



ELSI Interface Development version

User's Guide

The ELSI Team
<http://elsi-interchange.org>

January 2, 2020

Contents

1	Introduction	3
1.1	ELSI: ELectronic Structure Infrastructure	3
1.2	Solver Libraries Supported by ELSI	3
1.3	Citing ELSI	4
1.4	Acknowledgments	4
2	Installation of ELSI	5
2.1	Prerequisites	5
2.2	Quick Start	5
2.3	Configuration	6
2.3.1	Compilers	6
2.3.2	Solvers	6
2.3.3	Tests	7
2.3.4	List of All Configure Options	7
2.4	Importing ELSI into Third-Party Code Projects	8
2.4.1	Linking against ELSI: CMake	8
2.4.2	Linking against ELSI: Makefile	8
2.4.3	Using ELSI	9
3	The ELSI API	10
3.1	Overview of the ELSI API	10
3.2	Setting Up ELSI	10
3.2.1	Initializing ELSI	10
3.2.2	Setting Up MPI	12
3.2.3	Setting Up Matrix Formats	12
3.2.4	Setting Up Multiple k -points and/or Spin Channels	13
3.2.5	Reinitializaing ELSI	14
3.2.6	Finalizing ELSI	14
3.3	Solving Eigenvalues and Eigenvectors	14
3.4	Computing Density Matrices	15
3.5	Customizing ELSI	17

3.5.1	Customizing the ELSI Interface	17
3.5.2	Customizing the ELPA Solver	18
3.5.3	Customizing the libOMM Solver	19
3.5.4	Customizing the PEXSI Solver	20
3.5.5	Customizing the EigenExa Solver	22
3.5.6	Customizing the SLEPc-SIPs Solver	22
3.5.7	Customizing the NTPoly Solver	23
3.5.8	Customizing the MAGMA Solver	23
3.6	Getting Additional Results from ELSI	23
3.6.1	Getting Results from the ELSI Interface	23
3.6.2	Getting Results from the PEXSI Solver	25
3.6.3	Extrapolation of Wavefunctions and Density Matrices	25
3.7	Parallel Matrix I/O	27
3.7.1	Setting Up Matrix I/O	27
3.7.2	Writing Matrices	29
3.7.3	Reading Matrices	30
3.8	C/C++ Interface	31
3.9	Example Pseudo-Code	31

1 Introduction

1.1 ELSI: ELectronic Structure Infrastructure

Computer simulations based on electronic structure theory, particularly Kohn-Sham density-functional theory (KS-DFT), are facilitating scientific discoveries across a broad range of disciplines such as chemistry, physics, and materials science. Despite its remarkable success, routine application of KS-DFT to systems consisting of thousands of atoms is still difficult. The major computational bottleneck is an eigenvalue problem, whose solution scales cubically with respect to the problem size.

To overcome this bottleneck, researchers and software developers are actively improving the efficiency of eigensolvers and developing alternative algorithms that circumvent the explicit solution of the eigenproblem. The open-source ELSI library features a unified software interface that connects electronic structure codes to various high-performance solver libraries ranging from conventional cubic scaling eigensolvers to linear scaling density matrix solvers [1]. To date, it is adopted by four electronic structure packages (DFTB+ [2], DGDFT [3], FHI-aims [4], and SIESTA [5]).

1.2 Solver Libraries Supported by ELSI

Distributed-memory solvers supported in the current version of ELSI are: ELPA [6, 7, 8, 9], libOMM [10], PEXSI [11, 12, 13, 14, 15], EigenExa [16, 17], SLEPc-SIPc [18, 19, 20, 21], and NTPoly [22]. Shared-memory solvers supported in the current version of ELSI are: LAPACK [23] and MAGMA [24, 25].

What follows is a brief summary of the solvers supported in ELSI. For 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.

ELPA: The massively parallel dense eigensolver ELPA facilitates the solution of symmetric or Hermitian eigenproblems on high-performance computers. It features an efficient two-stage tridiagonalization algorithm which is better suited for parallel computing than the conventional one-stage algorithm.

libOMM: The orbital minimization method (OMM) bypasses the explicit solution of the Kohn-Sham eigenproblem by efficient iterative algorithms which directly minimize an unconstrained energy functional using a set of auxiliary Wannier functions. The Wannier functions are defined on the occupied subspace of the system, reducing the size of the problem. The density matrix is then obtained directly, without calculating the Kohn-Sham orbitals.

PEXSI: PEXSI is a Fermi operator expansion (FOE) based method which expands the density matrix in terms of a linear combination of a small number of rational functions (pole expansion). Evaluation of these rational functions exploits the sparsity of the Hamiltonian and overlap matrices using selected inversion to enable scaling to 100,000+ of MPI tasks for calculation of the electron density, energy, and forces in electronic structure calculations.

EigenExa: The EigenExa library consists of two massively parallel implementations of direct, dense eigensolver. Its `eigen_sx` method features an efficient transformation from full to pentadiagonal matrix. Eigenvalues and eigenvectors of the pentadiagonal matrix are directly solved with a divide-and-conquer algorithm. This method is particularly efficient when a large part of the eigenspectrum is of interest.

SLEPc-SIPs: SLEPc-SIPs is a parallel sparse eigensolver for real symmetric generalized eigenvalue problems. It implements a distributed spectrum slicing method and it is currently available through the SLEPc library built on top of the PETSc framework.

NTPoly: NTPoly is a massively parallel library for computing the functions of sparse, symmetric matrices based on

polynomial expansions. For sufficiently sparse matrices, most of the matrix functions can be computed in linear time. Distributed memory parallelization is based on a communication avoiding sparse matrix multiplication algorithm. Various density matrix purification algorithms which compute the density matrix as a function of the Hamiltonian matrix are implemented in NTPoly.

LAPACK: LAPACK provides routines for solving linear systems, least squares problems, eigenvalue problems, and singular value problems. In order to promote high efficiency on present-day computers, LAPACK routines are written to exploit BLAS, particularly level-3 BLAS, as much as possible. In ELSI, the tridiagonalization and the corresponding back-transformation routines in LAPACK are combined with the efficient divide-and-conquer tridiagonal solver in ELPA.

MAGMA: The MAGMA project aims to develop a dense linear algebra framework for heterogeneous architectures consisting of manycore and GPU systems. MAGMA incorporates the latest advances in synchronization-avoiding and communication-avoiding algorithms, and uses a hybridization methodology where algorithms are split into tasks of varying granularity and their execution scheduled over the available hardware components.

1.3 Citing ELSI

Key concepts of ELSI and the first version of its implementation are described in the following paper [1]:

V. W.-z. Yu, F. Corsetti, A. García, W. P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, Á. Vázquez-Mayagoitia, C. Yang, H. Yang, and V. Blum, ELSI: A Unified Software Interface for Kohn-Sham Electronic Structure Solvers, *Computer Physics Communications*, 222, 267-285 (2018).

In addition, an incomplete list of publications describing the solvers supported in ELSI may be found in the bibliography of this document. Please consider citing these articles when publishing results obtained with ELSI.

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 Prerequisites

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 1.2.0), and NTPoly (version 2.3.2). The installation of ELSI makes use of the CMake software. Minimum requirements include:

`CMake` [minimum version 3.0; newer version recommended]
`Fortran compiler` [Fortran 2003 compliant]
`C compiler` [C99 compliant]
`MPI` [MPI-3 recommended]

The PEXSi, EigenExa, SLEPc-SIPs, and MAGMA solvers are not enabled by default. Enabling the PEXSI solver requires:

`C++ compiler` [C++ 11 compliant]

Enabling the EigenExa solver requires:

`EigenExa` [version 2.3 or newer]

Enabling the SLEPc-SIPs solver requires:

`SLEPc` [version 3.9 or newer]
`PETSc` [version 3.9 or newer, with SuperLU_DIST, MUMPS, ParMETIS, and PT-SCOTCH enabled]

Enabling the MAGMA solver requires:

`MAGMA` [version 2.5 or newer]
`CUDA` [version 10.0 or newer]

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

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

By default, the redistributed versions of ELPA, libOMM, and NTPoly will be built. If PEXSI is enabled during configuration, the redistributed version of PEXSI and its dependencies, namely the SuperLU_DIST and PT-SCOTCH libraries, will be built as well. Optionally, the redistributed versions of ELPA, libOMM, PEXSI, SuperLU_DIST, PT-SCOTCH, and NTPoly may be substituted by user's optimized versions.

2.2 Quick Start

We recommend preparing configuration settings in a toolchain file that can be read by CMake. Edit one of the templates provided in the “toolchains” directory of the ELSI package. As an example, a minimal Intel toolchain looks like

```
# Modify contents in red if necessary
set(CMAKE_Fortran_COMPILER "mpiifort" CACHE STRING "MPI Fortran compiler")
set(CMAKE_Fortran_FLAGS "-O3 -ip -fp-model precise" CACHE STRING "Fortran flags")
set(LIB_PATHS "$ENV{MKLROOT}/lib/intel64" CACHE STRING "External library paths")
set(LIBS "mkl_scalapack_lp64 mkl_blacs_intelmpi_lp64 mkl_intel_lp64 mkl_sequential mkl_core"
  CACHE STRING "External libraries")
```

This will build ELSI with the redistributed ELPA, libOMM, and NTPoly solvers. A complete list of configure options may be found in 2.3.4.

