Finally, we describe a reverse communication interface encoding matrix-free iterative solver strategies that are amenable, e.g., for use with planewave basis sets.

Program summary
Program title: ELSI Interface
CPC Library link to program files: http://dx.doi.org/10.17632/473mbbznrs.1
Licensing provisions: BSD 3-clause
Programming language: Fortran 2003, with interface to C/C++

[☆] The review of this paper was arranged by Stephan Fritzsche.
[☆☆] 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 at: Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA.
E-mail address: volker.blum@duke.edu (V. Blum).

https://doi.org/10.1016/j.cpc.2020.107459

External routines/libraries: BLACS, BLAS, BSEPACK (optional), EigenExa (optional), ELPA, FortJSON, LAPACK, libOMM, MPI, MAGMA (optional), MUMPS (optional), NTPoly, ParMETIS (optional), PETSc (optional), PEXSI, PT-SCOTCH (optional), ScaLAPACK, SLEPc (optional), SuperLU_DIST
Nature of problem: Solving the electronic structure from given Hamiltonian and overlap matrices in electronic structure calculations.
Solution method: ELSI provides a unified software interface to facilitate the use of various electronic structure solvers including cubic scaling dense eigensolvers, linear scaling density matrix methods, and other approaches.

1. Introduction

Computer simulations based on electronic structure theory, particularly Kohn–Sham density-functional theory (KS-DFT) [1,2], are facilitating scientific discoveries across a broad range of disciplines such as chemistry, physics, and materials science. In KS-DFT, 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 $\hat{h}^{\mathrm{KS}}$ denotes the Kohn–Sham Hamiltonian, and ψ_l and ϵ_l are the Kohn–Sham orbitals and their associated eigenenergies. In most computer realizations of KS-DFT, Eq. (1) is discretized by expanding ψ_l with N_{basis} basis functions ϕ_j:

$$\psi_l(\boldsymbol{r}) = \sum_{j=1}^{N_{\mathrm{basis}}} c_{jl}\phi_j(\boldsymbol{r}), \tag{2}$$

which converts Eq. (1) into a generalized eigenproblem in a matrix form

$$\boldsymbol{HC} = \boldsymbol{SC\epsilon}. \tag{3}$$

Here $\boldsymbol{H}$ and $\boldsymbol{S}$ are the Hamiltonian matrix and the overlap matrix, respectively. $\boldsymbol{\epsilon}$ and $\boldsymbol{C}$ contain eigenvalues and eigenvectors of the eigensystem of $\boldsymbol{H}$ and $\boldsymbol{S}$. Solving Eq. (3) by diagonalization yields the Kohn–Sham orbitals (through Eq. (2)) and their eigenenergies, from which the electron density can be computed. Despite its success in a variety of traditional KS-DFT implementations, this approach has a computational cost that scales as O(N^3), with N being some measure of the system size. When handling complex systems consisting of many thousands of atoms, it often becomes prohibitively expensive even on today's most powerful supercomputers.

An alternative avenue to the electron density is available through the density matrix $\boldsymbol{P}$:

$$p_{ij} = \sum_{l=1}^{N_{\mathrm{basis}}} f_l c_{il} c_{jl}^*, \tag{4}$$

where f_l is the occupation number of the lth orbital, c_{il} and c_{jl} are elements of the eigenvector matrix $\boldsymbol{C}$ in Eq. (3), corresponding to coefficients of the ith and jth basis functions for the lth orbital, respectively, in Eq. (2). The density matrix $\boldsymbol{P}$ may be computed directly from $\boldsymbol{H}$ and $\boldsymbol{S}$ without explicitly solving the eigenproblem in Eq. (3). Since the early 1990s, density-matrix-based linear scaling algorithms [3–5], particularly for spatially localized basis functions, have been developed to overcome the scaling bottleneck of diagonalization. Even millions of atoms can be simulated with advanced implementations of linear scaling KS-DFT [6,7]. However, the larger computational prefactor associated with algorithms that target the density matrix without the diagonalization in Eq. (3) hinders their application in anything but very large systems. The diagonalization-based approach, in contrast, is generally applicable and highly efficient for systems comprised of up to several hundred atoms and can remain competitive up to several thousand atoms (see Section 3).

In the last decade, a number of new algorithms targeting the Kohn–Sham eigenproblem have emerged as software libraries, such as PEXSI (pole expansion and selected inversion) [8–11], CheSS (Fermi operator expansion by Chebyshev polynomials) [12], and iterative eigensolvers powered by spectrum slicing [13–16] or Chebyshev filtering [17–19]. Each of these algorithms has quite unique features, performance characteristics, and expert regimes. The crossover point between direct diagonalization and these alternative methods depends on the specifics of the simulation. It is thus a difficult task to select the optimal numerical method for a given system under study.

The ELectronic Structure Infrastructure (ELSI) project [20] provides an open-source, integrated software interface connecting electronic structure code packages to various high-performance eigensolvers and density matrix solvers. Encouragingly, the ELSI interface has attracted great interest from the community. To date, it supports five eigensolvers (ELPA [21–23], SLEPc-SIPs [13,14,24,25], EigenExa [26], LAPACK [27], MAGMA [28,29]), three density matrix solvers (libOMM [30], PEXSI [8–11], NTPoly [31]), and a special purpose eigensolver targeting the Bethe–Salpeter equation (BSEPACK [32]). ELSI has been integrated with four electronic structure packages (DFTB+ [33], DGDFT [34], FHI-aims [35], SIESTA [36]). It is open for adoption and modification by any other electronic structure code. In addition, ELSI is included in the Electronic Structure Library (ESL) project [37], a distribution of shared open-source libraries in the electronic structure community. In this paper, we present an update of the ELSI software from its 1.0.0 release (May 2017) [20] to the current 2.5.0 release (February 2020), covering newly added solvers, support of periodic and spin-polarized calculations, density matrix extrapolation routines for geometry optimization and molecular dynamics (MD) calculations, parallel matrix I/O, and a native Fortran library for JSON output. The ELSI interface and its integration into existing electronic structure codes enable us to rigorously benchmark the performance of multiple solvers on an equal footing. We report a systematic set of large-scale benchmarks performed with Kohn–Sham density-functional theory and density-functional tight-binding theory. Factors that strongly affect the efficiency of the solvers are identified and analyzed, based on which we propose a "decision layer" that assists users in selecting an appropriate solver for an arbitrary problem. Finally, we outline a reverse communication interface (RCI) encoding matrix-free iterative solver strategies, currently including the Davidson method [38,39], the orbital minimization method (OMM) [30,40], the projected preconditioned conjugate gradient (PPCG) method [41], and the Chebyshev filtering method [17,42]. This feature would benefit especially planewave-based DFT, but also other applications where only a few eigenvectors of a high-dimensional matrix are needed.

2. Upgraded and new features in ELSI (from 2017 to 2019)

In this section, we first briefly review the basic idea behind the ELSI software interface and its design, fundamentals of which were described in Ref. [20]. We then elaborate on new features added to ELSI since our previous publication, including support for new solvers and new matrix formats, API extension for managing multiple eigenproblems simultaneously across a given number of MPI tasks, new functions needed for routine electronic structure simulations such as the extrapolation of normalized density matrices between different underlying system geometries, and general utilities such as parallel matrix I/O and a native Fortran library for JSON output. An RCI framework for iterative eigensolvers will be discussed in Section 4.2 and will be published in more detail separately. A new CMake-based build system for ELSI will additionally be discussed in the Appendix.

2.1. Review of ELSI API

The ELSI infrastructure is intended for the rapid integration of a variety of eigensolvers and density matrix solvers into existing electronic structure codes. The API of ELSI is designed to be compatible with the workflow of an electronic structure code. There are three key steps to use ELSI in a code implementing the self-consistent field (SCF) method: (1) At the beginning of an SCF cycle, the electronic structure code initializes ELSI by calling the subroutine **elsi_init**, which returns an "ELSI handle" storing all runtime parameters of the ELSI interface and the solvers. "ELSI handle" is a derived data type defined in Fortran. C/C++ code can also access it via the iso_c_binding module of Fortran 2003. (2) Within the SCF cycle, the electronic structure code uses one of the ELSI solver interfaces to solve the Kohn–Sham eigenproblem by an eigensolver, or to compute the density matrix by a density matrix solver. This is done by calling **elsi_{ev|dm}_{real|complex}{_sparse}** for real or complex, dense or sparse matrices. (3) After the SCF cycle converges, the electronic structure code finalizes ELSI by calling **elsi_finalize**. At any point after the initialization of an ELSI handle and before its finalization, the electronic structure code can tune the parameters of the interface and the solvers. A detailed explanation of the ELSI workflow is available in Ref. [20] (see Fig. 3 and Algorithm 1 of that reference). Algorithms 1 and 2 in Section 2.7 also give a brief overview.

2.2. Solver updates

As of its 2.5.0 release, ELSI supports shared-memory eigensolvers LAPACK [27] and MAGMA [28,29], distributed-memory eigensolvers ELPA [21–23], SLEPc-SIPs [13,14,24,25], and EigenExa [26], distributed-memory density matrix solvers PEXSI [8–11], libOMM [30], and NTPoly [31], and the distributed -memory Bethe–Salpeter equation solver BSEPACK [32]. In the following sections, we cover a few algorithmic and technical aspects of the upgraded PEXSI solver and the newly added NTPoly and SLEPc-SIPs solvers. The reader is referred to the publications cited above for more details of the solvers supported in ELSI.

2.2.1. PEXSI (upgraded)

The pole expansion and selected inversion (PEXSI) algorithm [8,9] belongs to the category of Fermi operator expansion (FOE) methods. PEXSI expands the density matrix $\boldsymbol{P}$ as a sum of rational matrix functions:

$$\boldsymbol{P} = \sum_l \mathbb{Im}\left(\frac{\omega_l}{\boldsymbol{H} - (z_l + \mu)\boldsymbol{S}}\right), \tag{5}$$

where μ is the chemical potential of the system, $\{z_l\}$ and $\{\omega_l\}$ are complex shifts ("poles") and weights of the expansion terms. Here we assume matrices $\boldsymbol{H}$ and $\boldsymbol{S}$ to be real symmetric for simplicity. The formulation becomes slightly but not fundamentally different when $\boldsymbol{H}$ and $\boldsymbol{S}$ are complex Hermitian matrices.

For spatially localized basis functions and KS-DFT, only a subset of the elements of $\boldsymbol{H}$ and $\boldsymbol{S}$ (the set for which two basis functions overlap) will be non-zero and thus only a subset of the elements of the object $(\boldsymbol{H} - (z_l + \mu)\boldsymbol{S})^{-1}$ in Eq. (5), i.e. those corresponding to non-zero elements of $\boldsymbol{H}$ and $\boldsymbol{S}$, need to be computed for PEXSI. This is done using the parallel selected inversion method [43,44]. The computational complexity of Eq. (5) depends on the dimensionality of the system: $O(N)$, $O(N^{1.5})$, and $O(N^2)$ for 1D, 2D, and 3D systems, respectively, with N being the size of the system. This favorable scaling relies on the sparsity of the matrices which may be achieved with localized basis functions, but does not rely on the existence of an energy gap.

With PEXSI version 0.10, which was used in ELSI 1.0.0 [20], 40 to 100 poles are sufficient for the result obtained from PEXSI to be fully comparable (within 10^{-5} eV/atom [20]) to that obtained from diagonalization. The chemical potential is obtained by a Newton type method that may need several iterations to converge at the beginning of an SCF cycle. Eq. (5) is evaluated once in each of the iterations. As the SCF cycle proceeds and the electron density stabilizes, the quasi-Newton search can usually converge in one iteration.

The recently released PEXSI version 1.2 incorporates significant improvements over previous versions. First, the minimax rational approximation of the Fermi–Dirac distribution [45] becomes the default pole expansion method, reducing the number of terms needed in Eq. (5) to around $10 \sim 30$. Second, the Newton type method for finding the chemical potential is replaced by an algorithm, summarized below, that simultaneously improves the efficiency and robustness [11]. Instead of computing the exact chemical potential μ in every SCF iteration, the new algorithm tracks the upper and lower bounds of μ, denoted as $\mu_{\max}$ and $\mu_{\min}$. Then, an interpolation is carried out to estimate μ from $\mu_{\max}$ and $\mu_{\min}$. As the SCF cycle converges, $\mu_{\max}$ and $\mu_{\min}$ are guaranteed to converge from both sides towards the final, exact μ. Owing to the reduced number of poles and the elimination of the Newton iterations, PEXSI 1.2 shows a significant speed-up over previous versions.

2.2.2. NTPoly (new)

Density matrix purification is an established way to achieve linear scaling in electronic structure theory [4,5]. Assuming an orthogonal basis set, the density matrix $\boldsymbol{P}$ at zero temperature is known to satisfy three conditions:

$$\begin{aligned} &\text{Hermitian}: \boldsymbol{P} = \boldsymbol{P}^*, \\ &\text{Normalized}: \mathrm{Tr}(\boldsymbol{P}) = N_{\text{electron}}, \\ &\text{Idempotent}: \boldsymbol{P} = \boldsymbol{P}^2. \end{aligned} \tag{6}$$

Since an eigenproblem in a non-orthogonal basis can be transformed to an orthogonal basis by a decomposition of the overlap matrix $\boldsymbol{S}$, e.g. the Löwdin decomposition:

$$\begin{aligned} &\tilde{\boldsymbol{H}}\boldsymbol{C} = \boldsymbol{C}\boldsymbol{\epsilon}, \\ &\tilde{\boldsymbol{H}} = \boldsymbol{S}^{-1/2}\boldsymbol{H}\boldsymbol{S}^{-1/2}, \end{aligned} \tag{7}$$

we will stick to the assumption of an orthogonal basis set throughout this subsection.

