ELSI Interface (Development Version) User's Guide

The ELSI Team
elsi-interchange.org

May 15, 2018

Contents

1 Introduction

1.1 The Cubic Wall of Kohn-Sham Density-Functional Theory

Molecular and materials simulations based on Kohn-Sham density-functional theory (KS-DFT) [1] are widely used to provide atomic-scale insights, understanding, and predictions across a wide range of disciplines in the sciences and in engineering.

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

$$\hat{h}^{\mathrm{KS}}\psi_l = \epsilon_l \psi_l, \tag{1.1}$$

where ψ_l and ϵ_l are Kohn-Sham orbitals and their associated eigenenergies, and $\hat{h}^{\mathrm{KS}}$ denotes the Kohn-Sham Hamiltonian. In practice, N_{basis} basis functions $\phi_i(\boldsymbol{r})$ are employed to expand the Kohn-Sham orbitals:

$$\psi_l(\boldsymbol{r}) = \sum_{j=1}^{N_{\mathrm{basis}}} c_{jl}\phi_j(\boldsymbol{r}). \tag{1.2}$$

Using non-orthogonal basis functions (e.g., Gaussian functions, Slater functions, numeric atom-centered orbitals) in Eq. 1.2 converts Eq. 1.1 to a generalized eigenvalue problem that can be expressed in the following matrix form:

$$\boldsymbol{H}\boldsymbol{c} = \boldsymbol{\epsilon}\boldsymbol{S}\boldsymbol{c}. \tag{1.3}$$

Here, $\boldsymbol{H}$ and $\boldsymbol{S}$ are the Hamiltonian matrix and the overlap matrix, whose elements can be computed through numerical integrations. The matrix $\boldsymbol{c}$ and diagonal matrix $\boldsymbol{\epsilon}$ contain the eigenvectors and eigenvalues, respectively, of the eigensystem.

When using orthonormal basis sets (e.g., plane waves, multi-resolution wavelets), the eigenproblem described in Eq. 1.3 reduces to a standard form where $\boldsymbol{S}$ is an identity matrix.

The explicit solution of Eq. 1.3 yields the Kohn-Sham orbitals ψ_l, from which the electron density $n(\boldsymbol{r})$ can be computed following an orbital-based method that scales as $O(N^2)$:

$$n(\boldsymbol{r}) = \sum_{j=1}^{N_{\mathrm{basis}}} f_l \psi_l^*(\boldsymbol{r})\psi_l(\boldsymbol{r}), \tag{1.4}$$

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

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

$$n(\boldsymbol{r}) = \sum_{i,j}^{N_{\text{basis}}} \phi_i^*(\boldsymbol{r}) n_{ij} \phi_j(\boldsymbol{r}), \tag{1.5}$$

with n_{ij} being the elements of the density matrix that need to be computed before the density update.
From a viewpoint of computational complexity, almost all standard pieces of solving the Kohn-Sham equations can be formulated in a linear scaling fashion with respect to the system size. The only piece that can not be easily addressed in an $O(N)$ fashion is the Kohn-Sham eigenvalue problem.

Traditional solutions to the Kohn-Sham eigenvalue problem rely on diagonalization of the Hamiltonian matrix to obtain the Kohn-Sham orbitals ψ_l. The associated computational cost is $O(N^3)$, regardless of the diagonalization algorithm employed. Thanks to a low prefactor, these methods are advantageous to use for systems comprised of up to roughly a few thousands of atoms, which account for the bulk of KS-DFT applications. Systems beyond a few thousands of atoms, however, cannot be efficiently handled by diagonalization due to its cubic scaling nature. Alternatively, approaches with a lower scaling factor, e.g. linear scaling methods, usually directly compute the density matrix as a function of the Hamiltonian matrix. The large prefactor of these methods introduces a significant overhead for small systems, hindering the establishment of lower scaling approaches as mainstream methods of the field. In addition, such alternatives are typically restricted to certain classes of systems or problems. The transition from cubic scaling methods to linear scaling ones is therefore not trivial to automate. In practice, the KS eigenvalue problem remains to be a bottleneck of KS-DFT simulations on current HPC architectures and for system sizes significantly exceeding several thousands of atoms.

1.2 ELSI, the ELectronic Structure Infrastructure

ELSI unifies the community effort in overcoming the cubic-wall problem of KS-DFT by bridging the divide between developers of electronic structure solvers and KS-DFT codes. Via a unified interface, ELSI gives KS-DFT developers easy access to multiple solvers that solve or circumvent the Kohn-Sham eigenproblem efficiently. Solvers are treated on equal footing within ELSI, giving solver developers a unified platform for implementation and benchmarking across codes and physical systems. Solvers may be switched dynamically in an SCF cycle, allowing the KS-DFT developer to mix-and-match strengths of different solvers. Solvers can work cooperatively with one another within ELSI, allowing for acceleration greater than either solver can achieve individually. Most importantly, ELSI exists as a community for KS-DFT and solver developers to interact and work together to improve performance of solvers, with monthly web meetings to discuss progress on code development, yearly on-site "connector meetings", and planned webinars and workshops.

The current version of ELSI supports ELPA [3, 4], libOMM [2], PEXSI [5, 6], and SLEPc-SIPs [7, 8] solvers. Codes currently integrated with ELSI include DFTB+ [9], DGDFT [10], FHI-aims [11], NWChem [12] via Global Arrays, and SIESTA [13].

1.2.1 Design Tenets of ELSI

Versatility: ELSI supports real-valued and complex-valued density matrix, eigenvalue, and eigenvector calculations. A unified software interface designed for rapid integration into a variety of KS-DFT codes is provided. Fortran and C/C++ interfaces are provided.

Flexibility: ELSI supports both dense and sparse matrices as input/output. Supported matrix distribution layouts include 2D block-cyclic distribution, 1D block-cyclic distribution, and 1D block distribution. In situations where the input/output matrix format requested by the calling KS-DFT code and the format used internally by the requested solver are different, conversion and redistribution of matrices will be performed automatically.

Scalability: The solver libraries collected in ELSI are highly scalable. For instance, ELPA can scale to a hundred thousand CPU cores given a sufficiently large problem to solve, and PEXSI, with its efficient two-level parallelism, easily scales to tens of thousands of CPU cores.

Portability: ELSI and its redistributed library source packages have been confirmed to work on commonly-used HPC architectures (Cray, IBM, Intel, NVIDIA) using major compilers (Cray, GNU, IBM, Intel, PGI).

1.3 Kohn-Sham Solver Libraries Supported by ELSI

The current version of ELSI stably supports ELPA [3, 4], libOMM [2], PEXSI [5, 6], and SLEPc-SIPc solvers [7, 8]. The table below summarizes the supported data type, input/output matrix format, and possible output quantities of the solvers in ELSI.

Solver	Data Type	Matrix format	Output
ELPA	Real/complex	Dense/sparse	Eigenvalues, eigenvectors, density matrix, energy-weighted density matrix, chemical potential, electronic entropy
libOMM	Real/complex	Dense/sparse	Density matrix, energy-weighted density matrix
PEXSI	Real/complex	Dense/sparse	Density matrix, energy-weighted density matrix, chemical potential
SLEPc-SIPs	Real	Dense/sparse	Eigenvalues, eigenvectors, density matrix, energy-weighted density matrix, chemical potential, electronic entropy

What follows is a brief introduction of the solvers currently supported in ELSI. For detailed technical descriptions of the solvers, the reader is referred to the original publications of the solvers, e.g., those in the reference list of this document.

1.3.1 ELPA: Eigenvalue Solvers for Petaflop-Applications

ELPA [3, 4] solves the Kohn-Sham eigenvalue problem in Eq. 1.3 by direct diagonalization. In ELPA, the generalized eigenproblem is first transformed to the standard form by using Cholesky factorization of the overlap matrix. Then, the standard eigenproblem is either directly reduced to the tridiagonal form, using similar algorithms as implemented in ScaLAPACK eigensolvers, or first reduced to a banded intermediate form, then to the tridiagonal form. After the eigenvectors of the tridiagonal system are solved, they are back-transformed to the original form in either one step or two steps, depending on the algorithm employed in the diagonalization. Compared to the traditional one-step approach, although the two-step alternative introduces two additional steps (one forward transformation and one backward), it has been shown to enable faster computation and better parallel scalability on present-day computers. Specifically, the matrix-vector operations (BLAS level-2 routines) in the one-step diagonalization can be mostly replaced by more efficient matrix-matrix operations (BLAS level-3 routines) in the two-step version. The additional effort needed by back-transforming the eigenvectors is often alleviated in electronic structure calculations due to the fact that only a relatively small fraction of the eigensolutions need to be retrieved.