Once a toolchain file is ready, follow the steps below:

```
$ cd elsi-interface
$ ls

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

$ mkdir build
$ cd build
$ cmake -DCMAKE_TOOLCHAIN_FILE=YOUR_TOOLCHAIN_FILE ..

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

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

“YOUR_TOOLCHAIN_FILE” should be the user’s toolchain file. Commands in square brackets are optional.

If the compilation succeeds, the next step would be reading the code examples in the “test” directory of the ELSI package, which showcase the use of ELSI in C and Fortran programs.

2.3 Configuration

2.3.1 Compilers

CMake automatically detects compilers. The choices made by CMake often work, but they do not necessarily lead to 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 in [CMAKE_Fortran_COMPILER](#), [CMAKE_C_COMPILER](#), and [CMAKE_CXX_COMPILER](#). Please note that the C++ compiler is not needed when building ELSI without PEXSI.

In addition, it is highly recommended to specify the compiler flags in [CMAKE_Fortran_FLAGS](#), [CMAKE_C_FLAGS](#), and [CMAKE_CXX_FLAGS](#).

2.3.2 Solvers

The ELPA, libOMM, PEXSI, and NTPoly 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), which may be chosen by the [ELPA2_KERNEL](#) keyword. Available options are [AVX](#), [AVX2](#), and [AVX512](#), for architectures supporting Intel AVX, AVX2, and AVX512 instruction sets, re-

spectively. In ELPA, these kernels are employed to accelerate the calculation of eigenvectors, which is often a bottleneck when calculating a large portion of the eigenspectrum.

The PEXSI, EigenExa, SLEPc-SIPs, and MAGMA solvers are not enabled by default. They may be activated by the keywords [ENABLE_PEXSI](#), [ENABLE_EIGENEXA](#), [ENABLE_SIPS](#), and [ENABLE_MAGMA](#), respectively. PEXSI 1.2.0, EigenExa 2.3, 2.4, SLEPc 3.9, 3.10, 3.11, 3.12, and MAGMA 2.5 have been tested with this version of ELSI. Older/newer versions may or may not be compatible. The PETSc library, required by SLEPc, must be compiled with MPI support, and (at least) with packages SuperLU_DIST, MUMPS, ParMETIS, and PT-SCOTCH enabled.

Experienced users are encouraged to link ELSI against external, better optimized solver libraries. The keywords [USE_EXTERNAL_ELPA](#), [USE_EXTERNAL_OMM](#), [USE_EXTERNAL_PEXSI](#), and [USE_EXTERNAL_NTPOLY](#) control the usage of externally compiled ELPA, libOMM, PEXSI, and NTPoly, respectively.

All external libraries and include paths should be set via [INC_PATHS](#), [LIB_PATHS](#), and [LIBS](#), each of which is a list of items separated with “ ” (space) or “;” (semicolon). If an external library depends on additional libraries, [LIBS](#) should include all the relevant dependencies. For instance, [LIBS](#) should include the MAGMA library and CUDA libraries when enabling support for MAGMA.

2.3.3 Tests

Building ELSI test programs may be enabled by [ENABLE_TESTS](#). Then, the compilation of ELSI may be verified by “make test” or “ctest”. Note that the tests may not run if launching MPI jobs is prohibited on the user’s working platform.

2.3.4 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_UNSCORE	boolean	ON	Suffix C functions with an underscore
BUILD_SHARED_LIBS	boolean	OFF	Build ELSI as a shared library
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_EIGENEXA	boolean	OFF	Enable EigenExa support
ENABLE_MAGMA	boolean	OFF	Enable MAGMA support
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
MPIEXEC_NP	string	mpirun -n 4	Command to run tests in parallel with MPI
MPIEXEC_1P	string	mpirun -n 1	Command to run tests in serial with MPI
SCOTCH_LAST_RESORT	string	none	Command to invoke PT-SCOTCH header generator
USE_EXTERNAL_ELPA	boolean	OFF	Use external ELPA
USE_EXTERNAL_NTPOLY	boolean	OFF	Use external NTPoly
USE_EXTERNAL_OMM	boolean	OFF	Use external libOMM and MatrixSwitch
USE_EXTERNAL_PEXSI	boolean	OFF	Use external PEXSI (if PEXSI enabled)
USE_MPI_ALLGATHER	boolean	ON	Use non-blocking collective MPI functions

Remarks

- 1) **ADD_UNDERSCORE**: In the redistributed PEXSI and SuperLU_DIST code, there are calls to functions from the linear algebra libraries, e.g. “dgemm”. If **ADD_UNDERSCORE** is “ON”, the code will call “dgemm_” instead of “dgemm”. Turn this keyword off if routines are not suffixed with “_” in the linear algebra libraries.
- 2) **CMAKE_INSTALL_PREFIX**: ELSI may be installed to the location specified in **CMAKE_INSTALL_PREFIX** by “make install”.
- 3) **ELPA2_KERNEL**: There are a number of computational kernels available with the ELPA solver. Choose from “AVX” (Intel AVX), “AVX2” (Intel AVX2), and “AVX512” (Intel AVX512). See 2.3.2 for more information.
- 4) **SCOTCH_LAST_RESORT**: The compilation of PT-SCOTCH is a multi-step process. First, two auxiliary executables are created. Then, some header files are generated on-the-fly by the two executables. Finally, the main source files are compiled with the generated header files included. The header generation step may fail on platforms where directly running an executable is prohibited, e.g. login nodes of a supercomputer. Often this can be circumvented by requesting an interactive session to a compute node and compiling the code there, or by submitting the compilation as a job to the queuing system. However, this may still fail on platforms where an executable compiled with MPI must be launched by an MPI job launcher (aprun, mpirun, srun, etc). If the standard compilation of PT-SCOTCH fails due to this reason, the user may set **SCOTCH_LAST_RESORT** to the command that starts an MPI job with one MPI task, e.g. “**mpirun -n 1**”. This command will be used to launch the auxiliary executables to generate necessary header files for PT-SCOTCH.
- 5) External libraries: ELSI redistributes source code of ELPA, libOMM, NTPoly, 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 2.3.2 for more information.
- 6) **USE_MPI_IALLGATHER**: NTPoly makes use of non-blocking collective MPI functions such as **MPI_Iallgatherv** to overlap its computation and communication. If these MPI functions are not available in the user’s MPI version, set **USE_MPI_IALLGATHER** to ‘OFF’. Using this flag may lead to reduced performance. Upgrade MPI if possible.

2.4 Importing ELSI into Third-Party Code Projects

2.4.1 Linking against ELSI: CMake

A CMake configuration file called **elsiConfig.cmake** should be generated after ELSI is successfully installed. This file contains all the information about how the ELSI library and its dependencies should be included in an external project. For a project using CMake, only two lines are required to find and link to ELSI:

```
find_package(elsi REQUIRED)
target_link_libraries(my_project PRIVATE elsi::elsi)
```

If a minimum version of ELSI is required, this information may be passed to “**find_package**” by, e.g.:

```
find_package(elsi 2.0 REQUIRED)
```

2.4.2 Linking against ELSI: Makefile

For a project using makefiles, an example set of compiler flags to link against ELSI would be:

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

Enabling/disabling PEXSI, EigenExa, SLEPc-SIPs, MAGMA or linking ELSI against pre-installed solver libraries will require the user modify these flags accordingly.

2.4.3 Using ELSI

ELSI may be used in an electronic structure 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 electronic structure code access to the ELSI interface, which we will describe in the next chapter.

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. While this chapter serves as a reference to the ELSI subroutines, the user is encouraged to explore the demonstration pseudo-codes of ELSI in 3.9.

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.

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

In the source code of ELSI, there may exist subroutines that are not documented as public API here. Usage of those undocumented subroutines is not recommended, as they are usually experimental and subject to modification or removal without notice.

3.2 Setting Up ELSI

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
<code>handle</code>	<code>type(elsi_handle)</code>	out	Handle to ELSI.
<code>solver</code>	integer	in	Desired solver. Accepted values are: 0 (AUTO), 1 (ELPA), 2 (libOMM), 3 (PEXSI), 4 (EigenExa), 5 (SLEPc-SIPs), 6 (NTPoly), and 7 (MAGMA). See remark 1.
<code>parallel_mode</code>	integer	in	Parallelization mode. Accepted values are: 0 (SINGLE_PROC) and 1 (MULTI_PROC). See remark 4.
<code>matrix_format</code>	integer	in	Matrix format. Accepted values are: 0 (BLACS_DENSE), 1 (PEXSI_CSC), 2 (SIESTA_CSC), and 3 (GENERIC_COO). See remark 2.
<code>n_basis</code>	integer	in	Number of basis functions, i.e. global size of Hamiltonian.
<code>n_electron</code>	real double	in	Number of electrons.
<code>n_state</code>	integer	in	Number of states. See remark 3.

Remarks

1) `solver`: The `AUTO` option attempts to automate the solver selection procedure based on benchmarks performed and experiences gained in the ELSI project. User-supplied information may assist in finding the optimal solver. In particular,

see [elsi_set_dimensionality](#) and [elsi_set_energy_gap](#) in 3.5. Simply put, the solver selection favors ELPA for small-and-medium-sized problems, PEXSI for large, sparse, low-dimensional problems, and NTPoly for extra-large, sparse systems with a decent energy gap. When [ELPA](#) is chosen for the [SINGLE_PROC](#) parallel mode, the tridiagonalization and back-transformation routines in LAPACK and the divide-and-conquer tridiagonal solver routine in ELPA will be used.

2) matrix.format: [BLACS_DENSE](#)(0) refers to 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. [GENERIC_COO](#)(3) refers to a coordinate (COO) sparse matrix format in an arbitrary distribution. Please refer to 3.2.3 for specifications of these matrix formats.

3) n.state: If ELPA, EigenExa, SLEPc-SIPs, or MAGMA is the chosen solver, this parameter specifies the number of eigenstates to solve. EigenExa internally computes all the eigenstates unless [n.state](#) is 0. When [n.state](#) is larger than 0 and smaller than [n.basis](#), ELSI simply discards the unwanted solutions. libOMM, PEXSI and NTPoly do not make use of this parameter.

4) parallel.mode: The two allowed values of [parallel.mode](#), 0 ([SINGLE_PROC](#)) and 1 ([MULTI_PROC](#)), allow for three parallelization strategies commonly employed by electronic structure codes. See below.

3a) SINGLE_PROC: Solves the KS eigenproblem following a LAPACK-like fashion. This option may only be selected when ELPA or MAGMA is chosen as the solver. Every MPI task independently handles a group of k -points uniquely assigned to it.

- Example: 16 k -points, 4 MPI tasks.
- 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.