Starting from an initial guess of the density matrix $\boldsymbol{P}_0$, density matrix purification methods iteratively update the density matrix by applying

$$\boldsymbol{P}_{n+1} = f(\boldsymbol{P}_n), \tag{8}$$

where $\boldsymbol{P}_{\text{n}}$ is the density matrix in the nth purification iteration, $\boldsymbol{P}_{\text{n+1}}$ is the density matrix in the (n+1)th iteration, and $f(\boldsymbol{P})$ is usually a matrix polynomial, chosen to guarantee that $\boldsymbol{P}_{\text{n}}$ rapidly and stably converges to a matrix satisfying Eq. (6).

For a matrix to satisfy the idempotent condition in Eq. (6), its eigenvalues can only be 0 and 1. The initial guess $\boldsymbol{P}_0$ that enters Eq. (8) is usually obtained by scaling the Hamiltonian matrix to make its eigenvalues lie in between 0 and 1 [46]:

$$\boldsymbol{P}_0 = \frac{\epsilon_{\text{max}}\boldsymbol{I} - \boldsymbol{H}}{\epsilon_{\text{max}} - \epsilon_{\text{min}}}, \tag{9}$$

where ϵ_{max} and ϵ_{min} are the (estimated) maximum and minimum eigenvalues of $\boldsymbol{H}$. Then, a number of choices for $f(\boldsymbol{P})$ are able to drive all eigenvalues towards 0 or 1. We refer the reader to Refs. [4,5] and references therein for a review of density matrix purification algorithms.

For systems with a significant energy gap, the magnitude of the density matrix elements p_{ij} in Eq. (4) decreases exponentially with respect to the distance between the ith and jth basis functions, $|\boldsymbol{r}_i - \boldsymbol{r}_j|$ [3]. A cutoff distance r_{cutoff} is often used with linear scaling methods to truncate the density matrix, such that when $|\boldsymbol{r}_i - \boldsymbol{r}_j| > r_{\text{cutoff}}$, the corresponding matrix element p_{ij} is forced to be zero. This truncation leads to a highly sparse density matrix for large systems. Alternatively, the sparsity of the density matrix may be enforced by dynamically filtering out matrix elements smaller than a predefined threshold. Given sufficiently large systems with a proper gap, the density matrix will be highly sparse, and the computational complexity of density matrix purification is O(N).

Exploiting its sparse, massively parallel matrix–matrix multiplication kernel, the NTPoly library [31] implements the canonical purification algorithm [47] that preserves the number of electrons throughout the purification iterations, the trace resetting purification methods [46] that successively refine the number of electrons to the target value, and the generalized canonical purification method [48] that relies on the relationship between the electron density matrix and the hole density matrix to accelerate convergence. A comparison of these methods is given in Ref. [31]. In NTPoly, the sparsity of $\{\boldsymbol{P}_{\text{n}}\}$ is ensured by setting any matrix element smaller than a predefined threshold to zero. This approach allows for an easier integration with electronic structure codes, as it does not require any knowledge on the physical system or the basis functions.

2.2.3. SLEPc-SIPs (new)

The shift-and-invert spectral transformation method, implemented in the SLEPc library [13,25], transforms the eigenproblem in Eq. (3) by shifting the eigenspectrum:

$$(\boldsymbol{H} - \sigma\boldsymbol{S})\boldsymbol{C} = \boldsymbol{S}\boldsymbol{C}(\boldsymbol{\epsilon} - \sigma\boldsymbol{I}), \tag{10}$$

where σ is a shift and $\boldsymbol{I}$ is the identity matrix. This shifted eigenproblem is converted to the standard form by inverting $(\boldsymbol{H} - \sigma\boldsymbol{S})$ and $(\boldsymbol{\epsilon} - \sigma\boldsymbol{I})$:

$$(\boldsymbol{H} - \sigma\boldsymbol{S})^{-1}\boldsymbol{S}\boldsymbol{C} = \boldsymbol{C}(\boldsymbol{\epsilon} - \sigma\boldsymbol{I})^{-1}, \tag{11}$$

which is a standard eigenproblem

$$\begin{aligned} \tilde{\boldsymbol{H}}\boldsymbol{C} &= \boldsymbol{C}\tilde{\boldsymbol{\epsilon}}, \\ \tilde{\boldsymbol{H}} &= (\boldsymbol{H} - \sigma\boldsymbol{S})^{-1}\boldsymbol{S}, \\ \tilde{\boldsymbol{\epsilon}} &= (\boldsymbol{\epsilon} - \sigma\boldsymbol{I})^{-1}. \end{aligned} \tag{12}$$

The transformed standard eigenproblem in Eq. (12) is solved by an iterative Krylov–Schur algorithm in SLEPc [13]. If a shift can be chosen to be close to the target eigenvalue, the shift-and-invert transformation augments the magnitude of the eigenvalue, accelerating the solution of the standard eigenproblem in Eq. (12).

The shift-and-invert transformation becomes less effective when many eigenvalues need to be computed, because not all of them are close to the shift. The spectrum slicing technique is an advanced modification that employs multiple shifts to compute all eigenvalues contained in an interval. A large interval can be partitioned into independent slices and solved in parallel. The parallel spectrum slicing method implemented in SLEPc-SIPs [13,14] partitions the eigenspectrum of an eigenproblem into N_{slice} slices. Correspondingly, the computer processes involved in the calculation are split into N_{slice} groups, so that each slice is handled by one group. Thanks to this additional layer of parallelism, this approach has the potential to exhibit enhanced scalability over the direct diagonalization method, which has been demonstrated in calculations on the density-functional-based tight-binding (DFTB) level [14] as well as on the KS-DFT level [24]. Additionally, SLEPc-SIPs should greatly outperform direct diagonalization methods in cases where only a small fraction of the eigenspectrum is wanted.

2.3. Matrix format updates

In software implementations of electronic structure theory, matrices can be stored in any form. ELSI, as a unified solver interface, must be able to handle matrices stored in commonly used formats, and must be able to convert between these formats. Two matrix formats were implemented in the previous version of ELSI, namely the 2D block-cyclic distributed dense format (BLACS_DENSE) and the 1D block distributed compressed sparse column (CSC) sparse format (PEXSI_CSC) [20]. They have enabled the use of the ELPA, libOMM, and PEXSI solvers in the electronic structure codes FHI-aims and DGDFT. In the present version of ELSI, two additional matrix formats are available, namely the 1D block-cyclic distributed CSC sparse format (SIESTA_CSC) and the generic coordinate sparse format (GENERIC_COO). The 1D and 2D block-cyclic distribution schemes BLACS_DENSE and SIESTA_CSC essentially cover all variants of block/cyclic/block-cyclic distributions, as pure cyclic and pure block distributions can be viewed as special cases of the general block-cyclic distribution. The GENERIC_COO format is designed to support any data distribution scheme that might be employed to store matrices in an electronic structure code. On each process, a list of triplets is constructed, containing local matrix elements and their global row and column indices. Based on the indices, ELSI then redistributes the matrix to the format needed by the chosen solver. The triplet list can be arbitrarily distributed, sorted or unsorted.

Conversions between any two of the supported matrix formats are implemented with MPI. In order to avoid unnecessary data communication, all zeros in a matrix are always ignored in the redistribution process. As reported in Ref. [20], time spent on the conversion is negligible relative to the actual computation time.

2.4. Interface extension for spin-polarized and periodic systems

The parallelization strategy behind the ELSI interface depends on the physical system being simulated. The base case is an isolated system in vacuum, e.g. free atoms, molecules, clusters, without spin-polarization. In this case, there is one eigenproblem (Eq. (3)) in each iteration of an SCF cycle. The strategy to tackle this single eigenproblem has been detailed in Ref. [20]. We here introduce how spin-polarized and periodic systems are supported in ELSI.

When a spin-polarized periodic system is considered, Eq. (3) will have an index α denoting the spin channel, and an index $\boldsymbol{k}$ denoting points in reciprocal space:

$$\boldsymbol{H}_{\boldsymbol{k}}^{\alpha}\boldsymbol{C}_{\boldsymbol{k}}^{\alpha} = \boldsymbol{S}_{\boldsymbol{k}}\boldsymbol{C}_{\boldsymbol{k}}^{\alpha}\boldsymbol{\epsilon}_{\boldsymbol{k}}^{\alpha}. \tag{13}$$

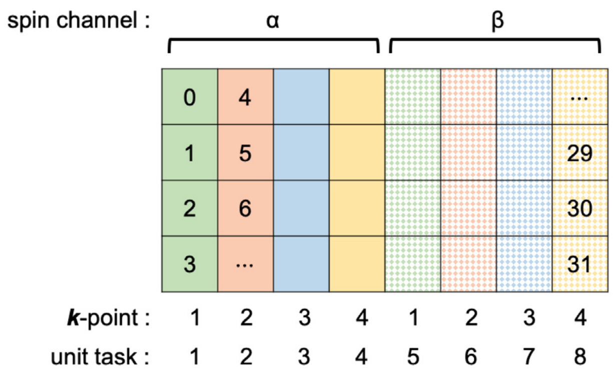
spin channel :
α
β
0
1
2
3
4
5
6
...
...
29
30
31
$\boldsymbol{k}$-point : 1 2 3 4 1 2 3 4
unit task : 1 2 3 4 5 6 7 8

Fig. 1. Parallel calculation of spin-polarized and periodic systems in ELSI exemplified with a fictitious system of two spin channels and four $\boldsymbol{k}$-points, solved by 32 parallel processes. Each square box represents one process, with its index (0, 1, …, 31) labeled inside. The 32 processes are divided into eight process groups, with four processes in each. Spin channel α and β are indicated by boxes with solid and chessboard background, respectively. $\boldsymbol{k}$-points 1, 2, 3, and 4 are indicated by green, red, blue, and yellow boxes, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Because of the periodicity, it is sufficient to study $\boldsymbol{k}$ within a single primitive unit cell in reciprocal space, usually the first Brillouin zone (1BZ). The physical quantities are represented by integrals in 1BZ. Take the electron density $n(\boldsymbol{r})$ as an example:

$$n(\boldsymbol{r}) = \sum_{\alpha=1}^{N_{\text{spin}}} \sum_{l=1}^{N_{\text{basis}}} \int_{\text{1BZ}} f_{l\boldsymbol{k}}^{\alpha} \psi_{l\boldsymbol{k}}^{\alpha *}(\boldsymbol{r}) \psi_{l\boldsymbol{k}}^{\alpha}(\boldsymbol{r}) d^3\boldsymbol{k}, \tag{14}$$

which is approximated by using a finite mesh of $\boldsymbol{k}$-points in 1BZ:

$$n(\boldsymbol{r}) \approx \sum_{\alpha=1}^{N_{\text{spin}}} \sum_{n=1}^{N_{\text{kpt}}} w_{\boldsymbol{k}_n} \sum_{l=1}^{N_{\text{basis}}} f_{l\boldsymbol{k}_n}^{\alpha} \psi_{l\boldsymbol{k}_n}^{\alpha *}(\boldsymbol{r}) \psi_{l\boldsymbol{k}_n}^{\alpha}(\boldsymbol{r}). \tag{15}$$

Here, $\psi_{l\boldsymbol{k}_n}^{\alpha}$ and $f_{l\boldsymbol{k}_n}^{\alpha}$ are the lth band at the nth $\boldsymbol{k}$-point in the α spin channel and its occupation number, $w_{\boldsymbol{k}_n}$ is the weight of the nth $\boldsymbol{k}$-point, N_{spin}, N_{kpt}, and N_{basis} are the number of spin channels, $\boldsymbol{k}$-points, and basis functions, respectively. The weights of all $\boldsymbol{k}$-points add up to 1. A denser grid of $\boldsymbol{k}$-points leads to a more accurate description of 1BZ, at the price of higher computational cost. If the unit cell in real space is small, an accurate description of the electronic structure requires thousands of $\boldsymbol{k}$-points or even more. If the unit cell in real space is large, the Brillouin zone may already be well-represented by the origin of the reciprocal space, known as the Γ point. The Hamiltonian and overlap matrices for multiple $\boldsymbol{k}$-points have a block-diagonal structure. Each block on the diagonal corresponds to an eigenproblem of one $\boldsymbol{k}$-point.