1.3.2 libOMM: Orbital Minimization Method

Instead of direct diagonalization, the orbital minimization method (OMM) relies on an iterative algorithm to minimize an unconstrained energy functional. The minimization is carried out in the occupied subspace of the system by using a set of auxiliary Wannier functions expanding this subspace. The minimized OMM energy functional can be shown to be equal to the sum of the eigenvalues of the occupied Kohn-Sham orbitals. Furthermore, the Wannier functions are driven towards perfect orthonormality at this minimum, avoiding an explicit orthonormalization step. The density matrix is then constructed from the final Wannier functions. Although this density matrix is sufficient for the electron density update following Eq. 1.5, without knowledge of individual eigenstates, the OMM method cannot handle systems with fractional occupation numbers.

Different from the originally proposed linear scaling OMM method, the implementation of OMM in the libOMM library [2] is a cubic scaling density matrix solver, utilizing dense matrix-matrix operations. Theoretically, this implementation has a small prefactor than direct diagonalization. In libOMM, the minimization of the OMM energy functional is performed with the conjugate-gradient (CG) method, whose overall performance mainly depends on the convergence rate of the minimization.

1.3.3 PEXSI: Pole Expansion and Selected Inversion

The pole expansion and selected inversion (PEXSI) method [5, 6] provides another alternative way for solving the Kohn-Sham electronic structure without diagonalization. PEXSI represents the density matrix using a pole expansion of the Fermi operator. The number of terms needed by this expansion is proportional to $\log(\beta\Delta E)$, where $\beta = 1/(k_B T)$, k_B is

the Boltzmann constant, T is the electronic temperature, and ΔE is the spectral radius. This logarithmic scaling makes the pole expansion a highly efficient approach. In most cases, ~ 20 poles are already sufficient for the result obtained from PEXSI to be fully comparable to that obtained from diagonalization.

In PEXSI, selected elements of the density matrix, corresponding to the non-zero pattern of $\boldsymbol{H}$ and $\boldsymbol{S}$, are evaluated using a parallel selected inversion algorithm. Its computational cost (for semilocal DFT) depends on the dimensionality of the system: $O(N)$ for 1D systems, $O(N^{1.5})$ for 2D systems, and $O(N^2)$ for 3D systems. This favorable scaling hinges on the sparse character of the Hamiltonian and overlap matrices, but not on any fundamental assumption about the localization properties of the single particle density matrix, i.e., this method is applicable to metallic systems as well as insulating and semi-conducting ones.

Designed in a multi-level parallelism structure, the PEXSI method is highly scalable, and can make efficient use of tens of thousands of processors on high performance computers.

1.3.4 SLEPc-SIPs: Shift-and-Invert Parallel Spectral Transformation Eigensolver in SLEPc

Built on top of the SLEPc and PETSc libraries, SLEPc-SIPs [7, 8] is a parallel sparse eigensolver for real symmetric generalized eigenvalue problems. As its name implies, SLEPc-SIPs partitions the eigenspectrum of a generalized eigenvalue problem into a number of slices. Accordingly, the processes involved in the calculation are split into subgroups, each of which independently solves an eigenvalue problem in one slice. Within the slices, carefully selected shifts are applied to transform the original problem, which accelerates the convergence of the iterative Krylov-Schur eigensolver used in SLEPc-SIPs. With this layer of parallelism across slices, the SLEPc-SIPs solver has the potential to exhibit enhanced scalability over direct diagonalization methods, especially when the load balance across slices can be guaranteed. Indeed, this has been reported to happen with very sparse Hamiltonian and overlap matrices out of density-functional tight-binding (DFTB) calculations [8].

1.4 Acknowledgments

ELSI is a National Science Foundation Software Infrastructure for Sustained Innovation - Scientific Software Integration (SI2-SSI) supported software infrastructure project. The ELSI Interface software and this User's Guide are based upon work supported by the National Science Foundation under Grant Number 1450280. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the National Science Foundation.

2 Installation of ELSI

2.1 Overview

The ELSI package contains the ELSI Interface software as well as redistributed source code for the solver libraries ELPA (version 2016.11.001), libOMM, PEXSI (version 1.0.0). We highly encourage all users to request access to our GitLab server (http://git.elsi-interchange.org/elsi-devel), where we regularly update ELSI between releases while preserving stability.

Starting from the May 2018 release, the installation of ELSI makes use of the CMake software.

2.2 Prerequisites

To build ELSI, the minimum requirements are:

```
CMake  [version 3.0 or newer]
Fortran compiler  [with Fortran 2003]
C compiler  [with C99]
MPI
```

Building the PEXSI solver (highly recommended) requires:

```
C++ compiler  [with C++ 11]
```

Building the SLEPc-SIPs solver requires:

```
SLEPc  [version 3.8.3 only]
PETSc  [version 3.8.4 only, with SuperLU_DIST, MUMPS, ParMETIS, and PT-SCOTCH enabled]
```

Linear algebra libraries should be provided for ELSI to link against:

```
BLAS, LAPACK, BLACS, ScaLAPACK
```

By default, the redistributed ELPA and libOMM libraries will be built. If PEXSI is enabled during configuration, the redistributed PEXSI library and its dependencies, namely the SuperLU_DIST and PT-SCOTCH libraries, will be built as well. Optionally, the redistributed ELPA, libOMM, SuperLU_DIST and PT-SCOTCH libraries may be substituted by user's optimized versions. Please note that in the current version of ELSI, an external version of PEXSI is not officially supported.

2.3 CMake Basics

This section covers some basics of using CMake. Users who are familiar with CMake may safely skip this section.

The typical workflow of using CMake to build ELSI looks like:

```
$ ls

  CMakeLists.txt  external/  src/  test/  ...

$ mkdir build
$ cd build
$ cmake [options] ..

  ...
  ...
  -- Generating done
  -- Build files have been written to: /current/dir

$ make [-j np]
$ make install
```

whenever CMake is invoked, one of the command line arguments must point to the path where the top level CMakeLists.txt file exists, hence the ".." in the above example.

An option can be defined by adding

```
-DKeyword=Value
```

to the command line when invoking CMake. If "Keyword" is of type boolean, its "Value" may be "ON" or "OFF". If "Keyword" is a list of libraries or include directories, its items should be separated with ";" (semicolon) or " " (space).

For example,

```
-DCMAKE_INSTALL_PREFIX=/path/to/install/elsi
-DCMAKE_C_COMPILER=gcc
-DENABLE_TESTS=OFF
-DENABLE_PEXSI=ON
-DINC_PATHS="/path/to/include;/another/path/to/include"
-DLIBS="library1 library2 library3"
```

Available options are introduced in the next sections.

2.4 Configuration

2.4.1 Compilers

CMake automatically detects and sets compilers. The choices made by CMake often work, but not necessarily guarantee the optimal performance. In some cases, the compilers picked up by CMake may not be the ones desired by the user. To build ELSI, it is mandatory that the user explicitly sets the identification of the compilers:

```
-DCMAKE_Fortran_COMPILER=YOUR_MPI_FORTRAN_COMPILER
-DCMAKE_C_COMPILER=YOUR_MPI_C_COMPILER
-DCMAKE_CXX_COMPILER=YOUR_MPI_C++_COMPILER
```

Please note that the C++ compiler is not needed if building ELSI without PEXSI.

In addition, it is highly recommended to specify the compiler flags, in particular the optimization flags:

```
-DCMAKE_Fortran_FLAGS=YOUR_FORTRAN_COMPILE_FLAGS
-DCMAKE_C_FLAGS=YOUR_MPI_C_COMPILE_FLAGS
-DCMAKE_CXX_FLAGS=YOUR_MPI_C++_COMPILE_FLAGS
```

2.4.2 Solvers

The ELPA, libOMM, and PEXSI solver libraries, as well as the SuperLU_DIST and PT-SCOTCH libraries (both required by PEXSI), are redistributed with the current ELSI package.

The redistributed version of ELPA comes with a few "kernels" specifically written to take advantage of processor architecture (e.g. vectorization instruction set extensions). A kernel may be chosen by the ELPA2_KERNEL keyword. Available options are:

```
-DELPA2_KERNEL=BGQ
-DELPA2_KERNEL=AVX
-DELPA2_KERNEL=AVX2
-DELPA2_KERNEL=AVX512
```

for the IBM Blue Gene Q, Intel AVX, Intel AVX2, and Intel AVX512 architectures, respectively. In ELPA, these kernels are employed to accelerate the calculation of eigenvectors, which is often a computational bottleneck when calculating a large percentage of eigenvectors. If this is the case in the user's application, it is highly recommended that the user selects the kernel most suited to their system architecture.