```
call elsi_init (eh, ..., parallel_mode=0, ...)
...
do i_kpt = 1, n_kpt_local
  call elsi_ev_{real|complex} (eh, ham_this_kpt, ovlp_this_kpt, eval_this_kpt, evec_this_kpt)
end do
```

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

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

- Example: 4 k -points, 16 MPI tasks.
- 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.

```
call elsi_init (eh, ..., parallel_mode=1, ...)
call elsi_set_mpi (eh, my_mpi_comm)
call elsi_set_kpoint (eh, n_kpt, my_kpt, my_weight)
call elsi_set_mpi_global (eh, mpi_comm_global)
...
call elsi_{ev|dm}_{real|complex} (eh, my_ham, my_ovlp, ...)
```

Please note that when there is more than one k -point, a global MPI communicator must be provided for inter- k -point communications. See 3.2.4 for [elsi_set_kpoint](#), [elsi_set_spin](#), and [elsi_set_mpi_global](#), which are used to set up a calculation with two spin channels and/or multiple k -points.

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 solving one problem corresponding to one k -point and one spin channel. See 3.2.4 for details.

```
elsi_set_mpi(handle, mpi_comm)
```

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

3.2.3 Setting Up Matrix Formats

Four matrix formats are supported by ELSI, namely 2D block-cyclic distributed dense matrix format ([BLACS_DENSE](#)), 1D block distributed compressed sparse column format ([PEXSL_CSC](#)), 1D block-cyclic distributed compressed sparse column format, ([SIESTA_CSC](#)), arbitrarily distributed coordinate sparse format ([GENERIC_COO](#)).

When using the [BLACS_DENSE](#) format, 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 the [BLACS_DENSE](#) format.

```
elsi_set_blacs(handle, blacs_ctxt, block_size)
```

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
blacs_ctxt	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 [PEXSL_CSC](#) or [SIESTA_CSC](#) format, 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 CSC sparse matrix formats.

```
elsi_set_csc(handle, global_nnz, local_nnz, local_col, row_idx, col_ptr)
```

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
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.
row_idx	integer, rank-1 array	in	Local row index array. Dimension: local_nnz.
col_ptr	integer, rank-1 array	in	Local column pointer array. Dimension: local_col+1.

The block size of the [PEXSL_CSC](#) format cannot be set by the user. This is because the PEXSI solver requires that the block size must be $\text{floor}(\text{N_basis}/\text{N_procs})$, where $\text{floor}(x)$ is the greatest integer less than or equal to x , [N_basis](#) and [N_procs](#) are the number of basis functions and the number of MPI tasks, respectively. The block size of the [SIESTA_CSC](#) must be explicitly set by calling [elsi_set_csc_blk](#).

`elsi_set_csc_blk(handle, block_size)`

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

In most cases, input and output matrices should be distributed across all MPI tasks. The only exception is when using the PEXSI solver, the sparse density matrix interface `elsi_dm_{real|complex}_sparse`, and the `PEXSI_CSC` matrix format. In this case, an additional parameter, `pexsi_np_per_pole`, must be set by the user. Input and output matrices should be 1D-block-distributed among the first `pexsi_np_per_pole` MPI tasks (not all the MPI tasks). Please also read the 2nd remark in 3.5.4 for more information.

When using the `GENERIC_COO` format, the sparsity pattern should be passed into ELSI via the `elsi_set_coo` subroutine. It is necessary to call this subroutine before calling any solver interface that makes use of the COO sparse matrix format.

`elsi_set_coo(handle, global_nnz, local_nnz, row_idx, col_idx)`

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>global_nnz</code>	integer	in	Global number of non-zeros.
<code>local_nnz</code>	integer	in	Local number of non-zeros.
<code>row_idx</code>	integer, rank-1 array	in	Local row index array. Dimension: <code>local_nnz</code> .
<code>col_idx</code>	integer, rank-1 array	in	Local column index array. Dimension: <code>local_nnz</code> .

The distribution of matrix elements in the `GENERIC_COO` format is arbitrary. Both sorted and unsorted inputs are supported.

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 set up to support parallel calculation of the k -points and/or spin channels. The base case is a system isolated in space, e.g. free atoms, molecules, clusters, without spin-polarization. In this case, there is one eigenproblem in each iteration of an SCF cycle. When a spin-polarized periodic system is considered, there is an index α denoting the spin channel, and an index k denoting points in reciprocal space. In total, there are $N_{\text{kpt}} \times N_{\text{spin}}$ eigenproblems that can be solved in an embarrassingly parallel fashion. In ELSI, these eigenproblems are considered as equivalent “unit tasks”. The available computer processes are divided into $N_{\text{kpt}} \times N_{\text{spin}}$ groups, each of which is responsible for one unit task.

To set up the ELSI interface for a calculation with more than one k -point and/or more than one spin channel, the `elsi_set_kpoint` and/or `elsi_set_spin` subroutines are called to pass the required information into ELSI. The MPI communicator for each unit task is passed into ELSI by calling `elsi_set_mpi`. In addition, a global MPI communicator for all tasks is passed into ELSI by calling `elsi_set_mpi_global`. 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 handling more than one k -point and/or more than one spin channel. In ELSI, the two spin channels are always coupled by a uniform chemical potential. The distribution of electrons among the two channels, and thus the net spin moment of the system, cannot be specified. Calculations with a fixed, user-specified spin moment can be performed by initializing two independent ELSI instances for the two spin channels.

In this version of ELSI, the SLEPc-SIPs eigensolver is not supported in spin-polarized and/or periodic calculations.

`elsi_set_kpoint(handle, n_kpt, i_kpt, weight)`

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>n_kpt</code>	integer	in	Total number of k -points.
<code>i_kpt</code>	integer	in	Index of the k -point handled by this MPI task.
<code>weight</code>	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
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>n_spin</code>	integer	in	Total number of spin channels.
<code>i_spin</code>	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
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>mpi_comm_global</code>	integer	in	Global MPI communicator used for communications among all k -points and spin channels.

3.2.5 Reinitializaing ELSI

When a geometry update takes place in geometry optimization or molecular dynamics calculations, the overlap matrix changes due to the movement of localized basis functions. Calling `elsi_reinit` instructs ELSI to flush geometry-related variables and arrays that cannot be used in the new geometry step, e.g., the overlap matrix and its sparsity pattern. Other runtime parameters are kept within the ELSI instance and reused throughout multiple geometry steps. Note that the chemical potential determination in PEXSI must be restarted for every new geometry. See the 5th remark in 3.5.4 for details.

`elsi_reinit(handle)`

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.

3.2.6 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
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.

3.3 Solving Eigenvalues and Eigenvectors

The following subroutines return all the eigenvalues and a subset of eigenvectors of the provided generalized eigenproblem defined by H and S matrices. For standard eigenproblems, please see `elsi_set_unit_ovlp` in 3.5.1. ELPA, EigenExa, SLEPc-SIPs, or MAGMA may be selected as the solver when using these subroutines.

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

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	real double, rank-2 array	inout	Real Hamiltonian matrix in 2D block-cyclic dense format. See remark 1.
<code>ovlp</code>	real double, rank-2 array	inout	Real overlap matrix (or its Cholesky factorization) in 2D block-cyclic dense format. See remark 1.
<code>eval</code>	real double, rank-1 array	inout	Eigenvalues. See remark 2.
<code>evec</code>	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
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	complex double, rank-2 array	inout	Complex Hamiltonian matrix in 2D block-cyclic dense format. See remark 1.
<code>ovlp</code>	complex double, rank-2 array	inout	Complex overlap matrix (or its Cholesky factorization) in 2D block-cyclic dense format. See remark 1.
<code>eval</code>	real double, rank-1 array	inout	Eigenvalues. See remark 2.
<code>evec</code>	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
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	real double, rank-1 array	inout	Real Hamiltonian matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>ovlp</code>	real double, rank-1 array	inout	Real overlap matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>eval</code>	real double, rank-1 array	inout	Eigenvalues. See remark 2.
<code>evec</code>	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
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	complex double, rank-1 array	inout	Complex Hamiltonian matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>ovlp</code>	complex double, rank-1 array	inout	Complex overlap matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>eval</code>	real double, rank-1 array	inout	Eigenvalues. See remark 2.
<code>evec</code>	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 during computation. When using `elsi_ev_{real|complex}`, the overlap matrix will be overridden by its Cholesky factorization. When using `elsi_ev_{real|complex}_sparse`, the Cholesky factorization (not sparse) is stored internally in the BLACS_DENSE format.
- 2) The dimension of `eval` should always be `n_basis`, regardless of the choice of `n_state` specified in `elsi_init`.
- 3) `elsi_ev_{real|complex}_{sparse}` computes 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 work space. When using `elsi_ev_{real|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 band structure energy.

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

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	real double, rank-2 array	inout	Real Hamiltonian matrix in 2D block-cyclic dense format.
<code>ovlp</code>	real double, rank-2 array	inout	Real overlap matrix (or Cholesky factorization) in 2D block-cyclic dense format. See remark 1.
<code>dm</code>	real double, rank-2 array	out	Real density matrix in 2D block-cyclic dense format.
<code>energy</code>	real double	out	Band structure energy.

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

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	complex double, rank-2 array	inout	Complex Hamiltonian matrix in 2D block-cyclic dense format.
<code>ovlp</code>	complex double, rank-2 array	inout	Complex overlap matrix (or its Cholesky factorization) in 2D block-cyclic dense format. See remark 1.
<code>dm</code>	complex double, rank-2 array	out	Complex density matrix in 2D block-cyclic dense format.
<code>energy</code>	real double	out	Band structure energy.

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

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	real double, rank-1 array	inout	Non-zero values of the real Hamiltonian matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>ovlp</code>	real double, rank-1 array	inout	Non-zero values of the real overlap matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>dm</code>	real double, rank-1 array	out	Non-zero values of the real density matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>energy</code>	real double	out	Band structure energy.

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

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_handle)</code>	inout	Handle to ELSI.
<code>ham</code>	complex double, rank-1 array	inout	Non-zero values of the complex Hamiltonian matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>ovlp</code>	complex double, rank-1 array	inout	Non-zero values of the complex overlap matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>dm</code>	complex double, rank-1 array	out	Non-zero values of the complex density matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
<code>energy</code>	real double	out	Band structure energy.

Remarks

1) When using `elsi_dm_{real|complex}` with ELPA, libOMM, or EigenExa, the Hamiltonian matrix will be destroyed during the computation. The overlap matrix will be used to store its Cholesky factorization, which will be reused until the overlap matrix changes.

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 ELSI API. Integers are used to represent logical, with 0 being false and any positive integer being true.