In total, there are $N_{\text{kpt}} \times N_{\text{spin}}$ eigenproblems in Eq. (13). They can be solved in an embarrassingly parallel fashion. In ELSI, eigenproblems in Eq. (13) are considered as equivalent "unit tasks", which are set up by the electronic structure code. The available computer processes are divided into $N_{\text{kpt}} \times N_{\text{spin}}$ groups, each of which is responsible for one unit task. An example with two spin channels (α and β) and four $\boldsymbol{k}$-points (1, 2, 3, 4) solved by 32 parallel processes (0, 1, …, 31) is given in Fig. 1. There are eight unit tasks in this example. Each unit task is handled by four processes in a "solve" phase followed by a "reduction" phase, which are further explained below. In an actual calculation, the total number of processes is not restricted to be a multiple of the number of eigenproblems $N_{\text{kpt}} \times N_{\text{spin}}$. Process groups can have different numbers of processes, although a uniform partition usually leads to the optimal load balance.

Computational steps carried out in the "solve" phase include:

- ELPA, SLEPc-SIPs, and EigenExa: Solve the eigenproblems in Eq. (13).
- PEXSI: Perform the pole expansion for each unit task at a chemical potential uniform across all tasks. Compute the number of electrons for each unit task.
- libOMM and NTPoly: Perform orbital minimization or density matrix purification for each unit task to obtain density matrices.

Computational steps carried out in the "reduction" phase include:

- ELPA, SLEPc-SIPs, and EigenExa: Eigensolutions for the unit tasks are coupled by the normalization condition of the number of electrons:

$$N_{\text{electron}} = \sum_{n=1}^{N_{\text{kpt}}} \sum_{\alpha=1}^{N_{\text{spin}}} \sum_{l=1}^{N_{\text{basis}}} w_{\boldsymbol{k}_n} f_{l\boldsymbol{k}_n}^{\alpha}, \tag{16}$$

  The eigenvalues solved for by each unit task are collected across all the tasks, for the determination of the chemical potential and occupation numbers.
- PEXSI: The total number of electrons is computed as a weighted summation over the number of electrons solved for all unit tasks.
- libOMM and NTPoly: None.

To use the ELSI interface in a spin-polarized and/or periodic calculation, two MPI communicators should be passed into ELSI, for data communication within a unit task in the "solve" phase and communication between all unit tasks in the "reduction" phase, respectively. We note that the parallelization strategy described in this subsection can only be applied when there are more processes than unit tasks. When the number of processes is small, the ELSI interface can be set up in such a way that each process invokes the eigensolver in LAPACK or MAGMA to solve a few unit tasks sequentially. This is referred to as the "SINGLE_PROC" parallelization mode in Ref. [20].

2.5. Calculation of the energy-weighted density matrix

In the context of geometry optimization or molecular dynamics (collectively referred to as "geometry calculations" hereafter), forces (derivatives of the total energy) are needed to evaluate the movement of atoms. There is a so-called "Pulay" force term that originates as localized basis functions move with the atoms [49]. One ingredient needed to compute the "Pulay" force is the energy-weighted density matrix $\boldsymbol{Q}$:

$$q_{ij} = \sum_{l=1}^{N_{\text{basis}}} \epsilon_l f_l c_{il} c_{jl}^*, \tag{17}$$

which is the density matrix $\boldsymbol{P}$ in Eq. (4) weighted by eigenvalues of the orbitals. Density matrix solvers compute $\boldsymbol{Q}$ directly from $\boldsymbol{H}$, $\boldsymbol{S}$, and $\boldsymbol{P}$. In ELSI, $\boldsymbol{Q}$ is not computed at the time when a density matrix solver is invoked through **elsi_dm_{real|complex}{_sparse}**. Since forces are typically only needed near or after the end of an SCF cycle, always computing $\boldsymbol{Q}$ together with $\boldsymbol{P}$ is unnecessary. Instead, **elsi_get_edm_{real|complex}{_sparse}** is dedicated to retrieving $\boldsymbol{Q}$ whenever it is needed.

2.6. Calculation of the electronic entropy

At zero temperature, the occupation number $\{f_{l\boldsymbol{k}}^{\alpha}\}$ of an orbital in a metallic system drops abruptly from 1 (omitting spin degeneracy) to 0 at the Fermi energy. This is undesired in two aspects. First, to accurately integrate this discontinuous function in, e.g., Eq. (15), a very fine grid of $\boldsymbol{k}$-points is required to sample the Brillouin zone. More $\boldsymbol{k}$-points lead to a more accurate description of the Brillouin zone, at the price of proportionally increasing computational cost. Second, when an orbital suddenly changes from unoccupied to occupied, or vice versa, its contribution to the electron density (Eq. (15)) suddenly appears or disappears, leading to instabilities in the convergence of an SCF cycle.

In practical calculations, a metallic system is usually treated at a higher electronic temperature by means of "smearing" of the density of states, which makes the occupation function decrease smoothly from 1 to 0 around the Fermi level. This continuous function can be integrated more accurately with a relatively coarse grid of $\boldsymbol{k}$-points. In addition, using a smearing function helps stabilize the SCF convergence for any system with a zero or small energy gap [50,51]. A side effect is that the energy functional being minimized no longer corresponds to the total energy E_{tot} at zero temperature, but the free energy $E_{\text{free}} = E_{\text{tot}} - TS$, where T and S are the electronic temperature and the electronic entropy, respectively [52,53]. Correspondingly, "forces" needed for geometry calculations should be computed as gradients of the free energy, instead of the regular total energy. In fact, in the analytical gradient of the free energy, the term involving the derivative of the fractional occupation numbers with respect to the atomic displacement exactly cancels with the corresponding term originating from the derivative of the electronic entropy [53].

In ELSI, we have implemented the Fermi–Dirac [54], Gaussian [55], Methfessel–Paxton [56], and Marzari–Vanderbilt [57] smearing functions. Given the exact number of electrons and the eigenenergies of the orbitals (computed by the ELPA, SLEPc-SIPs, EigenExa, LAPACK, or MAGMA eigensolver), the occupation numbers, chemical potential, and electronic entropy can be calculated with any one of the four smearing functions. The electronic entropy term is currently unavailable from ELSI when using a density matrix solver. The PEXSI solver is capable of directly computing the free energy density matrix [9], from which the free energy and Fermi–Dirac entropy can be deduced.

2.7. Reinitialization of ELSI

When the atomic positions get updated in geometry optimization or molecular dynamics calculations, localized basis functions move together with atoms, leading to a new overlap matrix $\boldsymbol{S}_1$, and a sparsity pattern different from that of the previous overlap matrix $\boldsymbol{S}_0$. In the previous version of ELSI, the user was responsible for finalizing ELSI and reinitializing it for an updated geometry, as shown in Algorithm 1. This guarantees that outdated information is not carried over to the updated geometry. However, this also discards information that can be reused, such as the MPI setup and most of the solver-specific settings. To maximize the reuse of information between geometry steps, and to minimize the coding effort on the electronic structure code side, we have introduced the **elsi_reinit** subroutine to reinitialize an instance of ELSI. The usage of **elsi_reinit** in geometry calculations is demonstrated in Algorithm 2. Compared to **ELSI_OLD**, the initialization and finalization of ELSI are moved out of the geometry loop, avoiding the repeated re-creation of ELSI instances. A new geometry step is indicated by calling **elsi_reinit**, which instructs ELSI to flush geometry-related variables and arrays that cannot be safely reused in the new geometry step, mainly the overlap matrix and its sparsity pattern. Other information is kept within the ELSI instance and reused throughout multiple geometry steps.

Algorithm 1 Usage of the previous version of the ELSI interface [20] in geometry optimization and molecular dynamics calculations. For simplicity, tasks belonging to the electronic structure code, e.g., the construction of electron density and the integration of Hamiltonian, are not shown.

```
procedure ELSI_OLD
while (geometry not converged) do
    call elsi_init
    call elsi_set_*
    while (SCF not converged) do
        call elsi_{ev|dm}
    end while
    call elsi_finalize
end while
```

Algorithm 2 Usage of the present version of the ELSI interface in geometry optimization and molecular dynamics calculations. Compared to the **ELSI_OLD** procedure in 1, repeated re-creation of ELSI instances is avoided by using **elsi_reinit**.

```
procedure ELSI_NEW
call elsi_init
call elsi_set_*
while (geometry not converged) do
    while (SCF not converged) do
        call elsi_{ev|dm}
    end while
    call elsi_reinit
end while
call elsi_finalize
```

2.8. Extrapolation of wavefunctions and the density matrix

In a single point total energy calculation, a simple way to construct an initial guess for the electron density is to use a superposition of free atom densities. In geometry calculations, the initial guess in the (n+1)th geometry step can be improved by reusing the wavefunctions or density matrix calculated in the nth geometry step. However, due to the movement of atoms and localized basis functions around them, wavefunctions obtained in the nth geometry step are no longer orthonormalized in the (n+1)th geometry step. Similarly, the density matrix from the nth geometry step $\boldsymbol{P}_0$ is no longer normalized with respect to the new overlap matrix $\boldsymbol{S}_1$, i.e., $\text{Tr}(\boldsymbol{P}_0\boldsymbol{S}_1) \neq N_{\text{electron}}$ and $\boldsymbol{P}_0 \neq \boldsymbol{P}_0\boldsymbol{S}_1\boldsymbol{P}_0$.

The wavefunction coefficients, i.e., eigenvectors in Eq. (3), can be reorthonormalized with respect to $\boldsymbol{S}_1$. This is implemented in ELSI with the Gram–Schmidt algorithm. For the density matrix, we decompose the previous and current overlap matrices $\boldsymbol{S}_0$ and $\boldsymbol{S}_1$, then extrapolate the previous density matrix $\boldsymbol{P}_0$ to a new density matrix $\boldsymbol{P}_1$ [58–60]. With a Cholesky decomposition of the overlap matrices, the process is:

$$\begin{aligned}
&\boldsymbol{S}_0 = \boldsymbol{L}_0\boldsymbol{L}_0^*,\\
&\boldsymbol{S}_1 = \boldsymbol{L}_1\boldsymbol{L}_1^*,\\
&\boldsymbol{P}_1 = (\boldsymbol{L}_1^{-1})^*\boldsymbol{L}_0^*\boldsymbol{P}_0\boldsymbol{L}_0\boldsymbol{L}_1^{-1},
\end{aligned} \tag{18}$$

An equivalent extrapolation formula exists when using the Löwdin decomposition. The common idea is transforming $\boldsymbol{P}_0$ to an orthogonal basis, then to the new non-orthogonal basis. Both algorithms are implemented for real and complex, dense and sparse matrices.

2.9. Parallel matrix I/O

Matrices from a given electronic structure calculation are often reusable in further computational steps – e.g., in the simplest case, to restart a particular calculation that could not be completed within a single run, or that needed to be revisited for future data extraction tasks. In such cases, it is helpful to write some information (e.g. the current atomic structure and density matrix) to file, which serves as a checkpoint to restart a calculation. When ELSI runs in parallel with multiple MPI tasks, matrices are distributed across tasks. The idea of writing a distributed matrix into N_{MPI} separate files, where N_{MPI} is the number of MPI tasks, is not promising due to the difficulty of post-processing a large number of files. Therefore, parallel matrix I/O (input/output) routines are preferred. Other use cases of this functionality include, e.g., sharing matrices between collaborating teams and testing the solvers with pre-generated matrices.

In the past, we have encountered some difficulties in using existing parallel I/O libraries [61,62] with thousands of MPI tasks on various supercomputers. This might be connected to overhead introduced by complex hierarchical data structures implemented in high level I/O libraries. Therefore, the data structure in ELSI is simply arrays that represent matrices, if necessary, accompanied by a few integers to define the dimensions of the matrices. It is more straightforward to directly employ the parallel I/O functionality defined in the MPI standard [63]. Our implementation of matrix I/O in ELSI is built around MPI_File_{read|write}_at_all, which allows distributed data to be written to (read from) a single file, as if the data was gathered onto a single MPI task then written to one file (read from one file by one MPI task then scattered to all). The optimal I/O performance, both with MPI I/O and in general, is obtained by making large and contiguous requests to access the file system, rather than small, non-contiguous, or random requests. Therefore, before writing a matrix to file, we redistribute it to a 1D block distribution. This guarantees that each task writes a contiguous chunk of data to a contiguous piece of file. Similarly, a matrix read from file is in a 1D block distribution. Whenever needed, it can be redistributed automatically to one of the supported distributions.

2.10. JSON output via the FortJSON library

The workflow of benchmarking multiple solvers across different electronic structure codes is greatly accelerated by using a consistent output format that is easily processed by external scripting utilities. To aid in accelerating this workflow when using ELSI, we provide JSON output from ELSI using FortJSON, an open source JSON library written by ELSI developers for Fortran 2003 code bases. All benchmarks presented in Section 3 were output using FortJSON.

FortJSON is bundled as part of the ELSI package and is available to codes linked against ELSI. Alternatively, it may be downloaded as a standalone package from the ELSI GitLab server and installed using a CMake build system. It uses a handle-based structure similar to ELSI's, in which the user initializes a FortJSON handle to open a JSON file, calls a consistent API to write to the JSON file, and finalizes the handle to finish output. JSON files written by FortJSON are fully compliant with the "ECMA-404 The JSON Data Interchange" standard [64]. FortJSON is, in principle, a standalone library that can be used to facilitate JSON output in any other context as well.

2.11. Summary of new features in ELSI

As a summary of this section, all major changes of ELSI since Ref. [20] are listed below.

