ELSI Interface (Development Version) User's Guide

The ELSI Team
elsi-interchange.org

January 2, 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. In the future, we will implement a "decision layer" that recommends the solver most suited for a problem based on the attributes of the problem (matrix size, sparsity, etc). 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 development version of ELSI supports ELPA [3, 4], libOMM [2], and PEXSI [5, 6] solvers. Two new solvers, CheSS [7] and SIPs [8], are being integrated into ELSI. Codes currently integrated with ELSI include DGDFT [9], FHI-aims [10], NWChem [11] via Global Arrays, and SIESTA [12].

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. The default dense matrix format is the "2D block-cyclic distributed dense matrix" as used in BLACS/ScaLAPACK; the default sparse matrix format is the "1D block distributed compressed sparse column matrix" as used in PEXSI. 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 ∼100k CPU cores given a sufficiently large problem to solve, and PEXSI, with its efficient two-level parallelism, easily scales to ∼500k CPU cores.

Portability: Whenever possible, ELSI redistributes source packages for supported solvers, which the user may build alongside the ELSI Interface via ELSI's unified make system. ELSI and its redistributed library source packages have been confirmed to work on commonly-used HPC architectures (Cray, IBM, Intel, NVIDIA) using major compilers (GNU, Intel, IBM, PGI, Cray). ELSI also provides the option for external linkage against preinstalled solver libraries for all supported solvers.

Extensibility: As the ELSI Interface software evolves, more features will be available with minimal API changes. Currently ongoing developments include (1) optimizing the performance of ELPA on hybrid CPU + GPU architectures (e.g. POWER) and on multi-core architectures (e.g. Intel Knights Landing), (2) finalizing the integration of the SIPs sparse eigensolver [8] and the CheSS sparse density matrix solver [7] into ELSI, (3) auto-decision of the solver for a given input problem, and (4) iterative eigensolvers via a reverse communication interface (RCI).

1.3 Kohn-Sham Solver Libraries Supported by ELSI

The current development version of ELSI stably supports ELPA [3, 4], libOMM [2], and PEXSI [5, 6] solvers. Since this user's guide focuses on the ELSI interface layer, given below are very brief introductions of the solvers. For more details and technical descriptions of each solver, the reader is referred to the original publications, e.g., those cited here, as well as an overview paper of the 2017.05 version of ELSI. [14].

1.3.1 ELPA: Eigenvalue soLvers for Petaflop-Applications

ELPA [3, 4] solves the Kohn-Sham eigenvalue problem in Eq. 1.3 by direct (tri)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 traditional ScaLAPACK eigensolvers, or first reduced to a banded intermediate form, then to the tridiagonal form. After the eigenvectors of the resulted tridiagonal system are solved, they must be back-transformed to the original form in either one step or two steps, depending on the algorithm employed in the tridiagonalization. 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 tridiagonalization can be mostly replaced by more efficient matrix-matrix operations (BLAS level-3 routines) in the two-step version of the algorithm. 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.

Since ELPA employs the same 2D block-cyclic matrix distribution as does ScaLAPACK, it can easily be substituted into existing codes that already support parallel linear algebra by ScaLAPACK.

1.3.2 libOMM: Orbital Minimization Method

Instead of direct diagonalization, the orbital minimization method (OMM) relies on efficient iterative algorithms to minimize an unconstrained energy functional using a set of auxiliary Wannier functions that expand the occupied subspace of the system. The OMM energy functional, when minimized, 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. 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 approach cannot handle systems with fractional occupation numbers resulting, e.g., from a finite electronic temperature, such as is typically required for metals.

Compared to other minimization methods with the orthonormality constraint of eigenstates, the advantage of OMM is that it only requires an unconstrained minimization without an explicit orthonormalization step. This makes the OMM a good candidate for linear scaling DFT. However, in order to do so, it is necessary to spatially confine the Wannier functions, which introduces a number of technical difficulties. Recently, the OMM algorithm has been found to result in an efficient iterative solver with conventional cubic scaling but a smaller prefactor than diagonalization. This has been taken by the new implementation in libOMM [2]. The minimization of the OMM energy functional is carried out in

libOMM by using the conjugate-gradient (CG) method, with an efficient preconditioning using the kinetic energy matrix when possible.