```
elsi_set_output(handle, output_level)
elsi_set_output_unit(handle, output_unit)
elsi_set_output_log(handle, output_log)
elsi_set_save_ovlp(handle, save_ovlp)
elsi_set_unit_ovlp(handle, unit_ovlp)
elsi_set_zero_def(handle, zero_def)
elsi_set_illcond_check(handle, illcond_check)
elsi_set_illcond_tol(handle, illcond_tol)
elsi_set_spin_degeneracy(handle, spin_degeneracy)
elsi_set_energy_gap(handle, energy_gap)
elsi_set_spectrum_width(handle, spectrum_width)
elsi_set_dimensionality(handle, dimensionality)
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)
elsi_set_mu_mp_order(handle, mu_mp_order)
elsi_set_write_unit(handle, write_unit)
elsi_set_sing_check(handle, sing_check)
elsi_set_sing_tol(handle, sing_tol)
```

Argument	Data Type	Default	Explanation
output_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.
output_unit	integer	6	Unit used in ELSI to write out information.
output_log	integer	0	If not 0, a separate log file in JSON format will be written out.
save_ovlp	integer	0	If not 0, the overlap matrix will be saved for extrapolation of density matrix or eigenvectors to a new geometry.
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.
illcond_check	integer	0	If not 0, the eigenvalues of the overlap matrix will be calculated in order to check if it is ill-conditioned. See remark 2.
illcond_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 2.
spin_degeneracy	real double	$2.0/n_{\text{spin}}$	Spin degeneracy that controls the maximum number of electrons on a state.
energy_gap	real double	0	Energy gap. See remark 3.
spectrum_width	real double	10^3	Width of the eigenspectrum. See remark 3.
dimensionality	integer	3	Dimensionality (1, 2, or 3) of the simulating system. Only used for automatic solver selection.
mu_broaden_scheme	integer	0	Broadening scheme employed to compute the occupation numbers and the Fermi level. 0: Gaussian. 1: Fermi-Dirac. 2: Methfessel-Paxton. 4: Marzari-Vanderbilt. See remark 4.
mu_broaden_width	real double	0.01	Broadening width employed to compute the occupation numbers and the Fermi level. See remark 5.
mu_tol	real double	10^{-13}	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.
mu_mp_order	integer	0	Order of the Methfessel-Paxton broadening scheme. No effect if Methfessel-Paxton is not the chosen broadening scheme.
write_unit	integer	6	Deprecated. Use elsi_set_output_unit instead.
sing_check	integer	0	Deprecated. Use elsi_set_illcond_check instead.
sing_tol	real double	10^{-5}	Deprecated. Use elsi_set_illcond_tol instead.

Remarks

- 1) If the overlap matrix is set to be an identity matrix, all settings related to the singularity (ill-conditioning) check take no effect. The [ovlp](#) argument passed into [elsi_{ev|dm}_{real|complex}_{sparse}](#) will not be referenced.
- 2) If the ill-conditioning check is not disabled, in the first iteration of each SCF cycle, all eigenvalues of the overlap matrix is computed. If there is any eigenvalue smaller than [illcond_tol](#), the matrix is considered to be ill-conditioned.
- 3) [spectrum_width](#) and [energy_gap](#) refer to the width and the gap of the eigenspectrum. Simply use the default values if there is no better estimate.
- 4) [mu_broaden_scheme](#), [mu_broaden_width](#), and [mu_tol](#) are only referenced when using [elsi_dm_{real|complex}_{sparse}](#) and an eigensolver. They are ignored when using [elsi_ev_{real|complex}_{sparse}](#), or [elsi_dm_{real|complex}_{sparse}](#) with a density matrix solver.
- 5) In all supported broadening schemes, there is a term $(\epsilon - E_F)/W$ 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 W in the same unit of ϵ and E_F .

3.5.2 Customizing the ELPA Solver

[elsi_set_elpa_solver](#)(handle, elpa_solver)

[elsi_set_elpa_n_single](#)(handle, elpa_n_single)

`elsi_set_elpa_gpu(handle, elpa_gpu)`

`elsi_set_elpa_autotune(handle, elpa_autotune)`

`elsi_set_elpa_gpu_kernels(handle, elpa_gpu_kernels)`

Argument	Data Type	Default	Explanation
<code>elpa_solver</code>	integer	2	1: ELPA 1-stage solver. 2: ELPA 2-stage solver. The latter is usually faster and more scalable.
<code>elpa_n_single</code>	integer	0	Number of SCF steps using single precision ELPA to solve standard eigenproblems. See remark 1.
<code>elpa_gpu</code>	integer	0	If not 0, try to enable GPU-acceleration in ELPA. See remark 2.
<code>elpa_autotune</code>	integer	1	If not 0, try to enable auto-tuning of runtime parameters in ELPA. See remark 3.
<code>elpa_gpu_kernels</code>	integer	0	Deprecated. No effect.

Remarks

1) `elpa_n_single`: If single precision arithmetic is available in an externally compiled ELPA library, it may be enabled by setting `elpa_n_single` to a positive integer, then the standard eigenproblems in the first `elpa_n_single` SCF steps will be solved with single precision. The transformations between generalized eigenproblem and the standard form are always performed with double precision. Although this keyword accelerates the solution of standard eigenproblems, the overall SCF convergence may be slower, depending on the physical system and the SCF settings used in the electronic structure code. This keyword is ignored if single precision calculations are not available, which is the case if the internal version of ELPA is used, or if an external ELPA has not been compiled with single precision support.

2) `elpa_gpu`: If GPU-acceleration is available in an externally compiled ELPA library, it may be enabled by setting `elpa_gpu` to a nonzero integer. This keyword is ignored 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 compiled with GPU support.

3) `elpa_autotune`: If auto-tuning of runtime parameters is available in an externally compiled ELPA library, it may be enabled by setting `elpa_autotune` to a nonzero integer. This keyword is ignored if auto-tuning is not available, which is the case if the internal version of ELPA is used.

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
<code>omm_flavor</code>	integer	0	Method to perform OMM minimization. See remark 1.
<code>omm_n_elpa</code>	integer	6	Number of SCF steps using ELPA. See remark 2.
<code>omm_tol</code>	real double	10^{-12}	Convergence tolerance of orbital minimization. See remark 3.

Remarks

1) `omm_flavor`: Allowed choices are 0 for a 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 (Cholesky) 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 (basic) 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_method(handle, pexsi_method)
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_temp(handle, pexsi_temp)
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)
elsi_set_pexsi_gap(handle, pexsi_gap)
elsi_set_pexsi_delta_e(handle, pexsi_delta_e)
```

Argument	Data Type	Default	Explanation
<code>pexsi_method</code>	integer	2	Pole expansion method used by PEXSI. 1: Contour integral. 2: Minimax rational approximation. See remark 1.
<code>pexsi_n_pole</code>	integer	20	Number of poles used by PEXSI. See remark 1.
<code>pexsi_n_mu</code>	integer	2	Number of mu points used by PEXSI. See remark 2.
<code>pexsi_np_per_pole</code>	integer	-	Number of MPI tasks assigned to each mu point. See remark 3.
<code>pexsi_np_symbo</code>	integer	1	Number of MPI tasks for symbolic factorization. See remark 4.
<code>pexsi_temp</code>	real double	0.002	Temperature. See remark 5.
<code>pexsi_mu_min</code>	real double	-10.0	Minimum value of mu. See remark 6.
<code>pexsi_mu_max</code>	real double	10.0	Maximum value of mu. See remark 6.
<code>pexsi_inertia_tol</code>	real double	0.05	Stopping criterion of inertia counting. See remark 6.
<code>pexsi_gap</code>	real double	0.0	Deprecated. Use <code>elsi_set_energy_gap</code> instead. See remark 7.
<code>pexsi_delta_e</code>	real double	10.0	Deprecated. Use <code>elsi_set_spectrum_width</code> instead.