Experienced users are encouraged to link the ELSI interface against external, better optimized solver libraries. Relevant options for this purpose are:

```
-DUSE_EXTERNAL_ELPA=ON
-DUSE_EXTERNAL_OMM=ON
-DUSE_EXTERNAL_SUPERLU=ON
```

The external libraries and the include paths should be set via the following three keywords:

```
-DLIB_PATHS=DIRECTORIES_CONTAINING_YOUR_EXTERNAL_LIBRARIES
-DINC_PATHS=INCLUDE_DIRECTORIES_OF_YOUR_EXTERNAL_LIBRARIES
-DLIBS=NAMES_OF_YOUR_EXTERNAL_LIBRARIES
```

Each of the above keywords is a space-separated or semicolon-separated list. If an external library depends on additional libraries, LIBS should include all the relevant libraries. For instance, LIBS should include the ELPA library and CUDA libraries when using an external ELPA compiled with GPU (CUDA) support; LIBS should include the SuperLU_DIST library and the sparse matrix reordering library used to compile SuperLU_DIST when using an external SuperLU_DIST. Please note that in the current version of ELSI, an external version of PEXSI is not officially supported.

The PEXSI and SLEPc-SIPs solvers are not enabled by default. PEXSI may be activated by specifying:

```
-DENABLE_PEXSI=ON
```

if using redistributed SuperLU_DIST with PT-SCOTCH, or

```
-DENABLE_PEXSI=ON
-DUSE_EXTERNAL_SUPERLU=ON
-DINC_PATHS="/path/to/superlu_dist/include;/path/to/matrix/reordering/include"
-DLIB_PATHS="/path/to/superlu_dist/library;/path/to/matrix/reordering/include"
-DLIBS="superlu_dist;your_choice_of_matrix_reordering_library"
```

if using an externally compiled SuperLU_DIST. SuperLU_DIST 5.3.0, which has been extensively tested with ELSI, is recommended. Older/newer versions may or may not work properly with this version of ELSI.

SLEPc-SIPs may be activated by specifying:

```
-DENABLE_SIPS=ON
-DUSE_EXTERNAL_SUPERLU=ON
-DINC_PATHS="/path/to/slepc/include;/path/to/petsc/include"
-DLIB_PATHS="/path/to/slepc/library;/path/to/petsc/library"
-DLIBS="slepc;petsc;superlu_dist;mumps;parmetis;ptscotch"
```

SLEPc 3.8.3 and PETSc 3.8.4, which have been extensively tested with ELSI, are recommended. Older/newer versions may or may not work properly with this version of ELSI. The PETSc library must be compiled with MPI support, and (at least) with external packages SuperLU_DIST, MUMPS, ParMETIS, and PT-SCOTCH enabled. The SuperLU_DIST library redistributed through ELSI must be turned off by setting USE_EXTERNAL_SUPERLU to "ON", as SuperLU_DIST is already present in the PETSc installation.

2.4.3 Build Targets

By default, a static library (libelsi.a) will be created as the target of the compilation. Building ELSI as a shared library may be enabled by:

```
-DBUILD_SHARED_LIBS=ON
```

Building ELSI test programs may be enabled by:

```
-DENABLE_TESTS=ON
```

In either case, linear algebra libraries, BLAS, LAPACK, BLACS, and ScaLAPACK, should be valid in the LIB_PATHS and LIBS keywords.

If test programs are enabled, the installation of ELSI may be verified by

```
$ ...
$ make [-j np]
$ make test
```

or

```
$ ...
$ make [-j np]
$ ctest
```

Note that the tests may not run if launching MPI jobs is prohibited on the user's working platform.

2.4.4 List of All Configure Options

The options accepted by the ELSI CMake build system are listed here in alphabetical order. Some additional explanations are made below the table.

Option	Type	Default	Explanation
ADD_UNDERSCORE	boolean	ON	Suffix C functions with an underscore
CMAKE_C_COMPILER	string	none	MPI C compiler
CMAKE_C_FLAGS	string	none	C flags
CMAKE_CXX_COMPILER	string	none	MPI C++ compiler
CMAKE_CXX_FLAGS	string	none	C++ flags
CMAKE_Fortran_COMPILER	string	none	MPI Fortran compiler
CMAKE_Fortran_FLAGS	string	none	Fortran flags
CMAKE_INSTALL_PREFIX	path	/usr/local	Path to install ELSI
ELPA2_KERNEL	string	none	ELPA2 kernel
ENABLE_C_TESTS	boolean	OFF	Build C test programs
ENABLE_PEXSI	boolean	OFF	Enable PEXSI support
ENABLE_SIPS	boolean	OFF	Enable SLEPc-SIPs support
ENABLE_TESTS	boolean	OFF	Build Fortran test programs
INC_PATHS	string	none	Include directories of external libraries
LIB_PATHS	string	none	Directories containing external libraries
LIBS	string	none	External libraries
USE_EXTERNAL_ELPA	boolean	OFF	Use external ELPA
USE_EXTERNAL_OMM	boolean	OFF	Use external libOMM and MatrixSwitch
USE_EXTERNAL_SUPERLU	boolean	OFF	Use external SuperLU_DIST

Remarks

1) ADD_UNDERSCORE: In the PEXSI and SuperLU_DIST code redistributed through ELSI, there are calls to functions of the linear algebra libraries, e.g. "dgemm". If ADD_UNDERSCORE is "ON", the code will call "dgemm_" instead of "dgemm". Turn this keyword on if routines are suffixed with "_" in external linear algebra libraries. Turn it off if routines are not suffixed with "_".

2) ELPA2_KERNEL: There are a number of computational kernels available with the ELPA solver. Choose from "BGQ" (IBM Blue Gene Q), "AVX" (Intel AVX), "AVX2" (Intel AVX2), and "AVX512" (Intel AVX512). See 2.4.2 for more information.

3) External libraries: ELSI redistributes source code of ELPA, libOMM, PEXSI, SuperLU_DIST, and PT-SCOTCH libraries, which by default will be built together with the ELSI interface. Experienced users are encouraged to link the ELSI interface against external, better optimized solver libraries. See 2.4.2 for more information.

2.4.5 "Toolchain" Files

It is sometimes convenient to edit the settings in a "toolchain" file that can be read by CMake:

```
-DCMAKE_TOOLCHAIN_FILE=YOUR_TOOLCHAIN_FILE
```

Example "toolchains" are provided in the "./toolchains" directory of the ELSI package, which the user may use as templates to create new ones.

2.5 Importing ELSI into KS-DFT Codes

2.5.1 Linking a KS-DFT Code against ELSI

By default, ELSI places its library and include files in the `lib` and `include` subfolders, respectively, of the build directory. An example set of compiler flags to link a generic KS-DFT code against ELSI are:

```
ELSI_INCLUDE = -I/PATH/TO/BUILD/ELSI/include
```

```
ELSI_LIB     = -L/PATH/TO/BUILD/ELSI/lib -lelsi \
               -lfortjson -lOMM -lMatrixSwitch -lelpa -lpexsi -lsuperlu_dist \
               -lptscotchparmetis -lptscotch -lptscotcherr \
               -lscotchmetis -lscotch -lscotcherr
```

Enabling/disabling PEXSI and SLEPc-SIPs or linking ELSI against preinstalled solver libraries will require the user modify these flags accordingly.

2.5.2 Importing ELSI into a KS-DFT Code

ELSI may be used in the KS-DFT code by importing the appropriate header file. For codes written in Fortran, this is done by using the ELSI module

```
USE ELSI
```

For codes written in C, the ELSI wrapper may be imported by including the header file

```
#include <elsi.h>
```

These import statements give the KS-DFT code access to the ELSI interface. In the next chapter, we will describe the API for the ELSI interface.

3 The ELSI API

3.1 Overview of the ELSI API

In this chapter, we present the public-facing API for the ELSI Interface. We anticipate that fine details of this interface may change slightly in the future, but the fundamental structure of the interface layer is expected to remain consistent.

ELSI provides a C interface in addition to the native Fortran interface. The vast majority of this chapter, while written from a Fortran-ic standpoint, applies equally to both interfaces. Information specifically about the C wrapper for ELSI may be found in Section 3.8.

To allow multiple instances of ELSI to co-exist within a single calling code, we define an `elsi_handle` data type to encapsulate the state of an ELSI instance, i.e., all runtime parameters associated with the ELSI instance.

An `elsi_handle` instance is initialized with the elsi_init subroutine and is subsequently passed to all other ELSI subroutine calls.