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. As a Fermi operator expansion (FOE) based method, PEXSI expands the density matrix using a P-term pole expansion, whose number of terms of the pole expansion is proportional to $\log(\beta\Delta E)$, where $\beta = 1/(k_B T)$ is the inverse of the thermal energy and ΔE is the spectral radius. This logarithmic scaling makes the pole expansion a highly efficient approach to expand the Fermi operator. Typically only ~ 20 poles are 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. The computational cost of the PEXSI technique depends on the dimensionality of the system: $O(N)$ for 1D systems, $O(N^{1.5})$ for 2D systems, and $O(N^2)$ for general 3D bulk systems. This favorable scaling hinges on the sparse character of the Hamiltonian and overlap matrices, but not on any fundamental assumption about the localization properties of the single particle density matrix, i.e., this method is applicable to metallic systems as well as insulating and semi-conducting ones.

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

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 git repository contains the ELSI Interface software as well as redistributed source code for the solver libraries ELPA (version 2017.05.001), libOMM, and PEXSI (preview version 1.0.0). We highly encourage all users to request access to our GitLab server at elsi-team@duke.edu, where we regularly update ELSI between releases while preserving stability.

2.2 Prerequisites

The standard and highly recommended installation of ELSI (compiled with ELPA, libOMM, and PEXSI support) requires the following dependencies:

1. A Fortran 2003 compiler
2. A C compiler supporting the C99 standard
3. A C++ compiler supporting the C++ 11 standard
4. An MPI library
5. BLAS, LAPACK and ScaLAPACK
6. PtScotch 6.0.4 (recommended) or ParMETIS 4.0.3
7. SuperLU_DIST 5.1.3

For those users that are unable to compile PtScotch, ParMETIS or SuperLU_DIST, we also provide a "minimum" installation of ELSI (compiled with ELPA and libOMM support, no PEXSI support) with the following dependencies:

1. A Fortran 2003 compiler
2. A C compiler
3. An MPI library
4. BLAS, LAPACK and ScaLAPACK

Tested compiler suites include Cray, GNU, IBM, Intel, and PGI.

2.3 Installation

The installation of ELSI uses a unified build system which, by default, builds all redistributed solver libraries (ELPA, libOMM, PEXSI) alongside the ELSI interface. Compilation settings are controlled by a make.sys file provided by the user (see Section 2.4). If the user already has well-optimized versions of these solver libraries available, they may link

against the individual solvers externally (see Section 2.4.2 for details.)

Once a properly configured make.sys file specifying compilation settings is provided by the user, the installation process of ELSI is a standard make process:

make	Run the unified ELSI build system.
make check	(Optional) Perform a number of quick tests.
make install	Finalize the ELSI installation.

The resulting libraries, modules, and executables files will be placed in the lib, include, and bin subfolders, respectively, of the ELSI root directory.

The unified ELSI build system will compile all redistributed solvers with the same compilation setting as the ELSI interface layer by default. This behavior may be overwritten by manually specifying compilation flags for each solver (c.f. Appendix A).

2.4 Compilation Settings: The make.sys File

The installation of ELSI is controlled by a user-specified make.sys file. Example make.sys files can be found in the make_sys subfolder of the root directory, which the user may adapt to their own system architecture. make.sys files for various HPCs are included (Titan, Mira, Theta, Edison, Cori) as well as make.sys files targeted towards standard GCC and Intel compilers.