Remarks

1) When using the pole expansion method based on contour integral, 60 to 100 poles are typically needed to get an accuracy that is comparable with the result obtained from diagonalization. Perform a convergence test if necessary. The electronic entropy is available with this method, and may be accessed via `elsi_get_entropy`.

When using the pole expansion method based on minimax rational approximation, 20 poles are recommended. PEXSI outputs an error message when it detects an unreasonable choice of number of poles. The electronic entropy cannot be computed with this method.

2) PEXSI determines the chemical potential by performing Fermi operator expansion at several chemical potential values (referred to as “points”) 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. Two points followed by a simple linear interpolation often yield reasonable results.

3) `pexsi_np_per_pole`: PEXSI has, by construction, a 3-level parallelism: 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` \times `pexsi_n_mu` \times `pexsi_n_pole`, i.e., all the three levels

of parallelism are fully exploited. In case that this is not feasible, PEXSI can also process the poles in serial, whereas all the chemical potential points must be evaluated simultaneously. The user should make sure that the total number of MPI tasks is divisible by the product of the number of MPI tasks per pole and the number of points. The code will stop if this requirement is not fulfilled.

When using the `BLACS_DENSE` or `SIESTA_CSC` matrix formats, `pexsi_np_per_pole` is automatically determined to balance the three levels of parallelism in PEXSI. Input and output matrices should be distributed across all MPI tasks in either a 2D block-cyclic distribution (`BLACS_DENSE`) or a 1D block-cyclic distribution (`SIESTA_CSC`).

Please note that when using the `PEXSI_CSC` matrix format together with the PEXSI solver, input and output matrices should be distributed among the first `pexsi_np_per_pole` MPI tasks (not all the MPI tasks) in a 1D block distribution. The block size of the distribution must be $\text{floor}(\text{N_basis}/\text{N_procs})$, where $\text{floor}(x)$ is the greatest integer less than or equal to x , `N_basis` and `N_procs` are the number of basis functions and the number of MPI tasks, respectively.

when using the `PEXSI_CSC` matrix format with the ELPA, libOMM, EigenExa, SLEPc-SIPs, or NTPoly solver, input and output matrices should be distributed across all the MPI tasks in a 1D block distribution. Again, the block size of the distribution must be $\text{floor}(\text{N_basis}/\text{N_procs})$.

4) `pexsi_np_symbolo`: Unless there is a memory bottleneck, using 1 MPI task for matrix reordering and symbolic factorization is favorable. When running in serial, the matrix reordering in PT-SCOTCH or ParMETIS introduces a minimal number of “fill-ins” to the factorized matrices. Using more MPI tasks introduces more fill-ins. As the matrix reordering and symbolic factorization are performed only once per SCF cycle (with a fixed overlap matrix), using 1 MPI task should not affect the overall timing too much. On the other hand, more fill-ins lead to slower numerical factorization in every SCF step. In addition, the number of MPI tasks used for matrix reordering and symbolic factorization cannot be too large. Otherwise, the symbolic factorization may fail. Therefore, the default number of MPI tasks for symbolic factorization is 1. It is worth testing and increasing this number for large-scale calculations.

5) `pexsi_temp`: This value corresponds to the $1/k_B T$ term (not T) in the Fermi-Dirac distribution function.

```
do geometry update
  mu_min = -10.0
  mu_max = 10.0
  delta_V_min = 0.0
  delta_V_max = 0.0

  do SCF cycle
    Update Hamiltonian

    call elsi_set_pexsi_mu_min (eh, mu_min + delta_V_min)
    call elsi_set_pexsi_mu_max (eh, mu_max + delta_V_max)

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

    call elsi_get_pexsi_mu_min (eh, mu_min)
    call elsi_get_pexsi_mu_max (eh, mu_max)

    Update electron density
    Update potential

    delta_V_min = minval (V_new - V_old)
    delta_V_max = maxval (V_new - V_old)

    Check SCF convergence
  end do
end do
```

6) 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 of each geometry step, 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 $(\mu_{\text{min}}^{n-1} + \Delta V_{\text{min}})$, `pexsi.mu_max` should be set to $(\mu_{\text{max}}^{n-1} + \Delta V_{\text{max}})$. Here, μ_{min}^{n-1} and μ_{max}^{n-1} are the lower bound and the upper bound of the chemical potential that are determined by PEXSI in the $(n-1)^{\text{th}}$ SCF step. They can be retrieved by calling `elsi.get_pexsi.mu_min` and `elsi.get_pexsi.mu_max`, respectively (see 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 pseudo-code above. The (re-)initialization and finalization of ELSI are omitted.

7) `pexsi_gap`: The PEXSI method does not require an energy gap. If no knowledge is available, the default value usually works.

3.5.5 Customizing the EigenExa Solver

`elsi.set_eigenexa.method`(handle, eigenexa_method)

Argument	Data Type	Default	Explanation
<code>eigenexa.method</code>	integer	2	1: Tridiagonalization solver <code>eigen.s</code> . 2: Pentadiagonalization solver <code>eigen.sx</code> . The latter is usually faster and more scalable.

3.5.6 Customizing the SLEPc-SIPs Solver

`elsi.set_sips.ev_min`(handle, sips_ev_min)

`elsi.set_sips.ev_max`(handle, sips_ev_max)

`elsi.set_sips.n_elpa`(handle, sips_n_elpa)

`elsi.set_sips.n_slice`(handle, sips_n_slice)

`elsi.set_sips.interval`(handle, sips_lower, sips_upper)

Argument	Data Type	Default	Explanation
<code>sips.ev_min</code>	real double	-2.0	Lower bound of eigenspectrum. See remark 1.
<code>sips.ev_max</code>	real double	2.0	Upper bound of eigenspectrum. See remark 1.
<code>sips.n_elpa</code>	integer	0	Number of SCF steps using ELPA. See remark 2.
<code>sips.n_slice</code>	integer	1	Number of slices. See remark 3.
<code>sips_lower</code>	real double	-2.0	Deprecated. Use <code>elsi.set_sips.ev_min</code> instead.
<code>sips_upper</code>	real double	2.0	Deprecated. Use <code>elsi.set_sips.ev_max</code> instead.

Remarks

1) `sips.ev_min` and `sips.ev_max`: 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 `ev_min` and `ev_max`. Depending on the results of inertia counting, this interval may expand or shrink to make sure that the 1st to the `n.state`th eigenvalues are all within this interval. If a good estimate of the lower or upper bounds of the eigenspectrum is available, it should be set by `elsi.set_sips.ev_min` or `elsi.set_sips.ev_max`.

2) `sips.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 `sips.n_elpa` SCF steps can circumvent the load imbalance of spectrum slicing in the initial SCF steps.

3) `sips_n_slice`: SLEPc-SIPs partitions the eigenspectrum into slices and solves the slices in parallel. The `sips_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 be divisible by `sips_n_slice`. Secondly, setting `sips_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 `sips_n_slice` depends on the actual problem as well as the computing hardware.

3.5.7 Customizing the NTPoly Solver

`elsi_set_ntpoly_method`(handle, ntpoly_method)

`elsi_set_ntpoly_filter`(handle, ntpoly_filter)

`elsi_set_ntpoly_tol`(handle, ntpoly_tol)

Argument	Data Type	Default	Explanation
<code>ntpoly_method</code>	integer	2	Method to perform density matrix purification. See remark 1.
<code>ntpoly_filter</code>	real double	10^{-15}	When performing sparse matrix multiplications, values below this filter will be discarded. See remark 2.
<code>ntpoly_tol</code>	real double	10^{-8}	Convergence tolerance of purification. See remark 2.

Remarks

1) `ntpoly_method`: Allowed choices are 0 for the canonical purification [26], 1 for the trace correcting purification [27], 2 for the 4th order trace resetting purification [27], and 3 for the generalized hole-particle canonical purification [28].

2) `ntpoly_filter` and `ntpoly_tol` control the accuracy and computational cost of a density matrix purification method. Tight choices of `ntpoly_filter` and `ntpoly_tol`, e.g. the default values here, lead to highly accurate results that are comparable to the results obtained from diagonalization. However, linear scaling can only be achieved with a relatively large `ntpoly_filter` such as 10^{-6} . Note that the purification may not converge if `ntpoly_filter` is too large relative to `ntpoly_tol`. Setting `ntpoly_filter` to be $\leq 10^{-3} \times \text{ntpoly_tol}$ is safe in most cases.

3.5.8 Customizing the MAGMA Solver

`elsi_set_magma_solver`(handle, magma_solver)

Argument	Data Type	Default	Explanation
<code>magma_solver</code>	integer	1	1: MAGMA 1-stage eigensolver. 2: MAGMA 2-stage eigensolver.

Remarks

1) MAGMA is enable to make use of multiple GPUs. The number of GPUs is controlled by the environment variable `MAGMA_NUM_GPUS`. Refer to the users' guide of MAGMA for more information.

3.6 Getting Additional Results from ELSI

In 3.3 and 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 a 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 the parameter to get.