3.2 Description of All ELSI APIs

In this section, we introduce the interface subroutines of ELSI. While this section serves as a reference to all subroutines, the user is encouraged to explore the demonstration codes of the ELSI interface in Section 3.7.

3.2.1 Initializing ELSI

The ELSI interface must be initialized via the elsi_init subroutine before any other ELSI subroutine may be called.

elsi_init(handle, solver, parallel_mode, matrix_format, n_basis, n_electron, n_state)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	out	Handle to the current ELSI instance.
solver	integer	in	Desired KS solver. Accepted values are: 1 (ELPA), 2 (LIBOMM), 3 (PEXSI), and 5 (SLEPc-SIPs).
parallel_mode	integer	in	Parallelization mode. See remark 3. Accepted values are: 0 (SINGLE_PROC) and 1 (MULTI_PROC).
matrix_format	integer	in	Matrix format. See remark 1. Accepted values are: 0 (BLACS_DENSE), 1 (PEXSI_CSC), and 2 (SIESTA_CSC).
n_basis	integer	in	Number of basis functions, i.e. global size of Hamiltonian.
n_electron	real double	in	Number of electrons.
n_state	integer	in	Number of states. See remark 2.

Remarks

1) matrix_format: BLACS_DENSE(0) refers a dense matrix format in a 2-dimensional block-cyclic distribution, i.e. the BLACS standard. PEXSI_CSC(1) refers to a compressed sparse column (CSC) matrix format in a 1-dimensional block distribution. SIESTA_CSC(2) refers to a compressed sparse column (CSC) matrix format in a 1-dimensional block-cyclic distribution. As the Hamiltonian, overlap, and density matrices are symmetric (Hermitian), compressed sparse row

(CSR) matrix format is effectively supported.

2) n_state: If ELPA is the chosen solver, this parameter specifies the number of eigenstates to solve by the eigensolver. If libOMM is the chosen solver, n_state must be exactly the number of occupied states, as libOMM cannot handle fractional occupation numbers[2]. PEXSI does not make use of this parameter and a dummy value may be passed.

3) parallel_mode: The two allowed values of parallel_mode, 0 (SINGLE_PROC) and 1 (MULTI_PROC), allow for three parallelization strategies commonly employed by KS-DFT codes. See below.

3a) SINGLE_PROC: Solves the KS eigenproblem following a LAPACK-like fashion. This option may only be selected when ELPA is chosen as the solver. This allows the following parallelization strategy:

SINGLE_PROC Example:

Every MPI task independently handles a group of k-points uniquely assigned to it.

Example number of k-points: 16
Example number of MPI tasks: 4

MPI task 0 handles k-points 1, 2, 3, 4 sequentially;
MPI task 1 handles k-points 5, 6, 7, 8 sequentially;
MPI task 2 handles k-points 9, 10, 11, 12 sequentially;
MPI task 3 handles k-points 13, 14, 15, 16 sequentially.

Pseudocode 1:

```
elsi_init(elsi_h, ..., parallel_mode=0, ...)
...
for i_kpt = 1, n_kpoints_local, 1 do
    elsi_ev_{real|complex}(elsi_h, ham_this_kpt, ovlp_this_kpt, eval_this_kpt, evec_this_kpt)
end
```

3b) MULTI_PROC: Solves the KS eigenproblem following a ScaLAPACK-like fashion. This allows the usage of the following two parallelization strategies:

MULTI_PROC Example:

Groups of MPI tasks coordinate to handle the same k-point, uniquely assigned to that group.

Example number of k-points: 4
Example number of MPI tasks: 16

MPI tasks 0, 1, 2, 3 cooperatively handle k-point 1;
MPI tasks 4, 5, 6, 7 cooperatively handle k-point 2;
MPI tasks 8, 9, 10, 11 cooperatively handle k-point 3;
MPI tasks 12, 13, 14, 15 cooperatively handle k-point 4.

Pseudocode 2:

elsi_init(elsi_h, ..., parallel_mode=1, ...)
elsi_set_mpi(elsi_h, my_mpi_comm)
...
elsi_ev_{real|complex}(elsi_h, my_ham, my_ovlp, my_eval, my_evec)

or

elsi_init(elsi_h, ..., parallel_mode=1, ...)
elsi_set_mpi(elsi_h, my_mpi_comm)
elsi_set_kpoint(elsi_h, n_kpt, my_kpt, my_weight)
elsi_set_mpi_global(elsi_h, mpi_comm_global)
...
elsi_dm_complex(elsi_h, my_ham, my_ovlp, my_dm, global_energy)

In the last example, please note that when there is more than one k-point, a global MPI communicator must be provided for inter-k-points communications. See the documentation of elsi_set_kpoint, elsi_set_spin, and elsi_set_mpi_global.

3.2.2 Setting Up MPI

The MPI communicator used by ELSI is passed into ELSI by the calling code via the elsi_set_mpi subroutine. When there is more than one k-point and/or spin channel, this communicator will be used only for one task defined

elsi_set_mpi(handle, mpi_comm)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
mpi_comm	integer	in	MPI communicator.

3.2.3 Setting Up Matrix Formats

When using the BLACS matrix format (BLACS_DENSE), BLACS parameters are passed into ELSI via the elsi_set_blacs subroutine. The matrix format used internally in the ELSI interface and the ELPA solver requires the block sizes of the 2-dimensional block-cyclic distribution are the same in the row and column directions. It is necessary to call this subroutine before calling any solver interface that makes use of BLACS_DENSE matrix format.

elsi_set_blacs(handle, blacs_context, block_size)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
blacs_context	integer	in	BLACS context.
block_size	integer	in	Block size of the 2D block-cyclic distribution, specifying both row and column directions.

When using the sparse matrix formats, namely 1D block distributed compressed sparse column (PEXSI_CSC) or 1D block-cyclic distributed compressed sparse column (SIESTA_CSC), the sparsity pattern should be passed into ELSI via the elsi_set_csc subroutine. It is necessary to call this subroutine before calling any solver interface that makes use of the sparse matrix formats.

elsi_set_csc(handle, global_nnz, local_nnz, local_col, row_index, col_pointer)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
global_nnz	integer	in	Global number of non-zeros.
local_nnz	integer	in	Local number of non-zeros.
local_col	integer	in	Local number of matrix columns.
col_pointer	integer, rank-1 array	in	Local column pointer array. Dimension: local_nnz.
row_index	integer, rank-1 array	in	Local row index array. Dimension: local_col+1.

When using the 1D block distributed compressed sparse column (PEXSI_CSC) format, the block size of the 1D distribution cannot be set by the user. This is because the PEXSI solver requires that the block size must be n_basis / n_procs, where n_basis is the total number of basis functions, and n_procs is the number of MPI tasks assigned to this problem. When using the 1D block-cyclic distributed compressed sparse column (SIESTA_CSC) format, the block size of the 1D distribution must be set by the user.

elsi_set_csc_blk(handle, block_size)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
global_nnz	integer	in	Block size of the 1D block-cyclic distribution.

3.2.4 Setting Up Multiple k-points and/or Spin Channels

When there is more than one k-point and/or spin channel in the simulating system, the ELSI interface can be configured to support parallel treatment of the k-points and/or spin channels. This is achieved by calling elsi_set_kpoint and elsi_set_spin to pass the required information into ELSI, and calling elsi_set_mpi_global to provide ELSI a global MPI communicator, which is used for a few communications among all the k-points and spin channels. Note that the current ELSI interface only supports the case where the eigenproblems for all the k-points and spin channels are fully parallelized, i.e., there is no MPI task that handles more than one k-point and/or spin channel.

elsi_set_kpoint(handle, n_kpt, i_kpt, weight)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
n_kpt	integer	in	Total number of k-points.
i_kpt	integer	in	Index of the k-point handled by this MPI task.
weight	integer	in	Weight of the k-point handled by this MPI task.

elsi_set_spin(handle, n_spin, i_spin)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
n_spin	integer	in	Total number of spin channels.
i_spin	integer	in	Index of the spin channel handled by this MPI task.

elsi_set_mpi_global(handle, mpi_comm_global)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
mpi_comm_global	integer	in	Global MPI communicator used for communications among all k-points and spin channels.

3.2.5 Finalizing ELSI

When an ELSI instance is no longer needed, its associated handle should be cleaned up by calling elsi_finalize.

elsi_finalize(handle)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.