2.4.1 Example make.sys File

We provide an example make.sys file using the Intel compiler suite and libraries (Intel MPI, Intel MKL). More details may be found in the next subsection.

```
# PLATFORM  : Generic Linux Cluster
# COMPILERS : INTEL 17.4
# VERIFIED  : DEC 18, 2017

# ==============================    ELSI    ==================================
#            Compilation settings for the ELSI interface
#            These flags will be used for compiling each redistributed
#            solver unless user explicitly provides flags for said solver
MPIFC         = mpiifort
MPICC         = mpiicc
MPICXX        = mpiicpc

FFLAGS        = -fast -no-ipo -fp-model precise
CFLAGS        = -fast -no-ipo -fp-model precise
CXXFLAGS      = -fast -no-ipo -fp-model precise -std=c++11

CXX_LIB       = -lstdc++
SCALAPACK_LIB = -L/PATH/TO/INTEL/MKL/lib/intel64 -lmkl_scalapack_lp64 \
                -lmkl_blacs_intelmpi_lp64 -lmkl_intel_lp64 \
                -lmkl_sequential -lmkl_core -lpthread -lm

# ============================== MAKE CHECK ==================================
MPI_EXEC      = mpirun  # Optional; recommended
```

```
# ==============================    ELPA    ==================================
#            Compilation flags specifically targeting ELPA.
#            The Generic kernel is the universal "safe" choice.
#            For optimal ELPA performance, use a better kernel
#            targeted for your system architecture!  See ELPA
#            and/or ELSI documentation for more details.
ELPA2_KERNEL   = AVX

# ==============================    PEXSI   ==================================
#            Compilation flags specifically targeting PEXSI.
PTSCOTCH_LIB   = -L/PATH/TO/PTSCOTCH/lib \
                 -lptscotchparmetis -lptscotch -lptscotcherr \
                 -lscotchmetis -lscotch -lscotcherr
SUPERLU_INC    = -I/PATH/TO/SUPERLU_DIST/include
SUPERLU_LIB    = -L/PATH/TO/SUPERLU_DIST/lib -lsuperlu_dist
```

2.4.2 Constructing a make.sys File

In this subsection, we list the important keywords that make up a make.sys file. A full list of supported keywords may be found in Appendix A.

Mandatory Keywords

The mandatory set of keywords contained in the make.sys file for a standard ELSI installation are:

MPIFC	= MPI Fortran compiler
MPICC	= MPI C compiler
MPICXX	= MPI C++ compiler
CXX_LIB	= Standard C++ library, e.g. -lstdc++
SCALAPACK_LIB	= BLAS, LAPACK, ScaLAPACK library flag(s)
PTSCOTCH_LIB	= PtScotch and Scotch library flag(s)
SUPERLU_INC	= SuperLU_Dist include flag(s)
SUPERLU_LIB	= SuperLU_Dist library flag(s)

These keywords have no default values. The compilation will fail if they are not set properly. However, compiler wrappers used on various supercomputering resources may already include libraries and optimization settings, in which case the associated keywords may be left blank.

The choice of MPI and math libraries make a huge difference in the performance of compiled code. Please use the MPI and math libraries recommended by your system administrators.

If the external dependencies PtScotch, ParMETIS, SuperLU_DIST cannot be installed by the user, ELSI may be compiled in a "minimum" installation without PEXSI support by including the following keyword in the make.sys file:

DISABLE_PEXSI	= yes [default: no]

When PEXSI support is disabled, the PTSCOTCH_LIB, PARMETIS_LIB, SUPERLU_INC, and SUPERLU_LIB keywords may be omitted. The minimum installation should be viewed as a last resort, as PEXSI achieves better performance for large-scale calculations compared to libOMM and ELPA.

Compiler Flags

While the keywords in the previous section are sufficient to compile ELSI, maximizing performance requires suitable optimization flags be set. Below are the keywords associated with compiler flags. It is recommended the user uses optimization flags recommended by system administrators.