- PEXSI v1.2: Default number of poles reduced to 20 without sacrificing accuracy; efficient and robust strategy for finding the chemical potential. (Section 2.2.1)
- NTPoly: Linear scaling density matrix purification methods (canonical purification, 2nd and 4th order trace resetting purification, generalized hole-particle canonical purification) in the NTPoly library. (Section 2.2.2)
- SLEPc-SIPs: Shift-and-invert parallel spectrum slicing eigensolver in the SLEPc library. (Section 2.2.3)
- EigenExa: Tridiagonalization and penta-diagonalization eigensolvers in the EigenExa library.
- MAGMA: GPU-accelerated one-stage and two-stage tridiagonalization eigensolvers in the MAGMA library.
- BSEPACK: Distributed-memory eigensolver in the BSEPACK library, specifically targeting the Bethe–Salpeter equation.
- 1D block-cyclic distributed compressed sparse column matrix format (SIESTA_CSC). (Section 2.3)
- Arbitrarily distributed coordinate sparse matrix format (GENERIC_COO). (Section 2.3)
- Interface extension for spin-polarized and periodic systems. (Section 2.4)
- Energy-weighted density matrix available for all supported solvers and matrix formats. (Section 2.5)
- Electronic entropy with Fermi–Dirac, Gaussian, Methfessel–Paxton, and Marzari–Vanderbilt broadening schemes. (Section 2.6)
- Reinitialization of ELSI between geometry steps. (Section 2.7)
- Density matrix extrapolation between geometry steps. (Section 2.8)
- Gram–Schmidt orthogonalization of eigenvectors between geometry steps. (Section 2.8)
- MPI I/O based parallel matrix I/O for dense and sparse matrices. (Section 2.9)
- JSON output via the FortJSON library. (Section 2.10)
- Iterative eigensolvers in a reverse communication interface (RCI) framework. (Section 4.2)
- CMake build system. (Appendix)

3. Benchmarks and discussions

Since the previous version, we have successfully integrated the ELSI interface into two additional electronic structure projects, DFTB+ and SIESTA. It is now possible for the users of DFTB+, DGDFT, FHI-aims, and SIESTA to easily switch between eigensolvers and density matrix solvers supported in ELSI by specifying a single keyword in their package-specific input files. Depending on the needs and knowledge level of the users, ELSI can be used as a black box solution, or can be fine-tuned to allow for optimal performance in certain circumstances. Developments, enhancements, and fixes made in ELSI and the solvers are immediately available for users of these electronic structure codes. Knowledge gained from ELSI usage in one code often benefits users of other codes.

The ELSI interface and its integration with electronic structure codes enable us to rigorously benchmark the performance of different solvers on an equal footing. We present here a large set of cross-solver, cross-code benchmarks as an essential step to settle the respective efficiency of different solvers in different regimes. Testing Hamiltonian and overlap matrices are constructed from actual electronic structure calculations, at the levels of all-electron full-potential KS-DFT (with the FHI-aims code [35]), pseudopotential KS-DFT (with the SIESTA code [36]), and DFTB (with the DFTB+ code [33]).

We select eight benchmark atomic structure models, which include small and large models ranging from a hundred to tens of thousands of atoms, 1D, 2D, and 3D materials, light and heavy elements, and gapped and gapless systems. According to their composition and dimensionality, the structures are classified into

Table 1
List of solver-specific parameters, including the diagonalization algorithm and block size for ELPA, the pole expansion algorithm and number of poles for PEXSI, the density matrix purification algorithm, truncation parameter, and convergence criterion for NTPoly. The same settings are employed across all benchmarks reported in this paper.

Solver	Parameter	Value
ELPA	Algorithm	Two-stage diagonalization
ELPA	Block size	16
PEXSI	Algorithm	Minimax rational approximation
PEXSI	Number of poles	20
NTPoly	Algorithm	4th order trace-resetting
NTPoly	Truncation	10^{-5}
NTPoly	Convergence	10^{-2}

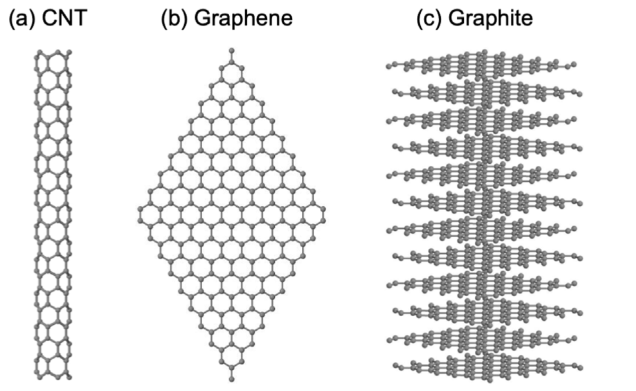

Fig. 2. Atomic structures of (a) 1D carbon nanotube (CNT), (b) 2D graphene, and (c) 3D graphite.

three sets: the carbon set (Section 3.1), the heavy set (Section 3.2), and the bulk set (Section 3.3). Our comparison of solver performance focuses on key computational steps of the solvers that are repeated in every SCF iteration. These repeated steps are: for ELPA, transforming the generalized eigenproblem in Eq. (3) to the standard form, solving the standard eigenproblem, back-transforming the eigenvectors, and constructing the density matrix via Eq. (4); for PEXSI, computing the density matrix via Eq. (5); for NTPoly, transforming the generalized eigenproblem in Eq. (3) to the standard form by Eq. (7), computing the density matrix via Eq. (8), and back-transforming the density matrix. There are other computationally expensive steps that are needed only once for a fixed geometry, such as the Cholesky or LU factorization of the overlap matrix. They have less significant effects on the total time of an SCF cycle, thus are not discussed here.

As explained in Section 2.4, there are multiple eigenproblems in a spin-polarized system and/or a periodic system with multiple $\boldsymbol{k}$-points. Since these eigenproblems can be solved fully in parallel, the total computational cost can be roughly predicted from the cost of one eigenproblem. Therefore, we stick to a spin-non-polarized, periodic setting with a $1 \times 1 \times 1$ $\boldsymbol{k}$-grid (i.e., Γ point only) in all our benchmarks. Table 1 summarizes solver-specific parameters used in our benchmarks. For ELPA, we use the two-stage diagonalization algorithm [22,65]. A block size of 16 is chosen for the block-cyclic matrix distribution, which usually leads to the optimal performance of the two-stage ELPA solver [22,66]. For PEXSI, we use 20 poles (z_l in Eq. (5)), derived from the minimax rational approximation [45]. For NTPoly, we use the 4th order trace-resetting purification method [46] with the truncation parameter (matrix elements below which are discarded in order to maintain the sparsity of the matrices) being 10^{-5} and the convergence criterion 10^{-2}. With these settings, the maximum difference in total energy for a given geometry is 0.5 μeV/atom between results obtained with ELPA and PEXSI, and 32.4 μeV/atom between results obtained with ELPA and NTPoly, indicating excellent accuracy of the solvers. The rest of this section will mainly focus on the performance of the solvers. Complete input files used in this section are available at the ELSI GitLab server [67].

3.1. Benchmark set I: Carbon allotropes

In the carbon benchmark set, we construct models of three carbon allotropes, namely (quasi) 1D carbon nanotube, 2D graphene, and 3D graphite, as shown in Fig. 2(a) (b) and (c), respectively. We run KS-DFT calculations with the PBE [68] semi-local exchange-correlation functional with two software packages, namely FHI-aims and SIESTA. Input geometries are made identical in FHI-aims and SIESTA calculations. Predefined "tier 1" and "DZP" (double-zeta plus polarization) basis sets are used in FHI-aims and SIESTA, respectively. Both basis sets contain the minimal sp functions and an additional set of spd functions. The number of basis functions per atom in SIESTA is one fewer than that in FHI-aims, because of the use of pseudopotentials in SIESTA. The dimensions of the models, the number of basis functions, and the sparsity factor of the corresponding matrices are summarized in Table 2.

Table 2
Dimension of structures in the carbon benchmark set, and sparsity of the Hamiltonian matrix in the calculation. $\tilde{n}_{\text{basis}}$ is the average number of basis functions per atom. N_{atom} is the number of atoms. The global dimension of the Hamiltonian, overlap etc. matrices is equal to the total number of basis functions $N_{\text{basis}} = \tilde{n}_{\text{basis}} \times N_{\text{atom}}$. Sparsity is defined as $N_{\text{zero}}/N_{\text{basis}}^2$, with N_{zero} being the number of zero matrix elements. A minimal + spd basis set is used to describe carbon atoms in FHI-aims and SIESTA.

Code	System	$\tilde{n}_{\text{basis}}$	N_{atom}	Sparsity (%)
FHI-aims	CNT	14	800–6400	91.3–98.9
FHI-aims	Graphene	14	800–7200	90.5–99.4
FHI-aims	Graphite	14	864–6912	75.1–96.6
SIESTA	CNT	13	800–6400	95.0–99.4
SIESTA	Graphene	13	800–7200	94.5–99.5
SIESTA	Graphite	13	864–6912	92.5–99.1

Calculations of the carbon set were performed on the Cray XC30 supercomputer Edison at National Energy Research Scientific Computing Center (NERSC). Each node of Edison was equipped with two 12-core Intel Ivy Bridge processors. 80 nodes were fully exploited by launching 24 MPI tasks on each node, yielding 1920 MPI tasks in total. No OpenMP parallelization was employed.

In Fig. 3, we compare the performance of the ELPA and PEXSI solvers in all-electron KS-DFT calculations with the FHI-aims code. Fig. 3(a) shows the wallclock time needed for ELPA to solve for 42.9% of the eigenspectrum, versus the time needed for PEXSI to compute the density matrix, in calculations of 1D carbon nanotube models consisting of 800 to 6400 atoms (11,200 to 89,600 basis functions). PEXSI consistently outperforms ELPA in these calculations. The benefit of using PEXSI becomes more significant as the size of the system increases, owing to the O(N) computational complexity of PEXSI for 1D systems. Fig. 3(b) shows the same comparison between ELPA and PEXSI for 2D graphene models consisting of 800 to 7200 atoms (11,200 to 100,800 basis functions). These examples are identical to those reported in Fig. 7 (a) of Ref. [20], except that newer versions of ELPA and PEXSI are used here. In both Fig. 3(b) and Ref. [20], there is a crossover point between ELPA and PEXSI. The improvements of PEXSI discussed in Section 2.2.1, especially the reduction in the number of poles [11,45], contribute to a speed-up of PEXSI by a factor of two, bringing down the crossover point from 3000

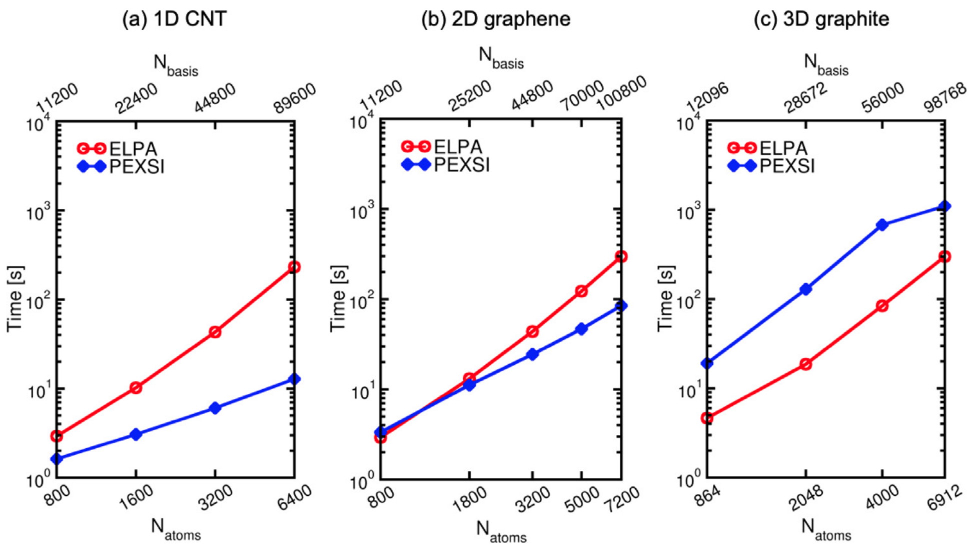

Fig. 3. Performance of key steps in ELPA and PEXSI for (a) carbon nanotube models, (b) graphene models, (c) graphite models, as a function of the number of atoms. For ELPA, the key steps include transforming the eigenproblem from the generalized form to the standard form, solving the standard eigenproblem, back-transforming the eigenvectors to the generalized form, and constructing the density matrix; for PEXSI, evaluating the pole expansion and selected inversion in Eq. (5) and assembling the density matrix. ELPA computes all eigenvalues and 42.9% of the eigenvectors. Results in this figure are obtained by running the FHI-aims code on the Edison supercomputer with 1920 CPU cores (MPI tasks).