[elsi_get_initialized](#)(handle, handle_init)
[elsi_get_version](#)(handle, major, minor, patch)
[elsi_get_datestamp](#)(handle, date_stamp)
[elsi_get_n_illcond](#)(handle, n_illcond)
[elsi_get_ovlp_ev_min](#)(handle, ev_min)
[elsi_get_ovlp_ev_max](#)(handle, ev_max)
[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)
[elsi_get_eval](#)(handle, eval)
[elsi_get_evec_real](#)(handle, evec_real)
[elsi_get_evec_complex](#)(handle, evec_complex)
[elsi_get_occ](#)(handle, occ)
[elsi_get_n_sing](#)(handle, n_sing)

Argument	Data Type	Explanation
handle_init	integer	0 if the ELSI handle has not been initialized; 1 if initialized.
major	integer	Major version number.
minor	integer	Minor version number.
patch	integer	Patch level.
date_stamp	integer	Date stamp of ELSI (yyyymmdd).
n_illcond	integer	Number of eigenvalues of the overlap matrix that are smaller than the ill-conditioning tolerance. See 3.5.1.
ev_min	real double	Lowest eigenvalue of the overlap matrix. See remark 1.
ev_max	real double	Highest eigenvalue of the overlap matrix. See remark 1.
mu	real double	Chemical potential. See remark 2.
ts	real double	Entropy. See remark 2.
edm_real	real double, rank-2 array	Real energy-weighted density matrix in 2D block-cyclic dense format. See remark 3.
edm_complex	complex double, rank-2 array	Complex energy-weighted density matrix in 2D block-cyclic dense format. See remark 3.
edm_real_sparse	real double, rank-1 array	Non-zero values of the real density matrix in 1D block CSC format. See remark 3.
edm_complex_sparse	complex double, rank-1 array	Non-zero values of the complex density matrix in 1D block CSC format. See remark 3.
eval	real double, rank-1 array	Eigenvalues. See remark 4.
evec_real	real double, rank-2 array	Real eigenvectors in 2D block-cyclic dense format. See remark 4.
evec_complex	complex double, rank-2 array	Complex eigenvectors in 2D block-cyclic dense format. See remark 4.
occ	real double, rank-1 array	Occupation numbers. See remark 4.
n_sing	integer	Deprecated. Use elsi_get_n_illcond instead.

Remarks

- 1) In ELSI, ill-conditioning check of the overlap matrix is enabled by default when ELPA is the chosen solver. It may be disabled by calling `elsi_set_illcond_check`, and is automatically disabled when the chosen solver is not ELPA. `ev_min` and `ev_max` are computed only if ill-conditioning check is enabled. Otherwise the return value may be zero.
- 2) In ELSI, the chemical potential will only be available if one of the density matrix solver interfaces has been called, with ELPA, PEXSI, EigenExa, or NTPoly 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, PEXSI (see also `elsi_set_pexsi_method`), or EigenExa being the chosen solver. ELSI may return zero when the chemical potential or the entropy is not ready.
- 3) 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. These subroutines 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.
- 4) When using `elsi_dm_{real|complex}_{sparse}` with an eigensolver, ELSI internally computes and stores the eigenvalues, eigenvectors, and occupation numbers, which are the ingredients to assemble the density matrix. These quantities may be retrieved by calling `elsi_get_eval`, `elsi_get_evec_{real|complex}`, and `elsi_get_occ`. The dimension of `eval` and `occ` should be equal to the value of `n_states` set in `elsi_init`. Even with `elsi_dm_{real|complex}_sparse`, the eigenvectors are returned in a dense format (`BLACS_DENSE`), as they are in general not sparse. The size of `evec_{real|complex}` should always correspond to a global array of size `n_basis` \times `n_basis`, regardless of the value of `n_states`.

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
<code>pexsi_mu_min</code>	real double	Minimum value of mu. See remark 1.
<code>pexsi_mu_max</code>	real double	Maximum value of mu. See remark 1.

Remarks

- 1) Please refer to the 5th remark in 3.5.4 for the chemical potential determination algorithm in PEXSI and ELSI.

3.6.3 Extrapolation of Wavefunctions and Density Matrices

In a single point total energy calculation, a simple way to construct an initial guess of the electron density is using the superposition of free atom densities. In geometry calculations, the initial guess in the (n+1)th geometry step can be made better than free atom superposition, by reusing the wavefunctions or density matrix calculated in the nth geometry step. However, due to the movement of atoms and localized basis functions around them, wavefunctions obtained in the nth geometry step are no longer orthonormalized in the (n+1)th geometry step. The following subroutines orthonormalize eigenvectors (coefficients of wavefunctions) in the nth geometry step with respect to the overlap matrix in the (n+1)th geometry step with a Gram-Schmidt algorithm.

`elsi_orthonormalize_ev_real`(handle, ovlp, evec)

Argument	Data Type	in/out	Explanation
<code>handle</code>	type(elsi_handle)	inout	Handle to ELSI.
<code>ovlp</code>	real double, rank-2 array	in	Real overlap matrix in 2D block-cyclic dense format.
<code>evec</code>	real double, rank-2 array	inout	Real eigenvectors in 2D block-cyclic dense format.

[elsi_orthonormalize_ev_complex](#)(handle, ovlp, evec)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
ovlp	complex double, rank-2 array	in	Complex overlap matrix in 2D block-cyclic dense format.
evec	complex double, rank-2 array	inout	Complex eigenvectors in 2D block-cyclic dense format.

[elsi_orthonormalize_ev_real_sparse](#)(handle, ovlp, evec)

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

[elsi_orthonormalize_ev_complex_sparse](#)(handle, ovlp, evec)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
ovlp	complex double, rank-1 array	in	Complex overlap matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format.
evec	complex double, rank-2 array	inout	Complex eigenvectors in 2D block-cyclic dense format. See remark 1.

Remarks

1) When using [elsi_orthonormalize_ev_{real|complex}_sparse](#), the eigenvectors are stored in a dense format ([BLACS_DENSE](#)), as they are in general not sparse.

The following subroutines extrapolate density matrix in the n^{th} geometry step to the overlap matrix in the $(n+1)^{\text{th}}$ geometry step.

[elsi_extrapolate_dm_real](#)(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
ovlp	real double, rank-2 array	inout	Real overlap matrix in 2D block-cyclic dense format. See remark 1.
dm	real double, rank-2 array	inout	Real density matrix in 2D block-cyclic dense format. See remark 2.

[elsi_extrapolate_dm_complex](#)(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
ovlp	complex double, rank-2 array	inout	Complex overlap matrix in 2D block-cyclic dense format. See remark 1.
dm	complex double, rank-2 array	inout	Complex density matrix in 2D block-cyclic dense format. See remark 2.

[elsi_extrapolate_dm_real_sparse](#)(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
ovlp	real double, rank-1 array	inout	Real overlap matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format. See remark 1.
dm	real double, rank-1 array	out	Real density matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format. See remark 3.

[elsi_extrapolate_dm_complex_sparse](#)(handle, ovlp, dm)

Argument	Data Type	in/out	Explanation
handle	type(elsi_handle)	inout	Handle to ELSI.
ovlp	complex double, rank-1 array	inout	Complex overlap matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format. See remark 1.
dm	complex double, rank-1 array	out	Complex density matrix in 1D block CSC, 1D block-cyclic CSC, or generic COO sparse format. See remark 3.

Remarks

- 1) [ovlp](#): This should be the overlap matrix in the $(n+1)^{\text{th}}$ geometry step. [elsi_set_save_ovlp](#) must have been called to store the overlap matrix in the n^{th} geometry step internally. Depending on the chosen solver, [ovlp](#) may be overridden by its Cholesky factorization, which will be reused by subsequent calls to the solver interfaces.
- 2) [dm](#): Input should be the density matrix in the n^{th} geometry step. Output is the extrapolated density matrix.
- 3) [dm](#): With the sparse density matrix extrapolation interface, the density matrix in the n^{th} geometry step is stored internally. Output is the extrapolated density matrix.

3.7 Parallel Matrix I/O

The ELSI interface is able to read and write distributed matrices in parallel. There exist a number of libraries for high-performance parallel I/O that are particularly capable of reading and writing a large amount of data with hierarchical structures and complex metadata. However, the data structure in ELSI is simply arrays that represent matrices, with a few integers to define the dimension of the matrices. In order to circumvent the development and performance overhead associated with a high level I/O library, ELSI directly uses the parallel I/O functionality defined in the MPI standard.

Writing the distributed matrices into N_{procs} separate files, where N_{procs} is the number of MPI tasks, is not promising, as manipulating a large number of files is usually difficult. The implementation of matrix I/O in ELSI adopts collective MPI I/O routines to write data to (read data from) a single binary file, as if the data was gathered onto a single MPI task then written to one file (read from one file by one MPI task then scattered to all tasks). The optimal I/O performance, both with MPI I/O and in general, is achieved by making large and contiguous requests to access the file system. Therefore, ELSI always redistributes the matrices to a 1D block distribution before writing it to file. This guarantees that each MPI task writes a contiguous chunk of data to a contiguous piece of file. Similarly, matrices read from file are in a 1D block distribution, and can be redistributed automatically if needed. ELSI always stores matrices in a sparse CSC format. The conversion between dense and sparse formats is handled automatically.

3.7.1 Setting Up Matrix I/O

An `elsi_rw_handle` must be initialized via the [elsi_init_rw](#) subroutine before any other matrix I/O subroutine may be called. This `elsi_rw_handle` is subsequently passed to all other matrix I/O subroutine calls.

[elsi_init_rw](#)(handle, task, parallel_mode, n_basis, n_electron)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	out	Handle to matrix I/O instance.
task	integer	in	Matrix I/O task to perform. Accepted values are: 0 (READ_MATRIX) and 1(WRITE_MATRIX).
parallel_mode	integer	in	Parallelization mode. The only accepted value is 1 (MULTIPROC) for now.
n_electron	real double	in	Number of electrons. See remark 1.
n_basis	integer	in	Number of basis functions, i.e. global size of matrix.