FFLAGS	= Fortran compiler flags [default: empty]
CFLAGS	= C compiler flags [default: empty]
CXXFLAGS	= C++ compiler flags [default: empty]

make check

Installation of ELSI is checked by using pre-generated matrices. Running "make check" requires a working MPI executable and an environment that permits MPI calculations. The MPI executable may be provided by the user using the keyword:

MPI_EXEC	= MPI executable, e.g. mpiexec, mpirun, aprun, srun [default: mpirun]

We note that many clusters do not allow running MPI jobs on login/compile nodes. In this case, the user will need to request an interactive session or submit a job script. Please see the cluster's documentation for more information.

ELPA2 Kernels

Solver-specific compilation settings for solver libraries redistributed with ELSI are generally handled by the unified ELSI build system, though they may be manually set by experienced users wishing to override the default behavior (c.f. Appendix A). There is one important exception that all users are highly encouraged to set explicitly: the ELPA2 kernel.

The first back-transformation step in ELPA2 for the eigenvectors is a computational bottleneck when a large percentage of all possible eigenvectors are calculated. This step has been heavily optimized using a collection of "kernels" specifically written to take advantage of processor architecture (e.g. vectorization instruction set extensions). The choice of ELPA2 kernels is specified via the ELPA2_KERNEL keyword in the make.sys file. It is highly recommended that the user consults their system administrators and selects the ELPA2 kernel most suited to their system architecture.

Valid choices for the ELPA2_KERNEL keyword are:

Kernel Choice	Description
Generic	The generic ELPA2 Fortran kernel, which is expected to work on all platforms. [default]
BGQ	Fortran code enhanced with assembler calls for IBM Blue Gene/Q
SSE	x86_64 assembler code using SSE instructions
AVX	Optimized intrinsic code using AVX instructions
AVX2	Optimized intrinsic code using AVX2 instructions
AVX512	Optimized intrinsic code using AVX512 instructions

C Binding

By default, the ELSI installer creates Fortran and C interfaces, and builds test programs written in both languages. The linkers to link Fortran and C programs and associated flags may be specified in the make.sys file using the following keywords:

LINKER	= Linker [default: $(MPIFC)]
FLILNKER	= Linker for Fortran programs [default: $(LINKER)]
CLINKER	= Linker for C programs [default: $(LINKER)]

LDFLAGS	= Linker flags [default: $(FFLAGS)]
FLDFLAGS	= Linker flags specific to Fortran programs [default: $(FFLAGS)]
CLDFLAGS	= Linker flags specific to C programs [default: $(CFLAGS)]

When using a Fortran compiler to link a C test program, the flag "-nofor-main" (Intel compiler; counterpart exists for some other compilers) may be necessary in order to indicate that the main program is not written in Fortran.

External Linkage against Preinstalled Solver Libraries

By default, the unified ELSI build system will build the ELPA, libOMM, and PEXSI solver libraries redistributed with ELSI, as well as the ELSI interface itself. Optionally, the user may link ELSI directly against a preinstalled version of any (or all) supported solver library, bypassing the solver version redistributed with ELSI. It is the responsibility of the user to verify that their preinstalled solver library is properly compiled and that the API of the preinstalled solver library matches the API that ELSI expects from the solver.

The relevant keywords for the make.sys file are below, organized by solver. An example make.sys demonstrating external linkage is provided in ./make_sys/make.sys-external-lib. The solver version expected by ELSI is also indicated.

1) ELPA 2017.05.001+

EXTERNAL_ELPA	= yes [default: no]
ELPA_INC	= ELPA include flag(s) [default: built-in version]
ELPA_LIB	= ELPA library flag(s) [default: built-in version]

2) libOMM

EXTERNAL_OMM	= yes [default: no]
OMM_INC	= libOMM include flag(s) [default: built-in version]
OMM_LIB	= libOMM library flag(s) [default: built-in version]

To enable external libOMM, the MatrixSwitch library in the OMM bundle must be available in OMM_LIB.