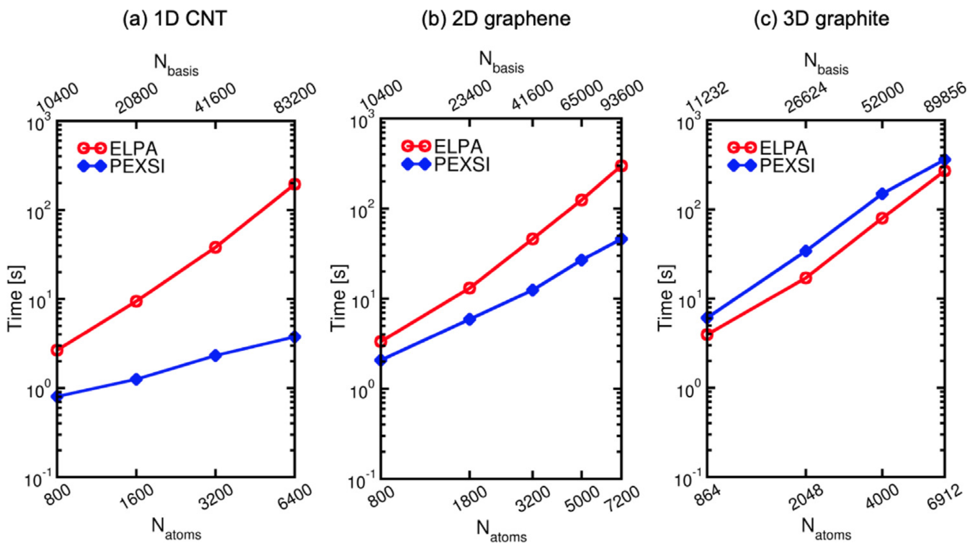

Fig. 4. Performance of key steps in ELPA and PEXSI for (a) carbon nanotube models, (b) graphene models, (c) graphite models, as a function of the number of atoms. ELPA computes all eigenvalues and 50% of the eigenvectors. Results in this figure are obtained by running the SIESTA code on the Edison supercomputer with 1920 CPU cores (MPI tasks).

atoms to about 1000 atoms. Again, it is increasingly beneficial to use PEXSI beyond 1000 atoms, which scales as $O(N^{1.5})$ for 2D systems. The same comparison for 3D graphite models consisting of 864 to 6912 atoms (12,096 to 96,768 basis functions) is shown in Fig. 3(c), where ELPA outperforms PEXSI. The performance of ELPA with fixed computational resources should only depend on the size of the eigenproblem, hence the roughly constant time to solution of ELPA for problems of similar size in Fig. 3(a) (b) and (c). In contrast, the performance of PEXSI heavily depends on the dimensionality of the system, favoring low dimensional systems. This can be attributed to the high sparsity of Hamiltonian matrices and generalized Cholesky factors in such systems. As quantified in the last column of Table 2, given the same number of atoms and basis functions, the Hamiltonian matrix is more sparse in lower dimensional systems where the surface-to-volume ratio is higher. Basis functions belonging to atoms on the surface have less overlap with the other functions, yielding more zero elements in the overlap and Hamiltonian matrices.

The same performance characteristics of ELPA and PEXSI are reproduced in pseudopotential KS-DFT calculations with the SIESTA code. Basis sets, although constructed and tuned differently, are of similar size (SIESTA has one fewer basis function per atom because the 1s state is treated by a pseudopotential). The most notable difference is the width of the eigenspectrum of $\boldsymbol{H}$ and $\boldsymbol{S}$. Timings measured from calculations of 1D carbon nanotube, 2D graphene, and 3D graphite models are shown in Fig. 4(a) (b) and (c), respectively. Again, PEXSI is computationally more efficient than ELPA in 1D and 2D calculations, but not in 3D calculations. For low dimensional systems, PEXSI consistently outperforms ELPA due to the reduced asymptotic complexity, and is particularly promising for very large systems.

Owing to the all-electron formalism implemented in FHI-aims and the pseudopotential formalism implemented in SIESTA, the

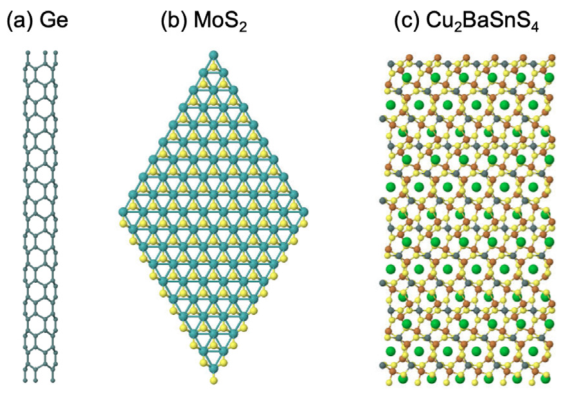

Fig. 5. Atomic structures of (a) 1D Ge nanotube, (b) 2D MoS$_2$ monolayer, and (c) 3D Cu$_2$BaSnS$_4$.

Table 3
Dimension of structures in the heavy benchmark set, and sparsity of the Hamiltonian matrices in the calculations. See caption of Table 2 for definitions of $\tilde{n}_{\text{basis}}$, N_{atom}, and sparsity. Minimal + spd basis sets are used to describe germanium and tin atoms. Minimal + spdf basis sets are used to describe sulfur, copper, molybdenum, and barium atoms. These basis sets are predefined as "tier 1" in FHI-aims. Please refer to Ref. [35] for details on the rationale for basis set definition in FHI-aims.

Code	System	$\tilde{n}_{\text{basis}}$	N_{atom}	Sparsity (%)
FHI-aims	Ge nanotube	27	400–3200	93.6–99.2
FHI-aims	MoS$_2$ monolayer	25.3	300–4800	83.9–99.4
FHI-aims	Cu$_2$BaSnS$_4$	26.8	192–3000	67.8–97.7

lowest energy level in FHI-aims, −275.99 eV, is lower than the lowest level in SIESTA, −24.34 eV, by an order of magnitude. By comparing Fig. 3 with Fig. 4, we confirm that the performance of PEXSI is only mildly affected by the eigenspectrum width. In general, the runtime of PEXSI in Fig. 4 is lower than that in Fig. 3 for any given system. This can be explained by the fact that SIESTA matrices are slightly smaller and more sparse than FHI-aims matrices, as detailed in Table 2. For instance, for the graphite model with 864 atoms, the sparsity factor of the $\boldsymbol{H}$ matrix in SIESTA is 92.5%, i.e. 7.5% matrix elements are non-zero. The $\boldsymbol{H}$ matrix in FHI-aims contains 24.9% non-zero elements, which makes PEXSI slower in FHI-aims calculations.

3.2. Benchmark set II: Heavy elements

The heavy benchmark set is comprised of (quasi) 1D germanium nanotubes, 2D MoS$_2$ monolayers, and 3D Cu$_2$BaSnS$_4$ supercells, as shown in Fig. 5(a) (b) and (c), respectively. Species in this set are intentionally chosen to be heavier than carbon (atomic number: S 16, Cu 29, Ge 32, Mo 42, Sn 50, Ba 56). KS-DFT calculations with the PBE functional are carried out using the all-electron full-potential FHI-aims code, yielding eigenproblems with wide eigenspectra. For instance, the lowest eigenvalue in Cu$_2$BaSnS$_4$ calculations is −38877.57 eV, whereas the lowest value in carbon calculations is only −275.99 eV. For all species, the predefined "tier 1" basis sets [35] are employed, yielding approximately twice as many as basis functions per atom compared to the carbon set. To keep the total number of basis functions comparable to that in the carbon set, models in the heavy set contain fewer atoms. Approximately, the size of matrices in the heavy set ranges from a few thousand to over one hundred thousand, close to the size of matrices in the carbon set. The dimensions of the models, the number of basis functions, and the sparsity factor of the corresponding matrices are summarized in Table 3.

Table 4
Dimension of structures in the bulk benchmark set, and sparsity of the Hamiltonian matrices in the calculations. See caption of Table 2 for definitions of $\tilde{n}_{\text{basis}}$, N_{atom}, and sparsity. Minimal basis sets are used to describe hydrogen, oxygen, and silicon atoms.

Code	System	$\tilde{n}_{\text{basis}}$	N_{atom}	Sparsity (%)
DFTB+	H_2O	2	5184–41,472	87.6–98.5
DFTB+	Si	4	2000–31,250	89.2–99.2

Calculations of the heavy set were performed on the Cray XC40 supercomputer Cori (Haswell partition) at NERSC. Each node of Cori is equipped with two 16-core Intel Haswell processors. 80 nodes were fully exploited by launching 32 MPI tasks on each node, yielding 2560 MPI tasks in total. No OpenMP parallelization was employed.

Fig. 6(a) (b) and (c) show the performance of the ELPA and PEXSI solvers in FHI-aims all-electron calculations of 1D Ge nanotube models (400 to 3200 atoms; 10,800 to 86,400 basis functions), 2D MoS$_2$ models (300 to 4800 atoms; 7600 to 121,600 basis functions), and 3D Cu$_2$BaSnS$_4$ models (192 to 3000 atoms; 5136 to 80,250 basis functions), respectively. The wider eigenspectra and more basis functions per atom in the heavy set do not appear to have a significant impact on the relative performance of the ELPA and PEXSI solvers. The dimensionality and sparsity of the system, as in the carbon set, can greatly influence the efficiency of PEXSI. In particular, PEXSI outperforms ELPA in all 1D Ge nanotube calculations and in 2D MoS$_2$ calculations with 1200 and more atoms. ELPA is still faster for small and dense systems, including 2D MoS$_2$ calculations with fewer than a thousand atoms and all 3D Cu$_2$BaSnS$_4$ calculations.

3.3. Benchmark set III: Bulk materials

Finally, we explore the possibility to accelerate large, 3D calculations using linear scaling density matrix purification methods implemented in the NTPoly library. A series of non-self-consistent-charge DFTB calculations of water clusters and silicon supercells, shown in Fig. 7, are carried out with the DFTB+ code. For hydrogen, oxygen, and silicon atoms, minimal s, sp, and sp basis sets are employed, respectively. The dimensions of the models, the number of basis functions, and the sparsity factor of the corresponding matrices are summarized in Table 4.

Calculations of the bulk set were performed on the Cray XC40 supercomputer Cori at NERSC. Architecture utilization is identical to those in Section 3.2.

Fig. 8 shows the performance of the ELPA, PEXSI, and NTPoly solvers in DFTB+ calculations of water (5,184 to 41,472 atoms; 10,368 to 82,944 basis functions) and silicon (2000 to 31,250 atoms; 8000 to 125,000 basis functions) models. Consistent with Figs. 3, 4, and 6, PEXSI is always slower than ELPA for these 3D structures. The NTPoly solver is able to outperform ELPA by an order of magnitude in all calculations of water molecules, as shown in Fig. 8(a). For the silicon case, NTPoly starts outperforming ELPA when the number of atoms becomes greater than 16,000 (64,000 basis functions). A speed-up of 4 can be achieved for 31,250 atoms (125,000 basis functions), and should be larger for more atoms.

It is worth noting that the density matrix purification parameters used in this subsection, i.e., matrix truncation threshold 10^{-5} and purification convergence criterion 10^{-2}, are crucial to maintaining the sparsity of matrices in intermediate steps in Eq. (8), and therefore achieving linear scaling computational cost. Density matrices computed from these parameters are sufficiently accurate for non-self-consistent calculations. The maximum difference in total energy is only 32.4 μeV/atom between results obtained with NTPoly and ELPA. Nevertheless, slower SCF convergence

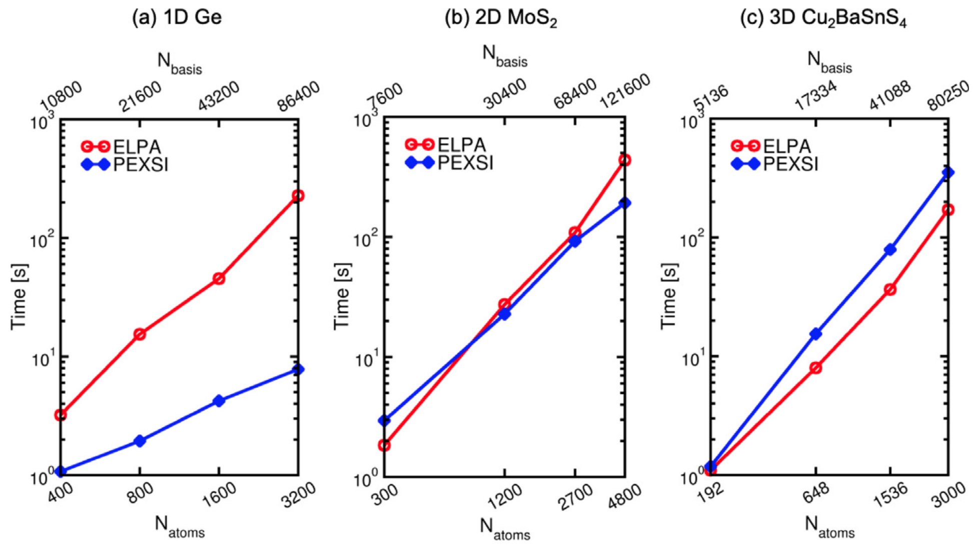

Fig. 6. Performance of key steps in ELPA and PEXSI for (a) Ge nanotube models, (b) MoS_2 monolayer models, (c) Cu_2BaSnS_4 models, as a function of the number of atoms. ELPA computes all the eigenvalues, and 79.6%, 63.8%, and 69.2% of the eigenvectors for the Ge nanotube models, MoS_2 monolayer models, and Cu_2BaSnS_4 models, respectively. Results in this figure are obtained by running the FHI-aims code on the Cori supercomputer (Haswell partition) with 2560 CPU cores (MPI tasks).