Remarks

1) [n_electron](#): Matrices written out with ELSI matrix I/O are usually from actual electronic structure calculations. Having the number of electrons available makes the matrix file useful for testing density matrix solvers such as PEXSI. Therefore, it is recommended to set the correct number of electrons when initializing an matrix I/O handle, although setting it to an arbitrary number will not affect the matrix I/O operation.

2) [n_basis](#): This can be set to an arbitrary value if [task](#) is 0 ([READ_MATRIX](#)). Its value will be read from file when calling [elsi_read_mat_dim](#) or [elsi_read_mat_dim_sparse](#).

The MPI communicator which encloses the MPI tasks to perform the matrix I/O operation needs to be passed into ELSI via the [elsi_set_rw_mpi](#) subroutine.

[elsi_set_rw_mpi](#)(handle, mpi_comm)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	inout	Handle to matrix I/O instance.
mpi_comm	integer	in	MPI communicator.

When reading or writing a dense matrix, BLACS parameters are passed into ELSI via the [elsi_set_rw_blacs](#) subroutine.

[elsi_set_rw_blacs](#)(handle, blacs_ctxt, block_size)

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

When writing a sparse matrix, its dimensions are passed into ELSI via the [elsi_set_rw_csc](#) subroutine. The only sparse matrix format currently supported by ELSI matrix I/O is the [PEXSI_CSC](#) format. When reading a sparse matrix, there is no need to call this subroutine. The relevant parameters will be read from file when calling [elsi_read_mat_dim](#) or [elsi_read_mat_dim_sparse](#).

[elsi_set_rw_csc](#)(handle, global_nnz, local_nnz, local_col)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	inout	Handle to matrix I/O 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.

When a matrix I/O instance is no longer needed, its associated handle should be cleaned up by calling [elsi_finalize_rw](#).

[elsi_finalize_rw](#)(handle)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	inout	Handle to matrix I/O instance.

3.7.2 Writing Matrices

The following two subroutines write a dense matrix to file. Before writing a dense matrix, MPI and BLACS should be set up properly using [elsi_set_rw_mpi](#) and [elsi_set_rw_blacs](#).

[elsi_write_mat_real](#)(handle, filename, mat)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	in	Handle to matrix I/O instance.
filename	string	in	Name of file to write.
mat	real double, rank-2 array	in	Local matrix in 2D block-cyclic dense format.

[elsi_write_mat_complex](#)(handle, filename, mat)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	in	Handle to matrix I/O instance.
filename	string	in	Name of file to write.
mat	complex double, rank-2 array	in	Local matrix in 2D block-cyclic dense format.

The following two subroutines write a sparse matrix to file. Before writing a sparse matrix, MPI and CSC matrix format should be set up properly using [elsi_set_rw_mpi](#) and [elsi_set_rw_csc](#).

[elsi_write_mat_real_sparse](#)(handle, filename, row_idx, col_ptr, mat)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	in	Handle to matrix I/O instance.
filename	string	in	Name of file to write.
row_idx	integer, rank-1 array	in	Local row index array.
col_ptr	integer, rank-1 array	in	Local column pointer array.
mat	real double, rank-1 array	in	Local non-zero values in 1D block CSC format.

[elsi_write_mat_complex_sparse](#)(handle, filename, row_idx, col_ptr, mat)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	in	Handle to matrix I/O instance.
filename	string	in	Name of file to write.
row_idx	integer, rank-1 array	in	Local row index array.
col_ptr	integer, rank-1 array	in	Local column pointer array.
mat	complex double, rank-1 array	in	Local non-zero values in 1D block CSC format.

When writing a dense matrix to file, values smaller than a predefined threshold will be discarded. The default value of this threshold is 10^{-15} . It can be overridden via [elsi_set_rw_zero_def](#).

[elsi_set_rw_zero_def](#)(handle, zero_def)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	inout	Handle to matrix I/O instance.
zero_def	real double	in	When writing a dense matrix to file, values below this threshold will be discarded.

An array of eight user-defined integers can be optionally set up via [elsi_set_rw_header](#). This array will be attached to the matrix file written out by the above subroutines. When reading a matrix file, this array may be retrieved via [elsi_get_rw_header](#).

[elsi_set_rw_header](#)(handle, header)

Argument	Data Type	in/out	Explanation
handle	type(elsi_rw_handle)	inout	Handle to matrix I/O instance.
header	integer, rank-1 array	in	An array of eight integers.

3.7.3 Reading Matrices

The following subroutines read a dense or sparse matrix from file. While writing a matrix to file can be done in one step, it is easier to read a matrix from file in two steps, i.e., first read the dimension of the matrix and allocate memory accordingly, then read the actual data of the matrix.

The following three subroutines read a dense matrix from file. Before reading a dense matrix, MPI and BLACS should be set up properly using `elsi_set_rw_mpi` and `elsi_set_rw_blacs`. `elsi_read_mat_dim` is used to read the dimension of a matrix, including the number of electrons in the physical system (for testing purpose), the global size of the matrix, and the local size of the matrix. Memory needs to be allocated according to the return values of `local_row` and `local_col`. Then `elsi_read_mat_{real|complex}` may be called to read a real or complex matrix, respectively.

`elsi_read_mat_dim`(handle, filename, n_electron, n_basis, local_row, local_col)

Argument	Data Type	in/out	Explanation
<code>handle</code>	type(<code>elsi_rw_handle</code>)	inout	Handle to matrix I/O instance.
<code>filename</code>	string	in	Name of file to read.
<code>n_electron</code>	real double	out	Number of electrons.
<code>n_basis</code>	integer	out	Number of basis functions, i.e. global size of matrix.
<code>local_row</code>	integer	out	Local number of matrix rows.
<code>local_col</code>	integer	out	Local number of matrix columns.

`elsi_read_mat_real`(handle, filename, mat)

Argument	Data Type	in/out	Explanation
<code>handle</code>	type(<code>elsi_rw_handle</code>)	inout	Handle to matrix I/O instance.
<code>filename</code>	string	in	Name of file to read.
<code>mat</code>	real double, rank-2 array	out	Local matrix in 2D block-cyclic distribution.

`elsi_read_mat_complex`(handle, filename, mat)

Argument	Data Type	in/out	Explanation
<code>handle</code>	type(<code>elsi_rw_handle</code>)	inout	Handle to matrix I/O instance.
<code>filename</code>	string	in	Name of file to read.
<code>mat</code>	complex double, rank-2 array	out	Local matrix in 2D block-cyclic distribution.

The following three subroutines read a sparse matrix from file. Before reading a sparse matrix, MPI should be set up properly using `elsi_set_rw_mpi`. `elsi_read_mat_dim_sparse` is used to read the dimension of a matrix, including the number of electrons in the physical system (for testing purpose), the global size of the matrix, and the local size of the matrix. Memory needs to be allocated according to the return values of `local_nnz` and `local_col`. Then `elsi_read_mat_{real|complex}_sparse` may be called to read a real or complex matrix, respectively.

`elsi_read_mat_dim_sparse`(handle, filename, n_electron, n_basis, global_nnz, local_nnz, local_col)

Argument	Data Type	in/out	Explanation
<code>handle</code>	type(<code>elsi_rw_handle</code>)	inout	Handle to matrix I/O instance.
<code>filename</code>	string	in	Name of file to read.
<code>n_electron</code>	real double	out	Number of electrons.
<code>n_basis</code>	integer	out	Number of basis functions, i.e. global size of matrix.
<code>global_nnz</code>	integer	out	Global number of non-zeros.
<code>local_nnz</code>	integer	out	Local number of non-zeros.
<code>local_col</code>	integer	out	Local number of matrix columns.

`elsi_read_mat_real_sparse(handle, filename, row_idx, col_ptr, mat)`

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_rw_handle)</code>	inout	Handle to matrix I/O instance.
<code>filename</code>	string	in	Name of file to read.
<code>row_idx</code>	integer, rank-1 array	out	Local row index array.
<code>col_ptr</code>	integer, rank-1 array	out	Local column pointer array.
<code>mat</code>	real double, rank-1 array	out	Local non-zero values in 1D block CSC format.

`elsi_read_mat_complex_sparse(handle, filename, row_idx, col_ptr, mat)`

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_rw_handle)</code>	inout	Handle to matrix I/O instance.
<code>filename</code>	string	in	Name of file to read.
<code>row_idx</code>	integer, rank-1 array	out	Local row index array.
<code>col_ptr</code>	integer, rank-1 array	out	Local column pointer array.
<code>mat</code>	complex double, rank-1 array	out	Local non-zero values in 1D block CSC format.

An array of eight user-defined integers can be optionally set up via `elsi_set_rw_header`. This array will be attached to the matrix file written out by the above subroutines. When reading a matrix file, this array may be retrieved via `elsi_get_rw_header`.

`elsi_get_rw_header(handle, header)`

Argument	Data Type	in/out	Explanation
<code>handle</code>	<code>type(elsi_rw_handle)</code>	inout	Handle to matrix I/O instance.
<code>header</code>	integer, rank-1 array	out	An array of eight integers.

3.8 C/C++ Interface

ELSI is written in Fortran. A C interface around the core Fortran code is provided, 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.

3.9 Example Pseudo-Code

Typical workflow of ELSI within an electronic structure code is demonstrated by the following pseudo-code. In the “test” directory of the ELSI package, there are also examples that showcase the usage of ELSI in C and Fortran.