3) PEXSI 1.0.0 (preview)

EXTERNAL_PEXSI	= yes [default: no]
PEXSI_INC	= PEXSI include flag(s) [default: built-in version]
PEXSI_LIB	= PEXSI library flag(s) [default: built-in version]

2.5 Linking and Importing ELSI into KS-DFT Codes

2.5.1 Linking a KS-DFT Code against ELSI

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

```
ELSI_INCLUDE = -I/PATH/TO/ELSI/include
ELSI_LIB     = -L/PATH/TO/ELSI/lib -lelsi \
               -lOMM -lMatrixSwitch -lelpa -lpexsi \
               -L$/PATH/TO/SUPERLU/lib -lsuperlu_dist \
               -L$/PATH/TO/PTSCOTCH/lib -lptscotchparmetis \
               -lptscotch -lptscotcherr -lscotchmetis \
               -lscotch -lscotcherr-lparmetis -lmetis
```

These flags may vary based on the user's needs. Linking ELSI against preinstalled solver libraries will require the user modify these flags accordingly.
If PEXSI is disabled by setting DISABLE_PEXSI = yes in the make.sys file (i.e. the "minimum" installation of ELSI), an example set of the resulting compiler flags would be:

```
ELSI_INCLUDE = -I/PATH/TO/ELSI/include
ELSI_LIB     = -L/PATH/TO/ELSI/lib -lelsi \
               -lOMM -lMatrixSwitch -lelpa
```

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/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 (public-facing) 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, since the C interface is deliberately kept aligned with the Fortran version. 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. For the Fortran code, this data type is defined in the `ELSI` module, and for C/C++ code this data type is defined in the `elsi.h` header file. The `elsi_handle` data type contains 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. The recommended method for modifying the state of the the ELSI instance is to use ELSI's public-facing API, as will be demonstrated in following sections.

3.2 Description of All ELSI APIs

In this section, we introduce the public-facing 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	The desired KS solver. Accepted values are: 1 (ELPA), 2 (LIBOMM), and 3 (PEXSI).
parallel_mode	integer	in	The type of parallelization used. See comment 3. Accepted values are: 0 (SINGLE_PROC) and 1 (MULTI_PROC).
matrix_format	integer	in	The matrix format used for the matrices handled by ELSI (Hamiltonian, overlap, density matrix, etc.) See comment 1. Accepted values are: 0 (BLACS_DENSE) and 1 (PEXSI_CSC).
n_basis	integer	in	Number of basis functions, i.e. the global size of the Hamiltonian matrix.
n_electron	real double	in	Number of electrons.
n_state	integer	in	Number of states, as interpreted by the solver. See comment 2.

Comments

1) matrix_format: BLACS_DENSE(0) is a dense matrix format using a 2-dimensional block-cyclic distribution, i.e. the BLACS standard. PEXSI_CSC(1) refers to a compressed sparse column (CSC) matrix format using a 1-dimensional block distribution.

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 1D block distributed compressed sparse column matrix format (PEXSI_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 PEXSI_CSC format.

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	Column pointer array for the CSC storage format in 1D block CSC format. Dimension: local_nnz.
row_index	integer, rank-1 array	in	Row index array for the CSC storage format. Dimension: local_col+1.

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 comment 1.
ovlp	real double, rank-2 array	inout	Real overlap matrix (or its Cholesky factor) in 2D block-cyclic dense format. See comment 1.
eval	real double, rank-1 array	inout	Eigenvalues. See comment 2.
evec	real double, rank-2 array	out	Real eigenvectors in 2D block-cyclic dense format. See comment 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 comment 1.
ovlp	complex double, rank-2 array	inout	Complex overlap matrix (or its Cholesky factor) in 2D block-cyclic dense format. See comment 1.
eval	real double, rank-1 array	inout	Eigenvalues. See comments 2.
evec	complex double, rank-2 array	out	Complex eigenvectors in 2D block-cyclic dense format. See comment 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 CSC sparse format.
ovlp	real double, rank-1 array	inout	Real overlap matrix in 1D block CSC sparse format.
eval	real double, rank-1 array	inout	Eigenvalues. See comment 2.
evec	real double, rank-2 array	out	Real eigenvectors in 2D block-cyclic dense format. See comment 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 CSC sparse format.
ovlp	complex double, rank-1 array	inout	Complex overlap matrix in 1D block CSC sparse format.
eval	real double, rank-1 array	inout	Eigenvalues. See comment 2.
evec	complex double, rank-2 array	out	Complex eigenvectors in 2D block-cyclic dense format. See comment 3.