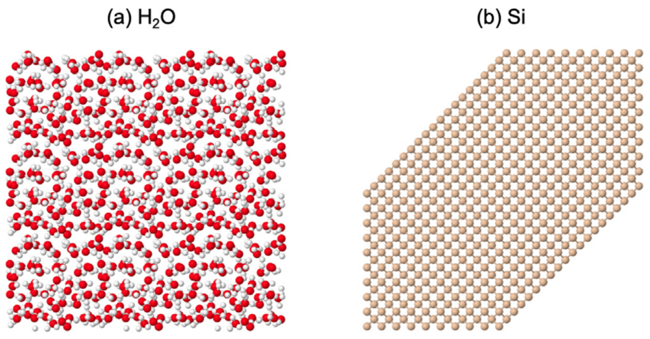

Fig. 7. Atomic structures of (a) water and (b) silicon.

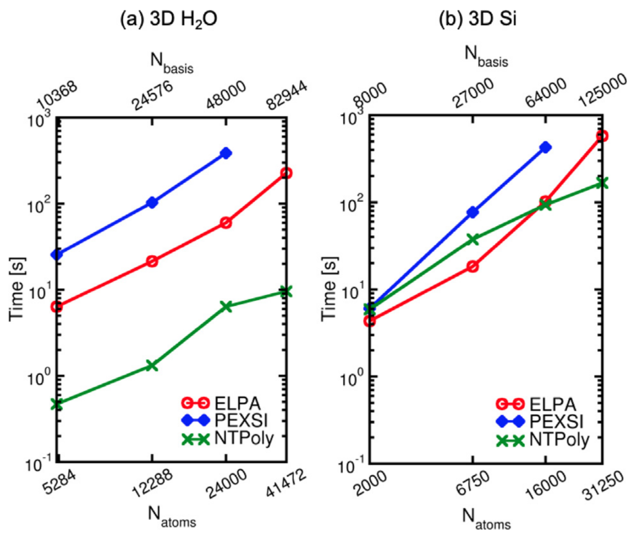

Fig. 8. Performance of key steps in ELPA, PEXSI, and NTPoly for (a) water models and (b) silicon models. ELPA computes all eigenvalues and eigenvectors. Results in this figure are obtained by running the DFTB+ code on the Cori supercomputer (Haswell partition) with 2560 CPU cores (MPI tasks).

may be observed in self-consistent calculations, where tighter parameters need to be considered.

We note that the PEXSI algorithm is highly scalable, as the poles in Eq. (5) can be evaluated fully in parallel. For each pole, the object $(\boldsymbol{H}-(z_l+\mu)\boldsymbol{S})^{-1}$ is computed by the PSelInv (parallel selected inversion) technique, which is able to make efficient use of thousands of CPU cores [43]. With only 20 CPU cores assigned to each pole, PEXSI calculations in Figs. 3, 4, 6, and 8 are far away from the scalability limit of the algorithm [10,20]. To test the parallel performance of the solvers, we take the 41,472-atom water model and the 31,250-atom silicon model in Table 4 and solve them using ELPA, PEXSI, and NTPoly with up to 40,960 CPU cores (MPI tasks). As shown in Fig. 9, the PEXSI solver exhibits a strong scaling superior to ELPA and NTPoly in both the water and silicon test cases. The ELPA solver is faster than PEXSI when using fewer than 20 thousand CPU cores, but it ceases to scale further and becomes slower than PEXSI when using more CPU cores. The NTPoly solver, although its strong scaling is not as good as PEXSI, is still the fastest solver for the two benchmark systems. The scalability of NTPoly may be extended by activating the 3D process grid feature for sparse matrix multiplications in NTPoly [31].

4. Outlook

4.1. Automatic solver selection

As an outcome of the benchmarks and analysis discussed in the previous section, we propose a semi-empirical mechanism to automatically select a solver for arbitrary problems. This mechanism is composed of two layers, a quick decision layer and a direct comparison layer. The former is intended to quickly estimate the solver performance using a few descriptors of a physical system, whereas the latter performs a more rigorous check in case that no quick decision can be made.

As Algorithm 3 shows, the quick decision layer takes four parameters of a physical system as its input, namely the system dimensionality (N_{dim}), the energy gap (E_{gap}), the number of basis functions employed (N_{basis}), and the matrix sparsity factor (defined as $N_{\text{zero}}/N_{\text{basis}}^2$ with N_{zero} being the number of zero matrix elements). Three protocols are implemented in this layer:

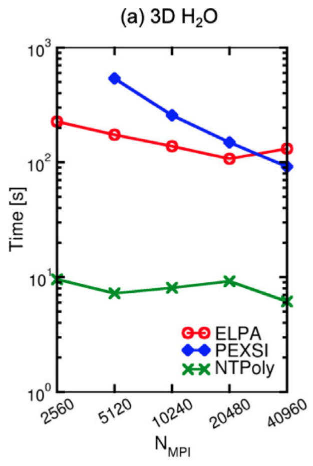

Fig. 9. Performance of key steps in ELPA, PEXSI, and NTPoly for (a) water 41,472-atom model and (b) silicon 31,250-atom model. ELPA computes all eigenvalues and eigenvectors. Results in this figure are obtained by running the DFTB+ code on the Cori supercomputer (Haswell partition).

(1) ELPA is chosen for systems with fewer than 20,000 basis functions, regardless of the values of the other parameters. In benchmarks in Section 3, ELPA has demonstrated its efficiency for small-to-medium-sized systems. Although PEXSI or NTPoly could outperform ELPA for some system geometries and/or potentially different MPI library environments and processor counts even when $N_{\text{basis}} < 20,000$, the performance difference should be rather small and probably not critical. (2) When $N_{\text{basis}} > 20,000$, PEXSI is chosen for 1D/2D systems whose matrix sparsity factor is higher than 95%. In our benchmarks, PEXSI consistently outperforms ELPA for large, sparse, low-dimensional systems thanks to its O(N) and O($N^{1.5}$) scaling for 1D and 2D systems, respectively. Knowledge of the dimensionality of the system currently relies on user's input. (3) When $N_{\text{basis}} > 100,000$, NTPoly is chosen for systems with an energy gap larger than 0.5 eV and a sparsity factor higher than 99%. The linear scaling density matrix purification methods in NTPoly rely on a proper energy gap, which ensures that the density matrix $\boldsymbol{P}$ is sufficiently sparse for large systems. The sparse matrix multiplication kernel in NTPoly efficiently utilizes this sparsity to maximize its performance. At present, the user is responsible for providing an estimate of the energy gap. When no estimate is available, the NTPoly solver will not be considered by the decision layer. Future plans include implementing fast, approximate algorithms to estimate the energy gap [69,70], and integrating more solvers into Algorithm 3.

Algorithm 3 Implementation of the quick decision layer for an automatic solver selection based on the analysis in Sec. 3. Input: system dimensionality N_{dim} (1, 2, or 3), energy gap E_{gap}, number of basis functions N_{basis}, and matrix sparsity factor. Output: choice of solver.

```
function QUICK_DECISION ($N_{\text{dim}}$, $E_{\text{gap}}$, $N_{\text{basis}}$, sparsity, solver)
if ($N_{\text{basis}} < 20,000$) then
    solver = ELPA
else if ($N_{\text{dim}} < 3$ and sparsity $> 95\%$) then
    solver = PEXSI
else if ($N_{\text{basis}} > 100,000$ and sparsity $> 99\%$ and $E_{\text{gap}} > 0.5$ eV) then
    solver = NTPoly
else
    solver = NOT_DECIDED
end if
return solver
```

The conditions in the three protocols above are checked sequentially. If the condition in the nth protocol has been satisfied, then a solver is chosen accordingly, and the (n+1)th and subsequent protocols will never be tested at all. There are cases where a quick decision cannot be made. For instance, a 3D, metallic system with more than 20,000 basis functions does not fall into any of the three protocols. In such cases, the direct comparison layer will take over. In this layer, we test candidate solvers one by one in order to figure out their performances relative to each other. Here, candidate solvers always include ELPA, may include PEXSI if the system is not 3D, and may include NTPoly if the system is highly sparse and not metallic. Suppose there are N_{solver} candidate solvers, we iterate over them in the first N_{solver} SCF steps, with different solvers being used in different SCF steps. Thus, after N_{solver} SCF steps the timings for candidate solvers are measured, from which the optimal solver is identified. By incorporating the direct comparison layer into the SCF cycle, the overhead associated with the solver selection procedure is minimized.

4.2. Reverse communication interface

Iterative eigensolvers are widely used in density-functional theory implementations based on planewave discretization, where the Hamiltonian matrix and the overlap matrix are applied as operators without being explicitly formed, as the sizes of these matrices are too large. Such an operator representation makes it less practical to design a single, universal interface of iterative eigensolvers for a wide range of planewave-based DFT codes, especially for those codes with OpenMP/MPI parallelization. Different DFT codes adopt different distribution patterns of wavefunctions. Implementing iterative solvers supporting a large variety of distribution patterns could still be considered, but such a design lacks extensibility. On the other hand, an interface that sticks to a particular distribution pattern would require conversions between different distribution patterns used by the interface and the user code, which might necessitate larger changes at the user code level and could also become a bottleneck in terms of time and/or memory. Therefore, iterative eigensolvers in ELSI are supported through a reverse communication interface (RCI) framework [71] within the ELSI-RCI subproject – i.e., as shown below, ELSI-RCI provides the algorithmic steps but allows a user code to retain control over matrix and vector distributions as well as other details of the linear algebra that is specific to the user code.

One observation that is common to many iterative eigensolvers is that they only require a limited set of operations, mostly involving the application of the Hamiltonian and overlap operators and basic linear algebra operations such as matrix–vector multiplications. Different iterative eigensolvers only differ in the ordering of these operations (the only exception is the preconditioning step [72]). Since these operations are often already implemented in a DFT code, arranging them in a particular sequence actually yields an iterative eigensolver, and various solvers can be easily implemented by only altering the sequence of the operations. Following this idea, the current version of ELSI-RCI supports the Davidson method [38,39], the orbital minimization method [30,40], the projected preconditioned conjugate gradient method [41], and the Chebyshev filtering method [17,42]. For the user-specified solver, ELSI-RCI provides a sequence of instructions, as shown in Fig. 10. ELSI-RCI has a data type **rci_handle**, which is initialized by the user code. **rci_handle** contains configurations of iterative eigensolvers, e.g., the solver type and the maximum number of iterations, and the status of an execution, e.g., the number of iterations so far. After initialization, ELSI-RCI conducts three stages of computations in a row: allocation stage, solver stage, and deallocation stage. Since different iterative eigensolvers use different numbers of temporary matrices of different sizes, the allocation stage and the deallocation stage are responsible for the allocation and deallocation of these temporary matrices. The allocation and deallocation instructions are given through **rci_solve_allocate** and **rci_solve_deallocate**, respectively. Each of these instructions should be implemented by the user code, i.e., the technical details such as matrix layouts etc. are intended to be defined and controlled by the user code.

The solver stage is the core of ELSI-RCI. It gives various instructions through **rci_solve** with the order depending on the choice of eigensolver. Again, the technical details associated with each instruction should be implemented and thus controlled by the user code. Details of the instructions are encapsulated in a data type **rci_instr**. Once an instruction is given, the user code executes it and returns nothing or a vector to ELSI-RCI. For most operations, no return from the user code is needed, while for some operations related to deflation, a vector of estimated eigenvalues is needed for ELSI-RCI to provide the next instruction. Such a two-step procedure is repeated until the prescribed stopping criterion is satisfied.

Instructions in the solver stage can be classified into three groups: controlling instructions, basic linear algebra instructions, and operator instructions. Controlling instructions include **RCI_NULL**, **RCI_CONVERGE**, and **RCI_STOP**, which indicate that no operation is needed, the chosen iterative eigensolver has converged, and the chosen solver has stopped without reaching convergence, respectively. Basic linear algebra instructions include some of the basic BLAS-level and LAPACK-level operations. Operator instructions include **RCI_H_MULTI** for the application of the Hamiltonian operator, **RCI_S_MULTI** for the application of the overlap operator, and **RCI_P_MULTI** for the application of the preconditioner. Since different preconditioners are preferred for different iterative eigensolvers, the user code is currently expected to prepare and provide preconditioners for the chosen solver to achieve optimal performance.

On the user side, a driver for ELSI-RCI is required, which performs allocation, deallocation, basic linear algebra operations, and operator applications per the instruction from ELSI-RCI. Since almost all these operations are standard parts of existing DFT codes, constructing an ELSI-RCI driver using existing code should be rather straightforward. With this driver, users are then able to call different iterative eigensolvers in ELSI-RCI with a simple change of the solver name in the initialization step of ELSI-RCI. This provides a unified access to a variety of iterative solvers through a single interface.