3.3 Solving Eigenvalues and Eigenvectors

The following subroutines return all the eigenvalues and (a subset of) eigenvectors of the provided H and S matrices. Only eigensolvers may be selected as the desired solver when using these subroutines.

elsi_ev_real(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	real double, rank-2 array	inout	Real Hamiltonian matrix in 2D block-cyclic dense format. See remark 1.
ovlp	real double, rank-2 array	inout	Real overlap matrix (or its Cholesky factor) in 2D block-cyclic dense format. See remark 1.
eval	real double, rank-1 array	inout	Eigenvalues. See remark 2.
evec	real double, rank-2 array	out	Real eigenvectors in 2D block-cyclic dense format. See remark 3.

elsi_ev_complex(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	complex double, rank-2 array	inout	Complex Hamiltonian matrix in 2D block-cyclic dense format. See remark 1.
ovlp	complex double, rank-2 array	inout	Complex overlap matrix (or its Cholesky factor) in 2D block-cyclic dense format. See remark 1.
eval	real double, rank-1 array	inout	Eigenvalues. See remark 2.
evec	complex double, rank-2 array	out	Complex eigenvectors in 2D block-cyclic dense format. See remark 3.

elsi_ev_real_sparse(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	real double, rank-1 array	inout	Real Hamiltonian matrix in 1D block or block-cyclic CSC sparse format.
ovlp	real double, rank-1 array	inout	Real overlap matrix in 1D block or block-cyclic CSC sparse format.
eval	real double, rank-1 array	inout	Eigenvalues. See remark 2.
evec	real double, rank-2 array	out	Real eigenvectors in 2D block-cyclic dense format. See remark 3.

elsi_ev_complex_sparse(handle, ham, ovlp, eval, evec)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	complex double, rank-1 array	inout	Complex Hamiltonian matrix in 1D block or block-cyclic CSC sparse format.
ovlp	complex double, rank-1 array	inout	Complex overlap matrix in 1D block or block-cyclic CSC sparse format.
eval	real double, rank-1 array	inout	Eigenvalues. See remark 2.
evec	complex double, rank-2 array	out	Complex eigenvectors in 2D block-cyclic dense format. See remark 3.

Remarks

1) The Hamiltonian matrix will be destroyed by ELPA during computation. ELPA will overwrite the overlap matrix in its initial execution with the Cholesky factor, which will be reused by subsequent subroutine calls to elsi_ev_real or elsi_ev_complex. When using elsi_ev_real_sparse, the Cholesky factor (which is not sparse) is stored internally in the BLACS_DENSE format. Starting from the second call to elsi_ev_real_sparse, the input sparse overlap matrix will not be referenced.

2) When using the ELPA solver, elsi_ev_real, elsi_ev_complex, elsi_ev_real_sparse, and elsi_ev_complex_sparse always compute all the eigenvalues, regardless of the choice of n_state specified in elsi_init. The dimension of eval thus should always be n_basis.

3) When using the ELPA solver, elsi_ev_real, elsi_ev_complex, elsi_ev_real_sparse, and elsi_ev_complex_sparse compute a subset of all eigenvectors. The number of eigenvectors to compute is specified by the keyword n_state in elsi_init. However, the local eigenvectors array should always be initialized to correspond to a global array of size n_basis × n_basis, whose extra part is used as working space in ELPA. Note that when using elsi_ev_real_sparse and elsi_ev_complex_sparse, the eigenvectors are returned in a dense format (BLACS_DENSE), as they are in general not sparse.

3.4 Computing Density Matrices

The following subroutines return the density matrix computed from the provided H and S matrices, as well as the energy corresponding to the occupied eigenstates. When the selected solver is ELPA, ELSI will internally construct the density matrix using the eigenvalues and eigenvectors returned by ELPA.

elsi_dm_real(handle, ham, ovlp, dm, bs_energy)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	real double, rank-2 array	inout	Real Hamiltonian matrix in 2D block-cyclic dense format.
ovlp	real double, rank-2 array	inout	Real overlap matrix (or Cholesky factor) in 2D block-cyclic dense format. See remark 1.
dm	real double, rank-2 array	out	Real density matrix in 2D block-cyclic dense format.
energy	real double	out	Energy corresponding to the occupied eigenstates ("band structure energy").

elsi_dm_complex(handle, ham, ovlp, dm, energy)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	complex double, rank-2 array	inout	Complex Hamiltonian matrix in 2D block-cyclic dense format.
ovlp	complex double, rank-2 array	inout	Complex overlap matrix (or its Cholesky factor) in 2D block-cyclic dense format. See remark 1.
dm	complex double, rank-2 array	out	Complex density matrix in 2D block-cyclic dense format.
energy	real double	out	Energy corresponding to the occupied eigenstates ("band structure energy").

elsi_dm_real_sparse(handle, ham, ovlp, dm, energy)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	real double, rank-1 array	inout	Non-zero values of the real Hamiltonian matrix in 1D block or block-cyclic CSC format.
ovlp	real double, rank-1 array	inout	Non-zero values of the real overlap matrix in 1D block or block-cyclic CSC format.
dm	real double, rank-1 array	out	Non-zero values of the real density matrix in 1D block or block-cyclic CSC format.
energy	real double	out	Energy corresponding to the occupied eigenstates ("band structure energy").

elsi_dm_complex_sparse(handle, ham, ovlp, dm, energy)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to the current ELSI instance.
ham	complex double, rank-1 array	inout	Non-zero values of the complex Hamiltonian matrix in 1D block or block-cyclic CSC format.
ovlp	complex double, rank-1 array	inout	Non-zero values of the complex overlap matrix in 1D block or block-cyclic CSC format.
dm	complex double, rank-1 array	out	Non-zero values of the complex density matrix in 1D block or block-cyclic CSC format.
energy	real double	out	Energy corresponding to the occupied eigenstates ("band structure energy").

Remarks

1) If the chosen solver is ELPA or libOMM, the Hamiltonian matrix will be destroyed during the computation. ELPA will overwrite the overlap matrix in its initial execution with the Cholesky factor, which will be reused by subsequent calls to elsi_dm_real.

3.5 Customizing ELSI

In ELSI, reasonable default values have been provided for a number of parameters used in the ELSI interface the the supported solvers. However, no set of default parameters can adequately cover all use cases. Parameters that can be overridden are described in the following subsections.

3.5.1 Customizing the ELSI Interface

In all the subroutines listed below, the first argument (input and output) is an elsi_handle. The second argument (input) of each subroutine is the name of parameter to set.

Note that logical variables are not used in all ELSI API. Integers are used to represent logical, with 0 being false and any positive integer being true.

elsi_set_output(handle, out_level)

elsi_set_output_log(handle, out_log)

elsi_set_write_unit(handle, write_unit)

elsi_set_unit_ovlp(handle, unit_ovlp)

elsi_set_zero_def(handle, zero_def)

elsi_set_sing_check(handle, sing_check)

elsi_set_sing_tol(handle, sing_tol)

elsi_set_sing_stop(handle, sing_stop)

elsi_set_uplo(handle, uplo)

elsi_set_mu_broaden_scheme(handle, mu_broaden_scheme)

elsi_set_mu_mp_order(handle, mu_mp_order)

elsi_set_mu_broaden_width(handle, mu_broaden_width)

elsi_set_mu_tol(handle, mu_tol)

Argument	Data Type	Default	Explanation
out_level	integer	0	Output level of the ELSI interface. 0: no output. 1: standard ELSI output. 2: 1 + info from the solvers. 3: 2 + additional debug info.
out_log	integer	0	If not 0, a separate log file in JSON format will be written out.
write_unit	integer	6	The unit used in ELSI to write out information.
unit_ovlp	integer	0	If not 0, the overlap matrix will be treated as an identity (unit) matrix in ELSI and the solvers. See remark 1.
zero_def	real double	10^{-15}	When converting a matrix from dense to sparse format, values below this threshold will be discarded.
sing_check	integer	0	If not 0, the singularity check of the overlap matrix will be performed. See remark 2.
sing_tol	real double	10^{-5}	Eigenfunctions of the overlap matrix with eigenvalues smaller than this threshold will be removed to avoid ill-conditioning. See remark 1.
sing_stop	integer	0	If not 0, the code always stops if the overlap matrix is detected to be singular. See remark 1.
uplo	integer	0	The input matrix format. 0: Input matrix is full, not triangular. This is currently the only supported option.
mu_broaden_scheme	integer	0	The broadening scheme employed to compute the occupation numbers and the Fermi level. 0: Gaussian. 1: Fermi-Dirac. 2: Methfessel-Paxton. 4: Marzari-Vanderbilt.
mu_mp_order	integer	0	The order of the Methfessel-Paxton broadening scheme. No effect if Methfessel-Paxton is not the chosen broadening scheme.
mu_broaden_width	real double	0.01	The broadening width employed to compute the occupation numbers and the Fermi level. See remark 3.
mu_tol	real double	10^{-13}	The convergence tolerance (in terms of the absolute error in electron count) of the bisection algorithm employed to compute the occupation numbers and the Fermi level.