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

SCF initialize

```
call elsi_init (eh, ELPA, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)

do SCF cycle
  Update Hamiltonian

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

  Update electron density
  Check SCF convergence
end do

call elsi_finalize(eh)
```

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

SCF initialize

```
call elsi_init (eh, ELPA, MULTI_PROC, SIESTA_CSC, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_set_csc (eh, global_nnz, local_nnz, local_col, row_idx, col_ptr)
call elsi_set_csc_blk (eh, block_size_csc)

do SCF cycle
  Update Hamiltonian

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

  Update electron density
  Check SCF convergence
end do

call elsi_finalize(eh)
```

Remarks

- 1) Eigenvectors are returned in the `BLACS_DENSE` format, which is required to be properly set up.

Arbitrarily Distributed COO Sparse Matrix + ELSI Eigensolver Interface

SCF initialize

```
call elsi_init (eh, ELPA, MULTI_PROC, GENERIC_COO, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_set_coo (eh, global_nnz, local_nnz, row_idx, col_idx)

do SCF cycle
  Update Hamiltonian

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

  Update electron density
  Check SCF convergence
end do

call elsi_finalize(eh)
```

Remarks

- 1) Eigenvectors are returned in the `BLACS_DENSE` format, which is required to be properly set up.

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

SCF initialize

```
call elsi_init (eh, OMM, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)

do SCF cycle
  Update Hamiltonian

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

  Update electron density
  Check SCF convergence
end do

call elsi_finalize(eh)
```

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

SCF initialize

```
call elsi_init (eh, PEXSI, MULTI_PROC, SIESTA_CSC, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_csc (eh, global_nnz, local_nnz, local_col, row_idx, col_ptr)
call elsi_set_csc_blk (eh, block_size)

do SCF cycle
  Update Hamiltonian

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

  Update electron density
  Check SCF convergence
end do

call elsi_finalize(eh)
```

Remarks

- 1) Refer to the 5th remark in 3.5.4 for the chemical potential determination algorithm in PEXSI.

Arbitrarily Distributed COO Sparse Matrix + ELSI Density Matrix Interface

SCF initialize

```
call elsi_init (eh, PEXSI, MULTI_PROC, GENERIC_COO, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_coo (eh, global_nnz, local_nnz, row_idx, col_idx)

do SCF cycle
  Update Hamiltonian

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

  Update electron density
  Check SCF convergence
end do

call elsi_finalize(eh)
```

Remarks

- 1) Refer to the 5th remark in 3.5.4 for the chemical potential determination algorithm in PEXSI.

Multiple k -points Calculations

SCF initialize

```
call elsi_init (eh, NTPOLY, MULTI_PROC, BLACS_DENSE, n_basis, n_electron, n_state)
call elsi_set_mpi (eh, mpi_comm)
call elsi_set_blacs (eh, blacs_ctxt, block_size)
call elsi_set_kpoint (eh, n_kpt, i_kpt, i_wt)
call elsi_set_mpi_global (eh, mpi_comm_global)

do SCF cycle
  Update Hamiltonian

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

  Update electron density
  Check SCF convergence
end do

call elsi_finalize(eh)
```

Remarks

- 1) When there are multiple k -points, there is no change in the way ELSI solver interfaces are called.
- 2) The electronic structure code needs to assemble the real-space density from the density matrices returned for the k -points. The returned band structure energy, however, is already summed over all k -points with respect to the weight of each k -point. Refer to 3.2.4 for more information.
- 3) Spin-polarized calculations may be set up similarly.

Geometry Relaxation Calculations

SCF initialize

```
call elsi_init (eh, ...)
call elsi_set_* (eh, ...)

do geometry
  do SCF cycle
    Update Hamiltonian

    call elsi_{ev|dm}_{real|complex} (eh, ham, ovlp, ...)

    Update electron density
    Check SCF convergence
  end do

  Update geometry (overlap)

  call elsi_reinit (eh)
end do

call elsi_finalize(eh)
```

Bibliography

- [1] V. Yu, F. Corsetti, A. García, W. P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, A. Vázquez-Mayagoitia, C. Yang, H. Yang, V. Blum, ELSI: A unified software interface for Kohn-Sham electronic structure solvers, *Computer Physics Communications* 222 (2018) 267–285.
- [2] B. Aradi, B. Hourahine, T. Frauenheim, DFTB+, a sparse matrix-based implementation of the DFTB method, *The Journal of Physical Chemistry A* 111 (2007) 5678–5684.
- [3] W. Hu, L. Lin, C. Yang, DGDFT: A massively parallel method for large scale density functional theory calculations, *The Journal of Chemical Physics* 143 (2015) 124110.
- [4] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, M. Scheffler, Ab initio molecular simulations with numeric atom-centered orbitals, *Computer Physics Communications* 180 (2009) 2175–2196.
- [5] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, D. Sánchez-Portal, The SIESTA method for ab initio order-N materials simulation, *Journal of Physics: Condensed Matter* 14 (2002) 2745–2779.
- [6] T. Auckenthaler, V. Blum, H. J. Bungartz, T. Huckle, R. Johanni, L. Kramer, B. Lang, H. Lederer, P. R. Willems, Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, *Parallel Computing* 37 (2011) 783–794.
- [7] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H. J. Bungartz, H. Lederer, The ELPA library: Scalable parallel eigenvalue solutions for electronic structure theory and computational science, *Journal of Physics: Condensed Matter* 26 (2014) 213201.
- [8] P. Kûs, H. Lederer, A. Marek, GPU optimization of large-scale eigenvalue solver, in: *Numerical Mathematics and Advanced Applications ENUMATH 2017*, Springer International Publishing, 2019, pp. 123–131.
- [9] P. Kûs, A. Marek, S. Koecher, H.-H. Kowalski, C. Carbogno, C. Scheurer, K. Reuter, M. Scheffler, H. Lederer, Optimizations of the eigensolvers in the ELPA library, *Parallel Computing* 85 (2019) 167–177.
- [10] F. Corsetti, The orbital minimization method for electronic structure calculations with finite-range atomic basis sets, *Computer Physics Communications* 185 (2014) 873–883.
- [11] L. Lin, J. Lu, L. Ying, R. Car, W. E, 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 (2009) 755–777.
- [12] L. Lin, M. Chen, C. Yang, L. He, Accelerating atomic orbital-based electronic structure calculation via pole expansion and selected inversion, *Journal of Physics: Condensed Matter* 25 (2013) 295501.
- [13] L. Lin, A. García, G. Huhs, C. Yang, SIESTA-PEXSI: Massively parallel method for efficient and accurate ab initio materials simulation without matrix diagonalization, *Journal of Physics: Condensed Matter* 26 (2014) 305503.
- [14] M. Jacquelin, L. Lin, C. Yang, PSellInv - A distributed memory parallel algorithm for selected inversion: The symmetric case, *ACM Transactions on Mathematical Software* 43 (2016) 21.
- [15] W. Jia, L. Lin, Robust determination of the chemical potential in the pole expansion and selected inversion method for solving Kohn-Sham density functional theory, *The Journal of Chemical Physics* 147 (2017) 144107.
- [16] T. Imamura, S. Yamada, M. Machida, Development of a high-performance eigensolver on a peta-scale next-generation supercomputer system, *Progress in Nuclear Science and Technology* (2011) 643–650.

- [17] T. Fukaya, T. Imamura, Performance evaluation of the EigenExa eigensolver on Oakleaf-FX: Tridiagonalization versus pentadiagonalization, in: IEEE International Parallel and Distributed Processing Symposium Workshop, 2015, pp. 960–969.
- [18] V. Hernandez, J. E. Roman, V. Vidal, SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, *ACM Transactions on Mathematical Software* 31 (2005) 351–362.
- [19] C. Campos, J. E. Roman, Strategies for spectrum slicing based on restarted Lanczos methods, *Numerical Algorithms* 60 (2012) 279–295.
- [20] M. Keçeli, H. Zhang, P. Zapol, D. A. Dixon, A. F. Wagner, Shift-and-invert parallel spectral transformation eigensolver: Massively parallel performance for density-functional based tight-binding, *Journal of Computational Chemistry* 37 (2016) 448–459.
- [21] M. Keçeli, F. Corsetti, C. Campos, J. E. Roman, H. Zhang, Álvaro Vázquez-Mayagoitia, P. Zapol, A. F. Wagner, SIESTA-SIPs: Massively parallel spectrum-slicing eigensolver for an ab initio molecular dynamics package, *Journal of Computational Chemistry* 39 (2018) 1806–1814.
- [22] W. Dawson, T. Nakajima, Massively parallel sparse matrix function calculations with NTPoly, *Computer Physics Communications* 225 (2018) 154–165.
- [23] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK users’ guide*, SIAM, 1999.
- [24] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, *Parallel Computing* 36 (2010) 232–240.
- [25] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, Accelerating numerical dense linear algebra calculations with GPUs, *Numerical Computations with GPUs* (2014) 1–26.
- [26] A. H. R. Palser, D. E. Manolopoulos, Canonical purification of the density matrix in electronic-structure theory, *Physical Review B* 58 (1998) 12704–12711.
- [27] A. M. N. Niklasson, Expansion algorithm for the density matrix, *Physical Review B* 66 (2002) 155115.
- [28] L. A. Truflandier, R. M. Dianzinga, D. R. Bowler, Communication: Generalized canonical purification for density matrix minimization, *The Journal of Chemical Physics* 144 (2016) 091102.

License and Copyright

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

Copyright (c) 2015-2020, the ELSI team.
All rights reserved.

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 (LGPL3), libOMM (BSD2), NTPoly 2.3.2 (MIT), PEXSI 1.2.0 (BSD3), PT-SCOTCH 6.0.0 (CeCILL-C), and SuperLU_DIST 6.2.0 (BSD3) are redistributed through this version of ELSI. Individual license of each library can be found in the corresponding subfolder.