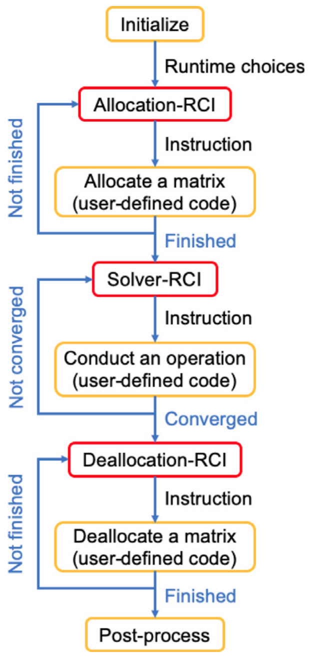

Fig. 10. Flow chart that describes the general workflow of ELSI-RCI. Yellow boxes: Zone of matrices and parallel operations implemented in user's driver. Red boxes: Zone of scalars and sequential operations implemented in ELSI-RCI. Black text outside boxes: Data structures being passed between functions. Depending on the user-specified runtime choices, ELSI-RCI gives instructions on the execution of an iterative eigensolver as well as the allocation/deallocation of matrices needed as workspace. The details of each RCI instruction must be defined and controlled by the user code. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.3. Towards GPU-accelerated high-performance computing

The past ten years witnessed significant growth in the usage of GPU accelerators in high-performance computing (HPC). According to the TOP500 list [73], the number of GPU-accelerated supercomputers skyrocketed from two in 2010 to over a hundred in the most recent release of the list (November 2019). Representatives of state-of-the-art GPU supercomputers include Summit at Oak Ridge National Laboratory [74] and Sierra at Lawrence Livermore National Laboratory [75]. Combining IBM POWER9 CPUs and NVIDIA Volta GPUs, Summit and Sierra rank 1st and 2nd, respectively, on the current TOP500 list. They, as well as many other supercomputers with GPU accelerators, allow for a substantial boost in performance and power efficiency compared to traditional CPU-only machines.

The computation power from GPUs has been utilized by the electronic structure community for larger and faster simulations [76–88]. Various eigensolver and density matrix solver implementations targeting hybrid CPU–GPU machines have been reported. The GPU-enabled ELPA one-stage solver outperforms the CPU version on a few compute nodes when the matrix size exceeds several thousands [23]. Other publicly available GPU-enabled eigensolvers [28,29,89], to our knowledge, are currently limited to shared memory executions on a single compute node. The 2nd order trace resetting density matrix purification method was ported to GPUs [90] by using matrix multiplications provided in the cuBLAS library [91]. Similarly, the GPU-accelerated sparse matrix linear algebra routines in the DBCSR library [92,93] can be employed to compute the density matrix on GPUs.

Several challenges are involved in developing full-fledged eigensolvers and density matrix solvers running on distributed memory, massively parallel, GPU-accelerated supercomputers. MPI communications involving GPUs are more expensive than those between CPUs. GPUs implement a deep memory hierarchy which requires data to be copied out from GPU to CPU to participate in MPI communications. Therefore, existing algorithms must be redesigned to reduce CPU–GPU and GPU–GPU data communications, and to overlap communications with computations as much as possible. In the context of dense eigensolvers, GPUs may perform large amounts of matrix–matrix operations, especially in the two-stage tridiagonalization [22,65], to amortize the cost of data transfers. However, the back-transformation of eigenvectors in the two-stage algorithm is not naturally suited for GPU acceleration, as it lacks an explicit data-parallel computation pattern to be executed on a large amount of GPU cores. On the other hand, several sparse matrix computation techniques do not display large contiguous blocks of data on which GPUs can efficiently operate. Sparse matrix algebra relies heavily on indirect addressing to access data in the sparse matrix. This type of memory access does not generally perform well on GPUs, which prefer memory accesses to be organized in a very specific way, called coalescing, for maximum performance.

The situation is changing with the arrival of new GPU supercomputers such as Summit and Sierra. The new generation of NVIDIA Volta GPUs provides unprecedented memory bandwidth and data transfer speed compared to its predecessors. Intra-node data movements can take advantage of the NVLink technology [94] for high-bandwidth interconnect between GPUs and between GPUs and CPUs. Inter-node data movements can benefit from CUDA-aware MPI, which is now supported by the hardware. In addition, the large cache of Volta GPUs has the potential to substantially accelerate indirect addressing in sparse matrix computations. All of these features make platforms like Summit and Sierra good candidates for accelerating eigensolvers and density matrix solvers. We are planning to leverage them within ELSI. For eigenproblems on individual nodes, the MAGMA library [28,29] has long constituted the state of the art. Additionally, we plan to port and optimize the distributed-parallel ELPA two-stage eigensolver (ELPA2) and the PEXSI and NTPoly density matrix solvers to GPU platforms. In fact, a separate benchmark paper for GPU acceleration in the ELPA one-stage eigensolver already exists [23]. Development of GPU-accelerated ELPA2 [95] and PEXSI is ongoing, and will be reported separately by some of the authors.

5. Conclusions

In this paper, we summarize recent developments of the ELSI electronic solver interface. Among all the upgraded and new features, the following are highlighted:

1. New solvers, namely the upgraded PEXSI density matrix solver, the linear scaling density matrix solver in the NTPoly library, the parallel spectrum slicing sparse eigensolver in the SLEPc library, the penta-diagonalization-based dense eigensolver in the EigenExa library, and the GPU-accelerated, shared-memory dense eigensolvers in the MAGMA library.
2. Two new matrix formats. The SIESTA_CSC format has greatly simplified the integration of ELSI into the SIESTA and DFTB+ electronic structure code packages. The GENERIC_COO format offers maximal flexibility of matrix distribution. It is expected to aid in the integration of ELSI with packages using a custom matrix storage layout.
3. A backward-compatible extension of the interface that allows for parallel calculations of spin-polarized and/or periodic systems.
4. Several routines serving geometry optimization and molecular dynamics calculations. This includes calculation of the energy-weighted density matrix, extrapolation of the density matrix and wavefunctions, and a “smart” reinitialization of ELSI which reuses information across geometry steps.
5. Efficient parallel matrix I/O routines based on MPI I/O.
6. Standardized JSON output via the FortJSON library.

Furthermore, we have assessed the performance of three electronic structure solvers, ELPA, PEXSI, and NTPoly, by running a systematic set of benchmark calculations with both Kohn–Sham density-functional theory and density-functional tight-binding theory. Unsurprisingly, the performance of the solvers depends on the specifics of the problem. For small-to-medium-sized structures up to several hundreds of atoms, the highly optimized dense eigensolver ELPA is always a stable and efficient solution. As the system size increases, the Hamiltonian, overlap, and density matrices become more sparse, rendering lower scaling methods based on sparse linear algebra more favorable. In particular, the pole expansion and selected inversion method PEXSI is best-suited for low dimensional systems. It also exhibits a nearly ideal parallel scalability for at least 40 thousand CPU cores. For large systems with an energy gap, a speed-up over ELPA can be achieved by using the density matrix purification algorithms in NTPoly. These results clearly identify the regimes where sparse density matrix solvers can beat diagonalization, based on which we propose a semi-empirical mechanism to automate the selection of the optimal solver for a given problem. The $O(N^3)$ diagonalization bottleneck in large-scale electronic structure simulations can thus be alleviated to some extent.

On the other hand, our results imply that reaching the crossover point between the diagonalization method and the state-of-the-art density matrix methods would still require many hundreds of, or even thousands of atoms. Nevertheless, the ELSI interface offers a platform for developing, validating, and comparing algorithms, which has been a driving force of improvements in eigensolvers and density matrix solvers. We expect this collaborative effort to continue to lower the computational cost of large-scale electronic structure theory. Our ongoing construction of a reverse communication interface (RCI) framework for iterative eigensolvers will hopefully facilitate the optimal use of eigensolvers in planewave-based density-functional theory implementations. We also hope to support dedicated optimizations targeting GPU architectures, which now seem critical for the realization of exascale electronic structure calculations in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported by the National Science Foundation (NSF), USA under Award No. 1450280. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy (DOE) Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. This research also used resources of the Argonne Leadership Computing Facility, a DOE, USA Office of Science User Facility supported under Contract No. DE-AC02-06CH11357. We appreciate the constructive feedback from many fellow researchers in the electronic structure community, including developers and users of the BigDFT, CP2K, DFTB+, DGDFT, FHI-aims, NTChem, SIESTA projects and of CECAM’s Electronic Structure Library project. Yu was additionally supported by a fellowship from the Molecular Sciences Software Institute under NSF, USA Award No. 1547580. García thanks EU H2020, European Union grant 824143 (“MaX: Materials at the eXascale” CoE), Spain’s AEI (PGC2018-096955-B-C44 and “Severo Ochoa” grant SEV-2015-0496), and GenCat, Spain 2017SGR1506.

Appendix. ELSI build system

Building of the ELSI library is managed by the CMake build system generator. CMake is a cross-platform free and open-source software with support for several build systems (e.g., GNU Make). The build process consists of largely two steps: the configuration step, which generates the build files (e.g., makefiles for GNU Make), and the compilation step.

The requirements for building ELSI are (as of its 2.5.0 release):

- CMake version $\geq$ 3.0
- Fortran compiler (Fortran 2003 compliant)
- C compiler (C99 compliant)
- MPI
- Linear algebra libraries (BLAS, LAPACK, ScaLAPACK)

Additionally, some optional requirements are:

- C++ compiler (C++11 compliant, for PEXSI support)
- SLEPc version 3.13 (for SLEPc-SIPs support)
- PETSc version 3.13 (for SLEPc-SIPs support)
- EigenExa version 2.4 (for EigenExa support)
- MAGMA version 2.5 (for MAGMA support)
- BSEPACK version 0.1 (for BSEPACK support)

During configuration, user-configurable settings for the project are stored in the CMake cache. The cache variables are persistent between builds, allowing the user to make changes to the build configuration with minimal recompilation of the source files. One method of setting the cache variables is to store them in a dedicated file. The command for configuring ELSI would then look as follows:

```
cmake -C <initial_cache.cmake> <elsi>
```

where `<initial_cache.cmake>` is the location of the initial cache file and `<elsi>` is the location of the ELSI source root directory. A template for an ELSI initial cache file is given below, showing the common variables that the user needs to set. Examples for different combinations of compilers and external libraries are shipped with the ELSI package. The full list of cache variables applicable to ELSI are detailed in the ELSI documentation. In order to ensure that the user is aware of all the performance-critical build details, compilers and external libraries are not automatically detected. If the user wants to link against, e.g., linear algebra libraries, then these need to be explicitly specified among the cache variables.

```
# An example CMake initial cache file for ELSI
set(CMAKE_Fortran_COMPILER "mpifort" CACHE STRING "MPI Fortran compiler")
set(CMAKE_C_COMPILER "mpicc" CACHE STRING "MPI C compiler")
set(CMAKE_CXX_COMPILER "mpicxx" CACHE STRING "MPI C++ compiler")
set(CMAKE_Fortran_FLAGS "-O3" CACHE STRING "Fortran flags")
set(CMAKE_C_FLAGS "-O3" CACHE STRING "C flags")
set(CMAKE_CXX_FLAGS "-O3" CACHE STRING "C++ flags")
set(ENABLE_PEXSI ON CACHE BOOL "Enable PEXSI")
set(ENABLE_TESTS ON CACHE BOOL "Enable tests")
set(LIB_PATHS "/path/to/external/libraries" CACHE STRING "Library paths")
set(LIBS "scalapack lapack blas" CACHE STRING "External libraries")
```

By default, all the solver libraries that are redistributed with ELSI are built. There is also the possibility of linking ELSI against any of those libraries compiled externally. This can be beneficial, e.g., for testing the latest upstream changes that are not yet present in a given version of ELSI. As an example, for using ELPA externally, one could turn on `USE_EXTERNAL_ELPA` in the initial cache file and adjust the `INC_PATHS`, `LIB_PATHS`, and `LIBS` variables for including ELPA.

After the configuration step, ELSI may be built by

```
make
```

when using GNU Make, or a more generic command

```
cmake --build <build>
```

where `<build>` is the build directory.

One of the major benefits of using CMake is that ELSI can be easily included in other CMake projects. Only two lines are necessary to link against ELSI,

```
find_package(elsi <version> REQUIRED PATHS <elsi_install>)
target_link_libraries(my_project PRIVATE elsi::elsi)
```

All the necessary libraries and directories with header and modules files, including the main ELSI library, propagate with the single `elsi::elsi` target. The optional argument `<version>` specifies the minimum required version of ELSI. This helps to avoid potential API conflicts with earlier ELSI versions. The arguments `REQUIRED` and `PATHS` specify whether to stop processing if the package is not found and where to search for the ELSI installation, respectively. If ELSI is installed in a standard system location, the `PATHS` argument may be omitted. In order to ensure that ELSI is found only at the specified location, the `NO_DEFAULT_PATH` option of `find_package` may be used.