Comments

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 not necessarily 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, 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 comment 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.

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 comment 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.

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 CSC format.
ovlp	real double, rank-1 array	inout	Non-zero values of the real overlap matrix in 1D block CSC format.
dm	real double, rank-1 array	out	Non-zero values of the real density matrix in 1D block CSC format.
energy	real double	out	Energy corresponding to the occupied eigenstates.

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 CSC format.
ovlp	complex double, rank-1 array	inout	Non-zero values of the complex overlap matrix in 1D block CSC format.
dm	complex double, rank-1 array	out	Non-zero values of the complex density matrix in 1D block CSC format.
energy	real double	out	Energy corresponding to the occupied eigenstates.

Comments

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 subroutine 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. All parameters can be overridden through ELSI setter subroutines, as 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_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_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.
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 comment 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 comment 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 comment 1.
sing_stop	integer	0	If not 0, the code always stops if the overlap matrix is detected to be singular. See comment 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. 2: Methfessel-Paxton 0th order. 3: Methfessel-Paxton 1st order.
mu_broaden_width	real double	0.01	The broadening width employed to compute the occupation numbers and the Fermi level.
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.

Comments

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

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.

3.5.2 Customizing the ELPA Solver

elsi_set_elpa_solver(handle, elpa_solver)

Argument	Data Type	Default	Explanation
elpa_solver	integer	2	1: ELPA 1-stage solver. 2: ELPA 2-stage solver. 2 is usually faster and more scalable.

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	2	Method to perform OMM minimization. See comment 1.
omm_n_elpa	integer	5	Number of ELPA steps before libOMM. See comment 2.
omm_tol	real double	10^{-10}	Convergence tolerance of orbital minimization. See comment 3.

Comments

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, 1 is a better choice to finish the remaining SCF cycle with possibly higher speed the ELPA or OMM flavor 2. See also comment 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 comment 1.
pexsi_n_mu	integer	2	Number of mu points used by PEXSI. See comment 1.
pexsi_np_per_pole	integer	-	Number of MPI tasks assigned to each mu point. See comment 2.
pexsi_np_symbo	integer	1	Number of MPI tasks for symbolic factorization. See comment 3.
pexsi_ordering_method	integer	0	Matrix reordering method. See comment 3.
pexsi_temp	real double	0.002	Temperature. See comment 4.
pexsi_gap	real double	0	Spectral gap. See comment 5.
pexsi_delta_e	real double	10	Spectral radius. See comment 6.
pexsi_mu_min	real double	-10	Minimum value of mu. See comment 7.
pexsi_mu_max	real double	10	Maximum value of mu. See comment 7.
pexsi_inertia_tol	real double	0.05	Stopping criterion of inertia counting. See comment 7.

Comments

1) Starting from PEXSI v1.0.0, the pole expansion is carried out by using the method proposed by Moussa. Usually, 20 poles are enough to get an accuracy that is comparable with the result obtained from diagonalization. Also introduced in PEXSI v1.0.0 is a robust and fast (no longer iterative) algorithm to determine the chemical potential, which performs Fermi operator expansion at several chemical potential values (referred to as "points" by PEXSI developers) in an SCF step, then interpolates the results at all points to get the final answer. The pexsi_n_mu setter controls the number of

chemical potential "points" to be evaluated. In most cases, 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 in parallel as the 3rd level of parallelization. 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. As a consequence, 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).