Remarks

1) If the input overlap matrix is an identity matrix, all settings related to the singularity (ill-conditioning) check take no effect.

2) If the singularity check is not disabled, in the first iteration of each SCF cycle, possible singularity of the overlap matrix is checked by computing all its eigenvalues. If there is any eigenvalue smaller than sing_tol, the matrix is considered to be singular.

3) In all supported broadening schemes, there is a term $(\epsilon - E_{\mathrm{F}})/W$ in the distribution function, where ϵ is the energy of an eigenstate, and E_{F} is the Fermi level. The broadening_width parameter should be set to W, in the unit of ϵ and E_{F}.

3.5.2 Customizing the ELPA Solver

elsi_set_elpa_solver(handle, elpa_solver)

elsi_set_elpa_gpu(handle, use_gpu)

elsi_set_elpa_gpu_kernels(handle, use_gpu_kernels)

Argument	Data Type	Default	Explanation
elpa_solver	integer	2	1: ELPA 1-stage solver. 2: ELPA 2-stage solver. The 2-stage solver is usually faster and more scalable.
use_gpu	integer	0	If not 0, try to enable GPU-acceleration in ELPA. See remark 1.
use_gpu_kernels	integer	0	If not 0, try to enable GPU-acceleration and GPU kernels in ELPA. See remark 1.

Remarks

1) use_gpu and use_gpu_kernels: If GPU-acceleration is available in an externally compiled ELPA library, it may be enabled by setting use_gpu to a non-zero value. use_gpu has no effect if GPU-acceleration is not available, which is the case if the internal version of ELPA is used, or if an external ELPA has not been complied with GPU support. Note that by setting use_gpu, the GPU kernels for eigenvector back-transformation will not be used. To enable the GPU kernels, use_gpu_kernels should be set to a non-zero value. Again, this takes no effect if the GPU kernels are not available.

3.5.3 Customizing the libOMM Solver

elsi_set_omm_flavor(handle, omm_flavor)

elsi_set_omm_n_elpa(handle, omm_n_elpa)

elsi_set_omm_tol(handle, omm_tol)

Argument	Data Type	Default	Explanation
omm_flavor	integer	0	Method to perform OMM minimization. See remark 1.
omm_n_elpa	integer	6	Number of ELPA steps before libOMM. See remark 2.
omm_tol	real double	10^{-12}	Convergence tolerance of orbital minimization. See remark 3.

Remarks

1) omm_flavor: Allowed choices are 0 for basic minimization of a generalized eigenproblem and 2 for a Cholesky factorization of the overlap matrix transforming the generalized eigenproblem to the standard form. Usually 2 leads to a faster convergence of the OMM energy functional minimization, at the price of transforming the eigenproblem. When using sufficiently many steps of ELPA to stabilize the SCF cycle, 0 is probably a better choice to finish the remaining SCF cycle. See also remark 2 below.

2) omm_n_elpa: It has been demonstrated that OMM is optimal at later stages of an SCF cycle where the electronic structure is closer to its expected local minimum, requiring only one CG iteration to converge the minimization of

the OMM energy functional. Accordingly, it is recommended to use ELPA initially, then switching to libOMM after omm_n_elpa SCF steps.

3) omm_tol: A large minimization tolerance of course leads to a faster convergence, however unavoidably with a lower accuracy. omm_tol should be tested and chosen to balance the desired accuracy and computation time of the calling code.

3.5.4 Customizing the PEXSI Solver

elsi_set_pexsi_n_pole(handle, pexsi_n_pole)

elsi_set_pexsi_n_mu(handle, pexsi_n_mu)

elsi_set_pexsi_np_per_pole(handle, pexsi_np_per_pole)

elsi_set_pexsi_np_symbo(handle, pexsi_np_symbo)

elsi_set_pexsi_ordering(handle, pexsi_ordering_method)

elsi_set_pexsi_temp(handle, pexsi_temp)

elsi_set_pexsi_gap(handle, pexsi_gap)

elsi_set_pexsi_delta_e(handle, pexsi_delta_e)

elsi_set_pexsi_mu_min(handle, pexsi_mu_min)

elsi_set_pexsi_mu_max(handle, pexsi_mu_max)

elsi_set_pexsi_inertia_tol(handle, pexsi_inertia_tol)

Argument	Data Type	Default	Explanation
pexsi_n_pole	integer	20	Number of poles used by PEXSI. See remark 1.
pexsi_n_mu	integer	2	Number of mu points used by PEXSI. See remark 1.
pexsi_np_per_pole	integer	-	Number of MPI tasks assigned to each mu point. See remark 2.
pexsi_np_symbo	integer	1	Number of MPI tasks for symbolic factorization. See remark 3.
pexsi_ordering	integer	0	Matrix reordering method. See remark 3.
pexsi_temp	real double	0.002	Temperature. See remark 4.
pexsi_gap	real double	0.0	Spectral gap. See remark 5.
pexsi_delta_e	real double	10.0	Spectral radius. See remark 6.
pexsi_mu_min	real double	-10.0	Minimum value of mu. See remark 7.
pexsi_mu_max	real double	10.0	Maximum value of mu. See remark 7.
pexsi_inertia_tol	real double	0.05	Stopping criterion of inertia counting. See remark 7.

Remarks

1) In PEXSI, 20 poles are usually sufficient to get an accuracy that is comparable with the result obtained from diagonalization. The chemical potential is determined by performing Fermi operator expansion at several chemical potential values (referred to as "points" by PEXSI developers) in an SCF step, then interpolating the results at all points to the final answer. The pexsi_n_mu parameter controls the number of chemical potential "points" to be evaluated. 2 points followed by a simple linear interpolation often yield reasonable results.

In short, we recommend pexsi_n_pole = 20 and pexsi_n_mu = 2.

2) pexsi_np_per_pole: PEXSI has, by construction, a 3-level parallelization: the 1st level independently handles all the poles in parallel; within each pole, the 2nd level evaluates the Fermi operator at all the chemical potential points in parallel; finally, within each point, parallel selected inversion is performed as the 3rd level. The value of pexsi_np_per_pole is the number of MPI tasks assigned to a single chemical potential point, for the parallel selected inversion at that point. Ideally, the total number of MPI tasks should be pexsi_np_per_pole × pexsi_n_mu × pexsi_n_pole, that is, all the three levels of the parallelization in PEXSI are fully exploited. In case that this is not feasible, PEXSI can also process the poles in serial, while the points must be handled simultaneously. The user should make sure that the total number of

MPI tasks is a multiple of the number of MPI tasks per point times the number of points. The code will stop if this requirement is not fulfilled.

Assuming the ideal case, the default value for pexsi_np_per_pole is: n_total_mpi / (n_pole × n_mu).

To use the dense density matrix solver interfaces, input and output matrices should be 2D-block-cyclic-distributed among all available MPI tasks. To use the sparse density matrix solver interfaces with the SIESTA_CSC format, input and output matrices should be 1D-block-cyclic-distributed among all available MPI tasks. Note that to use the sparse density matrix solver interfaces with the PEXSI_CSC format, input and output matrices should be 1D-block-distributed on the first pexsi_np_per_pole MPI tasks. PEXSI will then take over and automatically perform any necessary conversions to fit its need.

3) pexsi_np_symbo: The number of MPI tasks used for symbolic factorization cannot be too large. Otherwise, the symbolic factorization may fail. The default number of MPI tasks for symbolic factorization is set to 1 for safety. It is worth testing and increasing this number. The default matrix reordering method in PEXSI is set to parallel ordering (0). This may also fail in rare cases, where the sequential ordering (1) is worth trying.

4) pexsi_temp: This value corresponds to the $1/k_{\mathrm{B}}T$ term (not T) in the Fermi-Dirac distribution function.

5) pexsi_gap: The PEXSI method does not require a gap. If an estimate of the gap is unavailable, the default value usually works.

6) pexsi_delta_e: This is the spectral width of the eigensystem, i.e., the difference between the largest and smallest eigenvalues. Use the default value if no access to a better estimate.