References

[1] P. Hohenberg, W. Kohn, Phys. Rev. 136 (1964) B864–B871.
[2] W. Kohn, L.J. Sham, Phys. Rev. 140 (1965) A1133–A1138.
[3] W. Kohn, Phys. Rev. Lett. 76 (1996) 3168–3171.
[4] S. Goedecker, Rev. Modern Phys. 71 (1999) 1085–1123.
[5] D.R. Bowler, T. Miyazaki, Rep. Progr. Phys. 75 (2012) 036503.
[6] D.R. Bowler, T. Miyazaki, J. Phys.: Condens. Matter 22 (2010) 074207.
[7] J. VandeVondele, U. Borštnik, J. Hutter, J. Chem. Theory Comput. 8 (2012) 3565–3573.
[8] L. Lin, J. Lu, L. Ying, R. Car, W. E, Commun. Math. Sci. 7 (2009) 755–777.
[9] L. Lin, M. Chen, C. Yang, L. He, J. Phys.: Condens. Matter 25 (2013) 295501.
[10] L. Lin, A. García, G. Huhs, C. Yang, J. Phys.: Condens. Matter 26 (2014) 305503.
[11] W. Jia, L. Lin, J. Chem. Phys. 147 (2017) 144107.
[12] S. Mohr, W. Dawson, M. Wagner, D. Caliste, T. Nakajima, L. Genovese, J. Chem. Theory Comput. 13 (2017) 4684–4698.
[13] C. Campos, J.E. Roman, Numer. Algorithms 60 (2012) 279–295.
[14] M. Keçeli, H. Zhang, P. Zapol, D.A. Dixon, A.F. Wagner, J. Comput. Chem. 37 (2016) 448–459.
[15] Y. Li, H. Yang, Spectrum slicing for sparse Hermitian definite matrices based on Zolotarev's functions, 2017, arXiv:1701.08935.
[16] R. Li, Y. Xi, L. Erlandson, Y. Saad, SIAM J. Sci. Comput. 41 (2019) C393–C415.
[17] A.S. Banerjee, L. Lin, W. Hu, C. Yang, J.E. Pask, J. Chem. Phys. 145 (2016) 154101.
[18] A.S. Banerjee, L. Lin, P. Suryanarayana, C. Yang, J.E. Pask, J. Chem. Theory Comput. 14 (2018) 2930–2946.
[19] J. Winkelmann, P. Springer, E. Di Napoli, ACM Trans. Math. Software 45 (2019) 21.
[20] V. W.-z. Yu, F. Corsetti, A. García, W.P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, Álvaro Vázquez-Mayagoitia, C. Yang, H. Yang, V. Blum, Comput. Phys. Comm. 222 (2018) 267–285.
[21] T. Auckenthaler, V. Blum, H.J. Bungartz, T. Huckle, R. Johanni, L. Kramer, B. Lang, H. Lederer, P.R. Willems, Parallel Comput. 37 (2011) 783–794.
[22] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.J. Bungartz, H. Lederer, J. Phys.: Condens. Matter 26 (2014) 213201.
[23] P. Kůs, A. Marek, S.S. Köcher, H.-H. Kowalski, C. Carbogno, C. Scheurer, K. Reuter, M. Scheffler, H. Lederer, Parallel Comput. 85 (2019) 167–177.
[24] M. Keçeli, F. Corsetti, C. Campos, J.E. Roman, H. Zhang, Álvaro Vázquez-Mayagoitia, P. Zapol, A.F. Wagner, J. Comput. Chem. 39 (2018) 1806–1814.
[25] V. Hernandez, J.E. Roman, V. Vidal, ACM Trans. Math. Software 31 (2005) 351–362.
[26] T. Imamura, S. Yamada, M. Machida, Prog. Nucl. Sci. Technol. 2 (2011) 643–650.
[27] E. Anderson, Z. Bai, C. Bischof, L.S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, SIAM, 1999.
[28] S. Tomov, J. Dongarra, M. Baboulin, Parallel Comput. 36 (2010) 232–240.
[29] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, Numerical Computations with GPUs, Springer, 2014, pp. 3–28.
[30] F. Corsetti, Comput. Phys. Comm. 185 (2014) 873–883.
[31] W. Dawson, T. Nakajima, Comput. Phys. Comm. 225 (2018) 154–165.
[32] M. Shao, H. Felipe, C. Yang, J. Deslippe, S.G. Louie, Linear Algebra Appl. 488 (2016) 148–167.
[33] B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M.Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J.J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I.S. Lee, V. Lutsker, R.J. Maurer, S.K. Min, I. Mitchell, C. Negre, T.A. Niehaus, A.M.N. Niklasson, A.J. Page, A. Pecchia, G. Penazzi, M.P. Persson, J. Řezáč, C.G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, T. Frauenheim, J. Chem. Phys. 152 (2020) 124101.

[34] W. Hu, L. Lin, C. Yang, J. Chem. Phys. 143 (2015) 124110.
[35] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, M. Scheffler, Comput. Phys. Comm. 180 (2009) 2175–2196.
[36] J.M. Soler, E. Artacho, J.D. Gale, A. García, J. Junquera, P. Ordejón, D. Sánchez-Portal, J. Phys.: Condens. Matter 14 (2002) 2745–2779.
[37] M.J.T. Oliveira, N. Papior, Y. Pouillon, V. Blum, E. Artacho, D. Caliste, F. Corsetti, S. de Gironcoli, A.M. Elena, A. García, V.M. García-Suárez, L. Genovese, W.P. Huhn, G. Huhs, S. Kokott, E. Küçükbenli, A.H. Larsen, A. Lazzaro, I.V. Lebedeva, Y. Li, D. López-Durán, P. López-Tarifa, M. Lüders, M.A.L. Marques, J. Minar, S. Mohr, A.A. Mostof, A. O'Cais, M.C. Payne, T. Ruh, D.G.A. Smith, J.M. Soler, D.A. Strubbe, N. Tancogne-Dejean, D. Tildesley, M. Torrent, V. W.-z. Yu, The CECAM electronic structure library, 2020, arXiv:2005.05756.
[38] E.R. Davidson, J. Comput. Phys. 17 (1975) 87–94.
[39] G.L.G. Sleijpen, H.A. Van der Vorst, SIAM J. Matrix Anal. Appl. 17 (1996) 401–425.
[40] J. Lu, H. Yang, Multiscale Model. Simul. 15 (2017) 254–273.
[41] E. Vecharynski, C. Yang, J.E. Pask, J. Comput. Phys. 290 (2015) 73–89.
[42] Y. Zhou, Y. Saad, M.L. Tiago, J.R. Chelikowsky, J. Comput. Phys. 219 (2006) 172–184.
[43] M. Jacquelin, L. Lin, C. Yang, ACM Trans. Math. Software 43 (2016) 21.
[44] M. Jacquelin, L. Lin, C. Yang, Parallel Comput. 74 (2018) 84–98.
[45] J.E. Moussa, J. Chem. Phys. 145 (2016) 164108.
[46] A.M.N. Niklasson, Phys. Rev. B 66 (2002) 155115.
[47] A.H.R. Palser, D.E. Manolopoulos, Phys. Rev. B 58 (1998) 12704–12711.
[48] L.A. Truflandier, R.M. Dianzinga, D.R. Bowler, J. Chem. Phys. 144 (2016) 091102.
[49] P. Pulay, Mol. Phys. 17 (1969) 197–204.
[50] G. Kresse, J. Furthmüller, Phys. Rev. B 54 (1996) 11169.
[51] A.D. Rabuck, G.E. Scuseria, J. Chem. Phys. 110 (1999) 695–700.
[52] M.J. Gillan, J. Phys.: Condens. Matter 1 (1989) 689.
[53] M. Weinert, J.W. Davenport, Phys. Rev. B 45 (1992) 13709.
[54] N.D. Mermin, Phys. Rev. 137 (1965) A1441–A1443.
[55] C.-L. Fu, K.-M. Ho, Phys. Rev. B 28 (1983) 5480–5486.
[56] M. Methfessel, A.T. Paxton, Phys. Rev. B 40 (1989) 3616–3621.
[57] N. Marzari, D. Vanderbilt, A. De Vita, M.C. Payne, Phys. Rev. Lett. 82 (1999) 3296–3299.
[58] P. Löwdin, J. Chem. Phys. 18 (1950) 365–375.
[59] P.G. Mezey, Int. J. Quantum Chem. 63 (1997) 39–48.
[60] A.M.N. Niklasson, M. Challacombe, C.J. Tymczak, K. Németh, J. Chem. Phys. 132 (2010) 124104.
[61] M. Folk, A. Cheng, K. Yates, Proceedings of Supercomputing, Vol. 99, 1999, pp. 5–33.
[62] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, M. Zingale, Supercomputing, 2003 ACM/IEEE Conference, 2003, pp. 39–39.
[63] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J.-P. Prost, M. Snirt, B. Traversat, P. Wong, Input/Output in Parallel and Distributed Computer Systems, Springer, 1996, pp. 127–146.
[64] https://www.ecma-international.org/publications/s/publications/standards/Ecma-404.htm. (Accessed 11 December 2019).
[65] C. Bischof, X. Sun, B. Lang, Proceedings of IEEE Scalable High Performance Computing Conference, 1994, pp. 23–27.
[66] B. Cook, T. Kurth, J. Deslippe, P. Carrier, N. Hill, N. Wichmann, Concurr. Comput.: Pract. Exper. (2018) e4997.
[67] https://git.elsi-interchange.org/elsi-devel/elsi2_cpc_inputs.
[68] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865–3868.
[69] L. Lin, Y. Saad, C. Yang, SIAM Rev. 58 (2016) 34–65.
[70] E. Di Napoli, E. Polizzi, Y. Saad, Numer. Linear Algebra Appl. 23 (2016) 674–692.
[71] J. Dongarra, V. Eijkhout, A. Kalhan, Reverse Communication Interface for Linear Algebra Templates for Iterative Methods, Technical Report, University of Tennessee, Knoxville, TN, USA, 1995.
[72] A.V. Knyazev, Electron. Trans. Numer. Anal. 7 (1998) 104–123.
[73] https://www.top500.org. (Accessed 10 December 2019).
[74] Summit, Oak Ridge National Laboratory, 2019, https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit. (Accessed 11 December 2019).
[75] Sierra, Lawrence Livermore National Laboratory, 2019, https://computation.llnl.gov/computers/sierra. (Accessed 11 December 2019).
[76] K. Yasuda, J. Comput. Chem. 29 (2008) 334–342.
[77] K. Yasuda, J. Chem. Theory Comput. 4 (2008) 1230–1236.
[78] I.S. Ufimtsev, T.J. Martínez, J. Chem. Theory Comput. 4 (2008) 222–231.
[79] I.S. Ufimtsev, T.J. Martínez, J. Chem. Theory Comput. 5 (2009) 1004–1015.
[80] I.S. Ufimtsev, T.J. Martínez, J. Chem. Theory Comput. 5 (2009) 2619–2628.
[81] L. Genovese, M. Ospici, T. Deutsch, J.-F. Méhaut, A. Neelov, S. Goedecker, J. Chem. Phys. 131 (2009) 034103.
[82] S. Maintz, B. Eck, R. Dronskowski, Comput. Phys. Comm. 182 (2011) 1421–1427.
[83] M. Hacene, A. Anciaux-Sedrakian, X. Rozanska, D. Klahr, T. Guignon, P. Fleurat-Lessard, J. Comput. Chem. 33 (2012) 2581–2589.
[84] A.V. Titov, I.S. Ufimtsev, N. Luehr, T.J. Martínez, J. Chem. Theory Comput. 9 (2013) 213–221.
[85] W. Jia, Z. Cao, L. Wang, J. Fu, X. Chi, W. Gao, L.-W. Wang, Comput. Phys. Comm. 184 (2013) 9–18.
[86] W. Jia, J. Fu, Z. Cao, L. Wang, X. Chi, W. Gao, L.-W. Wang, J. Comput. Phys. 251 (2013) 102–115.
[87] L.E. Ratcliff, A. Degomme, J.A. Flores-Livas, S. Goedecker, L. Genovese, J. Phys.: Condens. Matter 30 (2018) 095901.
[88] W.P. Huhn, B. Lange, V. W.-z. Yu, M. Yoon, V. Blum, Comput. Phys. Comm. 254 (2020) 107314.
[89] https://docs.nvidia.com/cuda/cusolver. (Accessed 11 December 2019).
[90] M.J. Cawkwell, E.J. Sanville, S.M. Mniszewski, A.M.N. Niklasson, J. Chem. Theory Comput. 8 (2012) 4094–4101.
[91] https://docs.nvidia.com/cuda/cublas. (Accessed 11 December 2019).
[92] U. Borštnik, J. VandeVondele, V. Weber, J. Hutter, Parallel Comput. 40 (2014) 47–58.
[93] A. Lazzaro, J. VandeVondele, J. Hutter, O. Schütt, Proceedings of the Platform for Advanced Scientific Computing Conference, ACM, 2017, p. 3.
[94] D. Foley, J. Danskin, IEEE Micro 37 (2017) 7–17.
[95] V. W.-z. Yu, J. Moussa, P. Kůs, A. Marek, P. Messmer, M. Yoon, H. Lederer, V. Blum, Gpu-acceleration of the ELPA2 distributed eigensolver for dense symmetric and Hermitian eigenproblems, 2020, arXiv:2002.10991.