If the dense density matrix solver interfaces are called, the input and output matrices should be 2D-block-cyclic-distributed among all available MPI tasks. In contrast, note that if the sparse density matrix solver interfaces and called, the 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 larger than a magic number, depending on the matrix and the machine. The default value 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 using PtScotch or ParMETIS (0). The sequential ordering (1) may be necessary in rare cases.

4) pexsi_temp: This value corresponds to the $1/k_BT$ term 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 newly designed chemical potential determination algorithm relies on inertia counting to narrow down the chemical potential searching interval in the first few SCF steps. The pexsi_inertia_tol setter 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 nth SCF step, pexsi_mu_min should be set to $(mu_{min}^{n-1}+\Delta V_{min})$, pexsi_mu_max should be set to $(mu_{max}^{n-1}+\Delta V_{max})$. Here, mu_{min}^{n-1} and mu_{max}^{n-1} are the lower bound and the upper bound of the chemical potential that are determined by PEXSI in the (n-1)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, 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
```

3.6 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 not necessarily be useful for a normal SCF calculation, but can be valuable in some cases. One example is the energy-weighted density matrix that is employed to evaluate the Pulay forces during a geometry optimization calculation. The ELSI getter subroutines are used to retrieve such additional results from ELSI, as will be described in the following subsections.

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_ovlp_sing(handle, ovlp_sing)

elsi_get_n_sing(handle, n_sing)

elsi_get_mu(handle, mu)

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
ovlp_sing	integer	0 if the overlap matrix is singular; 1 if it is not.
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 comment 1.
edm_real	real double, rank-2 array	Real energy-weighted density matrix in 2D block-cyclic dense format. See comment 2.
edm_complex	complex double, rank-2 array	Complex energy-weighted density matrix in 2D block-cyclic dense format. See comment 2.
edm_real_sparse	real double, rank-1 array	Non-zero values of the real density matrix in 1D block CSC format. See comment 2.
edm_complex_sparse	complex double, rank-1 array	Non-zero values of the complex density matrix in 1D block CSC format. See comment 2.

Comments

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

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 comment 1.
pexsi_mu_max	real double	Maximum value of mu. See comment 1.

Comments

1) Please refer to the 7^{th} comment 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_set_elpa_solver(elsi_h, 2)
  elsi_ev_{real|complex}(elsi_h, ham, ovlp, eval, evec)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Comments

1) The MULTI_PROC parallel mode is compatible with ELPA and SIPs; the SINGLE_PROC mode is only compatible with ELPA.

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_set_unit_ovlp(elsi_h, 1)
  elsi_ev_{real|complex}_sparse(elsi_h, ham, dummy, eval, evec)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Comments

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 not necessarily in the same sparsity pattern. In fact, they can be not sparse at all. Therefore, the eigenvectors are returned in the BLACS_DENSE format, which is therefore required to be set up.

3.7.3 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_set_omm_tol(elsi_h, 1d-12)
  elsi_dm_{real|complex}(elsi_h, ham, ovlp, dm, energy)
  elsi_get_edm_{real|complex}(elsi_h, edm)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Comments

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

2) The returned energy corresponds to the eigenenergies of the occupied Kohn-Sham orbitals.

3.7.4 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, energy)
  elsi_get_edm_{real|complex}_sparse(elsi_h, edm)

  Update electron density
  Check SCF convergence
end