7) The chemical potential determination in PEXSI relies on inertia counting to narrow down the chemical potential searching interval in the first few SCF steps. The pexsi_inertia_tol parameter controls the stopping criterion of the inertia counting procedure. With a small interval obtained from the inertia counting step, PEXSI then selects a number of points in this interval to perform Fermi operator calculations, based on which a final chemical potential will be determined. The trick of this algorithm is that the chemical potential interval of the current SCF step can be used as a descent guess in the next SCF step. Therefore, the mechanism to choose input values for pexsi_mu_min and pexsi_mu_max is two-fold. For the first SCF iteration, they should be set to safe values that guarantee the true chemical potential lies in this interval. Then, for the n^{th} SCF step, pexsi_mu_min should be set to $(mu_{\mathrm{min}}^{\mathrm{n\text{-}1}} + \Delta V_{\mathrm{min}})$, pexsi_mu_max should be set to $(mu_{\mathrm{max}}^{\mathrm{n\text{-}1}} + \Delta V_{\mathrm{max}})$. Here, $mu_{\mathrm{min}}^{\mathrm{n\text{-}1}}$ and $mu_{\mathrm{max}}^{\mathrm{n\text{-}1}}$ are the lower bound and the upper bound of the chemical potential that are determined by PEXSI in the $(\mathrm{n\text{-}1})^{\mathrm{th}}$ SCF step. They can be retrieved by calling elsi_get_pexsi_mu_min and elsi_get_pexsi_mu_max, respectively (see Section 3.6.2. Suppose the effective potential (Hartree potential, exchange-correlation potential, and external potential) is stored in an array V, whose dimension is the number of grid points. From one SCF iteration to the next, ΔV denotes the potential change, and ΔV_{min} and ΔV_{max} are the minimum and maximum values in the array ΔV, respectively. The whole process is summarized in the following pseudo-code.

mu_min = -10.0
mu_max = 10.0
ΔV_{min} = 0.0
ΔV_{max} = 0.0

while *SCF not converged* **do**

Update Hamiltonian

elsi_set_pexsi_mu_min(elsi_h, mu_min + ΔV_{min})
elsi_set_pexsi_mu_max(elsi_h, mu_max + ΔV_{max})

elsi_dm_{real|complex}(elsi_h, ham, ovlp, dm, bs_energy)

elsi_get_pexsi_mu_min(elsi_h, mu_min)
elsi_get_pexsi_mu_max(elsi_h, mu_max)

Update electron density
Update potential

ΔV_{min} = minval(V_{new} - V_{old})
ΔV_{max} = maxval(V_{new} - V_{old})

Check SCF convergence

end

3.5.5 Customizing the SLEPc-SIPs Solver

elsi_set_sips_interval(handle, lower, upper)

elsi_set_sips_n_elpa(handle, n_elpa)

elsi_set_sips_n_slice(handle, n_slice)

Argument	Data Type	Default	Explanation
lower	real double	-2.0	Lower bound of eigenspectrum. See remark 1.
upper	real double	2.0	Upper bound of eigenspectrum. See remark 1.
n_elpa	integer	0	Number of ELPA steps before SLEPc-SIPs. See remark 2.
n_slice	integer	1	Number of slices. See remark 3.

Remarks

1) lower and upper: SLEPc-SIPs relies on some inertia counting steps to estimate the lower and upper bounds of the spectrum. Only eigenvalues within this interval, and their associated eigenvectors, will be solved. The inertia-counting-based eigenvalue searching starts from the interval determined by lower and upper. Depending on the results of inertia counting, this interval may expand or shrink to make sure that the 1^{st} to the n_stateth eigenvalues are all within this interval. If a good estimate of the lower and upper bounds of the eigenspectrum is available, it should be set by elsi_set_sips_interval.

2) n_elpa: The performance of SLEPc-SIPs mainly depends on the load balance across slices. Optimal performance is expected if the desired eigenvalues are evenly distributed across slices. In an SCF calculation, eigenvalues obtained in the current SCF step can be used as an approximated distribution of eigenvalues in the next SCF step. This approximation should become better as the SCF cycle approaches its convergence. On the other hand, at the beginning of an SCF cycle, the load balance is only coarsely checked by inertia calculations. Using the direct eigensolver ELPA in the first n_elpa SCF steps can circumvent the load imbalance of spectrum slicing in the initial SCF steps.

3) n_slice: SLEPc-SIPs partitions the eigenspectrum into slices and solves the slices in parallel. The n_slice parameter controls the number of slices to use in SLEPc-SIPs. The default value, 1, should always work, but by no means leads to the optimal performance of the solver. There are some general rules to set this parameter. Firstly, as a requirement of the SLEPc library, the total number of MPI tasks must by divisible by n_slice. Secondly, setting n_slice to be equal to the number of computing nodes (not MPI tasks) usually yields better performance, as the communication between nodes is minimized in this case. The optimal value of n_slice depends on the actual problem as well as the computing hardware.

3.6 Getting Additional Results from ELSI

In Section 3.3 and Section 3.4, the interfaces to compute and return the eigensolutions and the density matrices have been introduced. Internally, ELSI and the solvers perform additional calculations whose results may only be useful at a certain stage of an SCF calculation. One example is the energy-weighted density matrix that is employed to evaluate the Pulay forces during a geometry optimization calculation. The subroutines introduced in the following subsections are used to retrieve such additional results from ELSI.

3.6.1 Getting Results from the ELSI Interface

In all the subroutines listed below, the first argument (input and output) is an elsi_handle. The second argument (output) of each subroutine is the name of parameter to get.

elsi_get_n_sing(handle, n_sing)

elsi_get_mu(handle, mu)

elsi_get_entropy(handle, ts)

elsi_get_edm_real(handle, edm_real)

elsi_get_edm_complex(handle, edm_complex)

elsi_get_edm_real_sparse(handle, edm_real_sparse)

elsi_get_edm_complex_sparse(handle, edm_complex_sparse)

Argument	Data Type	Explanation
n_sing	integer	Number of eigenvalues of the overlap matrix that are smaller than the singularity tolerance. See Section 3.5.1.
mu	real double	Chemical potential. See remark 1.
ts	real double	Entropy. See remark 1.
edm_real	real double, rank-2 array	Real energy-weighted density matrix in 2D block-cyclic dense format. See remark 2.
edm_complex	complex double, rank-2 array	Complex energy-weighted density matrix in 2D block-cyclic dense format. See remark 2.
edm_real_sparse	real double, rank-1 array	Non-zero values of the real density matrix in 1D block CSC format. See remark 2.
edm_complex_sparse	complex double, rank-1 array	Non-zero values of the complex density matrix in 1D block CSC format. See remark 2.

Remarks

1) In ELSI, the chemical potential will only be available if one of the density matrix solver interfaces has been called, with either ELPA or PEXSI being the chosen solver. The chemical potential can be retrieved by calling elsi_get_mu. The entropy will only be available if one of the density matrix solver interfaces has been called with ELPA being the chosen solver. The user should avoid calling the subroutine when the chemical potential or the entropy is not ready.

2) In general, the energy-weighted density matrix is only needed in a late stage of an SCF cycle to evaluate forces. It is, therefore, not calculated when any of the density matrix solver interface is called. When the energy-weighted density matrix is actually needed, it can be requested by calling the elsi_get_edm subroutines. Note that these subroutines all have the requirement that the corresponding elsi_dm subroutine must have been invoked. For instance, elsi_get_edm_real_sparse only makes sense if elsi_dm_real_sparse has been successfully executed.

3.6.2 Getting Results from the PEXSI Solver

elsi_get_pexsi_mu_min(handle, pexsi_mu_min)

elsi_get_pexsi_mu_max(handle, pexsi_mu_max)

Argument	Data Type	Explanation
pexsi_mu_min	real double	Minimum value of mu. See remark 1.
pexsi_mu_max	real double	Maximum value of mu. See remark 1.

Remarks

1) Please refer to the 7$^{\text{th}}$ remark in Section 3.5.4 for the chemical potential determination algorithm implemented in PEXSI and ELSI.

3.7 Demonstration Pseudo-Code

The typical workflow of ELSI within a KS-DFT code is demonstrated by the following pseudo-code. The ELSI interface routines and the operations to be performed by a KS-DFT code are marked with blue and red, respectively.