elsi_finalize(elsi_h)

Comments

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

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

3) The returned energy corresponds to the eigenenergies of the occupied Kohn-Sham orbitals.

3.7.5 Multiple k-points Calculations

SCF initialization

elsi_init(elsi_h, PEXSI, 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, energy)
    elsi_get_edm_{real|complex}(elsi_h, edm)

    Update electron density
    Check SCF convergence
end

elsi_finalize(elsi_h)

Comments

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 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.

To enable the ELSI C interface, include the following keyword in the make.sys file:

C_INTERFACE	= yes [default: no]

A Full List of Keywords in the make.sys File

A.1 General Flags

MPIFC	= MPI Fortran compiler [default: empty]
MPICC	= MPI C compiler [default: empty]
MPICXX	= MPI C++ compiler [default: empty]
LINKER	= Linker [default: $(MPIFC)]
FLINKER	= Linker for Fortran test programs [default: $(MPIFC)]
CLINKER	= Linker for C test programs [default: $(MPIFC)]
FFLAGS	= Fortran compiler flags [default: empty]
CFLAGS	= C compiler flags [default: empty]
CXXFLAGS	= C++ compiler flags [default: empty]
LDFLAGS	= Linker flags [default: empty]
FLDFLAGS	= Linker flags for Fortran test programs[default: empty]
CLDFLAGS	= Linker flags for C test programs[default: empty]
ARCHIVE	= Archiver [default: ar]
ARCHIVEFLAGS	= Archiver (ar) flags [default: cr]
RANLIB	= Archive index creator [default: ranlib]
CXX_LIB	= Standard C++ library, e.g. -lstdc++ [default: empty]
SCALAPACK_LIB	= BLAS, LAPACK, and ScaLAPACK libraries [default: empty]

A.2 ELSI Interface Flags

MPI_EXEC	= Name of MPI executable [default: mpirun]
FFLAGS_I	= Fortran compiler flags for ELSI Interface [default: $(FFLAGS)]
C_INTERFACE	= Create C interface? [default: no]

A.3 ELPA Flags

ELPA2_KERNEL	= Choice of ELPA2 kernels [default: Generic]
FFLAGS_E	= Fortran compiler flags for ELPA [default: $(FFLAGS)]
CFLAGS_E	= C compiler flags for ELPA [default: $(CFLAGS)]
EXTERNAL_ELPA	= Use external, precompiled ELPA? [default: no]
ELPA_LIB	= ELPA library flag(s) [default: built-in version]
ELPA_INC	= ELPA include flag(s) [default: built-in version]

A.4 libOMM Flags

FFLAGS_O	= Fortran compiler flags for libOMM [default: $(FFLAGS)]
EXTERNAL_OMM	= Use external, precompiled libOMM? [default: no]
OMM_LIB	= libOMM library flag(s) [default: built-in version]
OMM_INC	= libOMM include flag(s) [default: built-in version]

A.5 PEXSI Flags

CFLAGS_P	= C compiler flags for PEXSI [default: $(CFLAGS)]
CXXFLAGS_P	= C++ compiler flags for PEXSI [default: $(CXXFLAGS)]
PARMETIS_LIB	= METIS and ParMETIS library flags [default: empty]
PTSCOTCH_LIB	= Scotch and PtScotch library flags [default: empty]
SUPERLU_LIB	= SuperLU_Dist library flags [default: empty]
SUPERLU_INC	= SuperLU_Dist include flags [default: empty]
EXTERNAL_PEXSI	= Use external, precompiled PEXSI? [default: no]
PEXSI_LIB	= PEXSI library flag(s) [default: built-in version]
PEXSI_INC	= PEXSI include flag(s) [default: built-in version]

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] S. Mohr et al., Efficient computation of sparse matrix functions for large-scale electronic structure calculations: The CheSS library, Journal of Chemical Theory and Computation, 13, 4684-4698 (2017).

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

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

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

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

[13] S. Mohr et al., Accurate and efficient linear scaling DFT calculations with universal applicability, Physical Chemistry Chemical Physics, 17, 31360-31370 (2015).

[14] V. Yu et al., ELSI: A unified software interface for Kohn-Sham electronic structure solvers, Computer Physics Communications, http://dx.doi.org/10.1016/j.cpc.2017.09.007.

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 (LGPL), libOMM (2-clause BSD), and PEXSI (3-clause BSD) are redistributed within this ELSI release. Individual license of each library can be found in the corresponding subfolder.