3.7.1 2D Block-Cyclic Distributed Dense Matrix + ELSI Eigensolver Interface

---

SCF initialize

elsi_init(elsi_h, ELPA, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
elsi_set_mpi(elsi_h, mpi_comm)
elsi_set_blacs(elsi_h, blacs_ctxt, block_size)

while *SCF not converged* **do**
  Update Hamiltonian

  elsi_ev_{real|complex}(elsi_h, ham, ovlp, eval, evec)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

3.7.2 1D Block Distributed CSC Sparse Matrix + ELSI Eigensolver Interface

---

SCF initialization

elsi_init(elsi_h, ELPA, MULTI_PROC, PEXSI_CSC, n_basis, n_electron, n_state)
elsi_set_mpi(elsi_h, mpi_comm)
elsi_set_blacs(elsi_h, blacs_ctxt, block_size)
elsi_set_csc(elsi_h, global_nnz, local_nnz, local_col, row_index, col_pointer)

while *SCF not converged* **do**
  Update Hamiltonian

  elsi_ev_{real|complex}_sparse(elsi_h, ham, ovlp, eval, evec)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Remarks

1) The SINGLE_PROC parallel mode is not compatible with the PEXSI_CSC matrix format.

2) Although the input Hamiltonian and overlap matrices are in a sparse format, the calculated eigenvectors are in general not sparse. Therefore, the eigenvectors are returned in the BLACS_DENSE format, which is required to be properly set up before calling elsi_ev_{real|complex}.

3.7.3 1D Block-Cyclic Distributed CSC Sparse Matrix + ELSI Eigensolver Interface

---

SCF initialization

elsi_init(elsi_h, ELPA, MULTI_PROC, SIESTA_CSC, n_basis, n_electron, n_state)
elsi_set_mpi(elsi_h, mpi_comm)
elsi_set_blacs(elsi_h, blacs_ctxt, block_size)
elsi_set_csc(elsi_h, global_nnz, local_nnz, local_col, row_index, col_pointer)
elsi_set_csc_blk(elsi_h, block_size)

while *SCF not converged* **do**
  Update Hamiltonian

  elsi_ev_{real|complex}_sparse(elsi_h, ham, ovlp, eval, evec)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Remarks

1) The SINGLE_PROC parallel mode is not compatible with the SIESTA_CSC matrix format.

2) Although the input Hamiltonian and overlap matrices are in a sparse format, the calculated eigenvectors are in general not sparse. Therefore, the eigenvectors are returned in the BLACS_DENSE format, which is required to be properly set up before calling elsi_ev_{real|complex}.

3.7.4 2D Block-Cyclic Distributed Dense Matrix + ELSI Density Matrix Interface

---

SCF initialization

elsi_init(elsi_h, LIBOMM, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
elsi_set_mpi(elsi_h, mpi_comm)
elsi_set_blacs(elsi_h, blacs_ctxt, block_size)

while *SCF not converged* **do**
  Update Hamiltonian

  elsi_dm_{real|complex}(elsi_h, ham, ovlp, dm, bs_energy)
  elsi_get_edm_{real|complex}(elsi_h, edm)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Remarks

1) The SINGLE_PROC parallel mode is not compatible with the ELSI density matrix interface.

3.7.5 1D Block Distributed CSC Sparse Matrix + ELSI Density Matrix Interface

---

SCF initialization

elsi_init(elsi_h, PEXSI, parallel_mode, PEXSI_CSC, n_basis, n_electron, n_state)
elsi_set_mpi(elsi_h, mpi_comm)
elsi_set_csc(elsi_h, global_nnz, local_nnz, local_col, row_index, col_pointer)

while *SCF not converged* **do**
  Update Hamiltonian

  elsi_dm_{real|complex}_sparse(elsi_h, ham, ovlp, dm, bs_energy)
  elsi_get_edm_{real|complex}_sparse(elsi_h, edm)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Remarks

1) The SINGLE_PROC parallel mode is not compatible with the ELSI density matrix interface.

2) Refer to the 7$^{\text{th}}$ remark in Section 3.5.4 for the chemical potential determination algorithm implemented in PEXSI.

3.7.6 1D Block-Cyclic Distributed CSC Sparse Matrix + ELSI Density Matrix Interface

---

SCF initialization

elsi_init(elsi_h, PEXSI, parallel_mode, SIESTA_CSC, n_basis, n_electron, n_state)
elsi_set_mpi(elsi_h, mpi_comm)
elsi_set_csc(elsi_h, global_nnz, local_nnz, local_col, row_index, col_pointer)
elsi_set_csc_blk(elsi_h, block_size)

while *SCF not converged* **do**
    Update Hamiltonian

    elsi_dm_{real|complex}_sparse(elsi_h, ham, ovlp, dm, bs_energy)
    elsi_get_edm_{real|complex}_sparse(elsi_h, edm)

    Update electron density
    Check SCF convergence
end

elsi_finalize(elsi_h)

Remarks

1) The SINGLE_PROC parallel mode is not compatible with the ELSI density matrix interface.

2) Refer to the 7$^{\text{th}}$ remark in Section 3.5.4 for the chemical potential determination algorithm implemented in PEXSI.

3.7.7 Multiple k-points Calculations

```
SCF initialization

elsi_init(elsi_h, ELPA, parallel_mode, BLACS_DENSE, n_basis, n_electron, n_state)
elsi_set_mpi(elsi_h, mpi_comm)
elsi_set_blacs(elsi_h, blacs_ctxt, block_size)

elsi_set_kpoint(elsi_h, n_kpt, i_kpt, weight)
elsi_set_mpi_global(elsi_h, mpi_comm_global)

while SCF not converged do
    Update Hamiltonian

    elsi_dm_{real|complex}(elsi_h, ham, ovlp, dm, bs_energy)
    elsi_get_edm_{real|complex}(elsi_h, edm)

    Update electron density
    Check SCF convergence
end

elsi_finalize(elsi_h)
```

Remarks

1) When there are multiple k-points, other than setting up the k-points and a global MPI communicator, there is no change in the way ELSI solver interfaces are called.

2) Calculations with two spin channels can be set up similarly.

3) One density matrix will be returned for each k-point. The KS-DFT code will take over to assemble the real-space density matrix. The returned band structure energy, however, is already summed over all k-points with respect to the weight of each k-point.

3.8 C/C++ Interface

ELSI is written in Fortran, but many KS-DFT codes use C/C++. ELSI provides a C interface around the core Fortran ELSI code, which can be called from a C or C++ program. Each C wrapper function corresponds to a Fortran subroutine, where we have prefixed the original Fortran subroutine name with `c_` for clarity and consistency. Argument lists are identical to the associated native Fortran subroutine. For the complete definition of the C interface, the user is encouraged to look at the `elsi.h` header file directly.

Bibliography

[1] W. Kohn and L.J. Sham, Self-consistent equations including exchange and correlation effects, Physical Review, 140, 1133-1138 (1965).

[2] F. Corsetti, The orbital minimization method for electronic structure calculations with finite-range atomic basis sets, Computer Physics Communications, 185, 873-883 (2014).

[3] T. Auckenthaler et al., Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, Parallel Computing, 37, 783-794 (2011).

[4] A. Marek et al., The ELPA library: Scalable parallel eigenvalue solutions for electronic structure theory and computational science, Journal of Physics: Condensed Matter, 26, 213201 (2014).

[5] L. Lin et al., Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems, Communications in Mathematical Sciences, 7, 755-777 (2009).

[6] L. Lin et al., Accelerating atomic orbital-based electronic structure calculation via pole expansion and selected inversion, Journal of Physics: Condensed Matter, 25, 295501 (2013).

[7] V. Hernandez et al., SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software, 31, 351-362 (2005).

[8] M. Keceli et al., Shift-and-invert parallel spectral transformation eigensolver: Massively parallel performance for density-functional based tight-binding, Journal of Computational Chemistry, 37, 448-459 (2016).

[9] B. Aradi et al., DFTB+, a sparse matrix-based implementation of the DFTB method, Journal of Physical Chemistry A, 111, 5678 (2007).

[10] W. Hu et al., DGDFT: A massively parallel method for large scale density functional theory calculations, The Journal of Chemical Physics, 143, 124110 (2015).

[11] V. Blum et al., Ab initio molecular simulations with numeric atom-centered orbitals, Computer Physics Communications, 180, 2175-2196 (2009).

[12] M. Valiev et al., NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations, Computer Physics Communications, 181, 1477-1489 (2010).

[13] J.M. Soler et al., The SIESTA method for ab initio order-N materials simulation, Journal of Physics: Condensed Matter, 14, 2745-2779 (2002).

License and Copyright

ELSI Interface software is licensed under the 3-clause BSD license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3) Neither the name of the "ELectronic Structure Infrastructure (ELSI)" project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The source code of ELPA 2016.11.001 (LGPL v3 license), libOMM (2-clause BSD license), PEXSI 1.0.0 (3-clause BSD license), SuperLU_DIST 5.3.0 (3-clause BSD license), and PT-SCOTCH 6.0.5a (CeCILL-C license) are redistributed through this version of ELSI. Individual license of each library can be found in the corresponding subfolder.