A proposal for the foundation of an ELSI test suite

Fabiano Corsetti

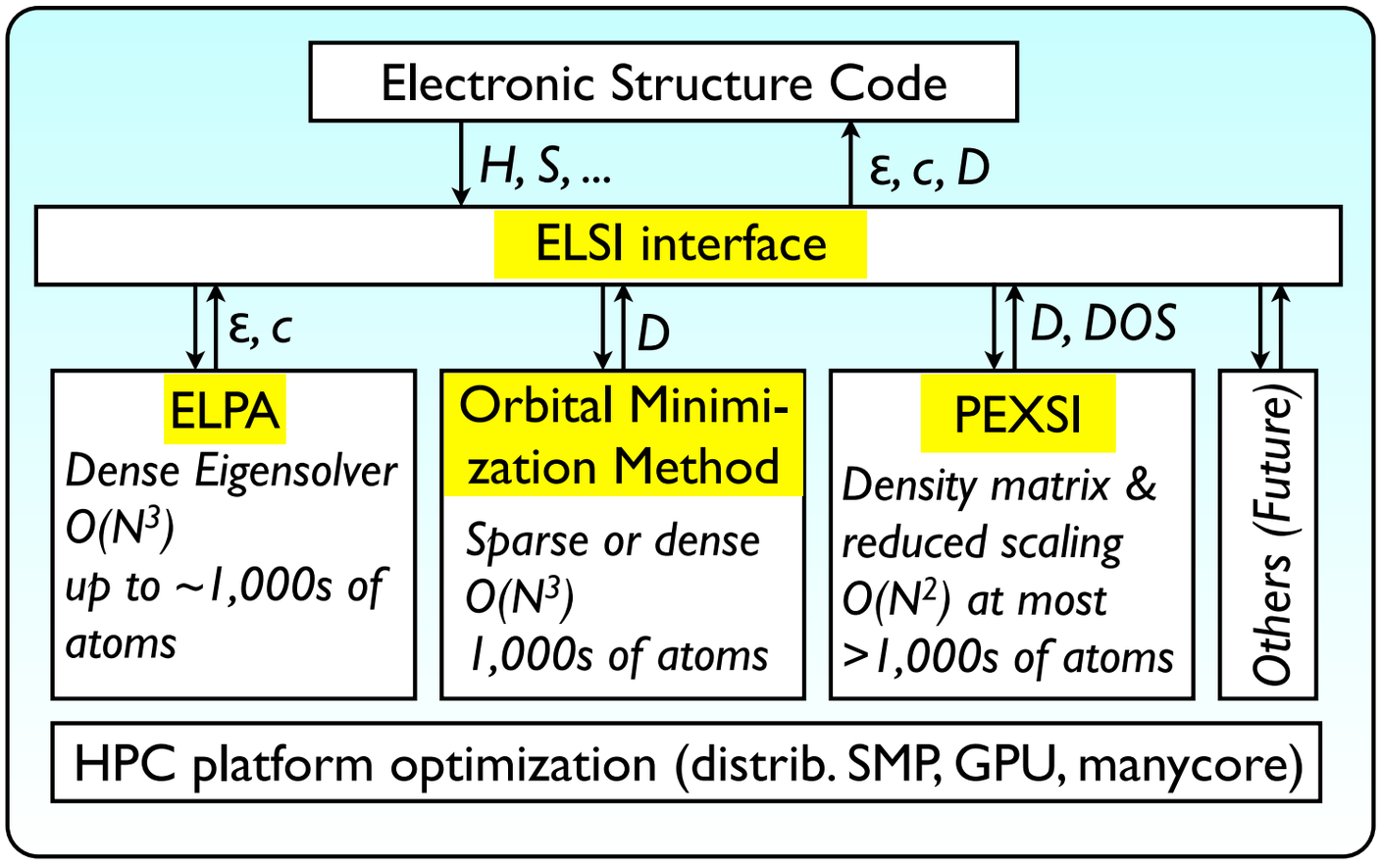
ELSI Connector Meeting 2016

Types of test (1)

- Unit tests within single solvers/interface layer… up to individual developers

- Global tests: examples of {*H*, *S*} for comparing solvers

- Global tests: example matrices for matrix algebra benchmarks



http://elsi-interchange.org

Types of test (2)

Run test within DFT code

Pros:
- Realistic matrices
- SCF iterations
- Small/portable input files

Cons:
- Hard to set up
- Hard to compile
- Bad framework for matrix algebra benchmarks

Store {*H*, *S*} generated by DFT code

Pros:
- Realistic matrices
- SCF iterations
- Useful for solver tests AND matrix algebra benchmarks

Cons:
- HUGE file storage and I/O problems
- Not very flexible (fixed problem size)

Generate random matrices on-the-fly

Pros:
- Simple
- Portable
- Flexible (matrix size and sparsity)

Cons:
- Not realistic

Future test suite

- What might it look like?
- Two-tier system:

Run test within DFT code

- Use sparingly… reserve for pre-release tests of ELSI

Generate {*H*, *S*} matrices for toy system on-the-fly

- Use for intense testing & benchmarking of:
  -ELSI interface
  -individual solvers
  -matrix algebra

Toy matrix tests (1)

- The basic strategy:

★ Store minimal tight-binding data generated by local-orbital DFT codes (SIESTA, FHI-aims, etc.)

★ Calculate {*H*, *S*} for arbitrary supercell on-the-fly

Pros:

- Realistic
- Very flexible (more later)
- Simple
- Portable
- Eliminates storage and I/O problems
- Suitable for solver tests and matrix algebra benchmarking

Cons:

- Can't handle large non-crystalline systems

Toy matrix tests (2)

- Flexibility: what do we want to be able to vary independently?

  ★ Matrix size

  ★ Matrix sparsity

  ★ System dimensionality (3D, 2D, 1D, 0D)

  ★ Band gap

  ★ Fraction of occupied eigenstates

  ★ Real/complex matrices (Γ/k-points)

  ★ Generalized/standard eigenvalue problem (with/without *S*)

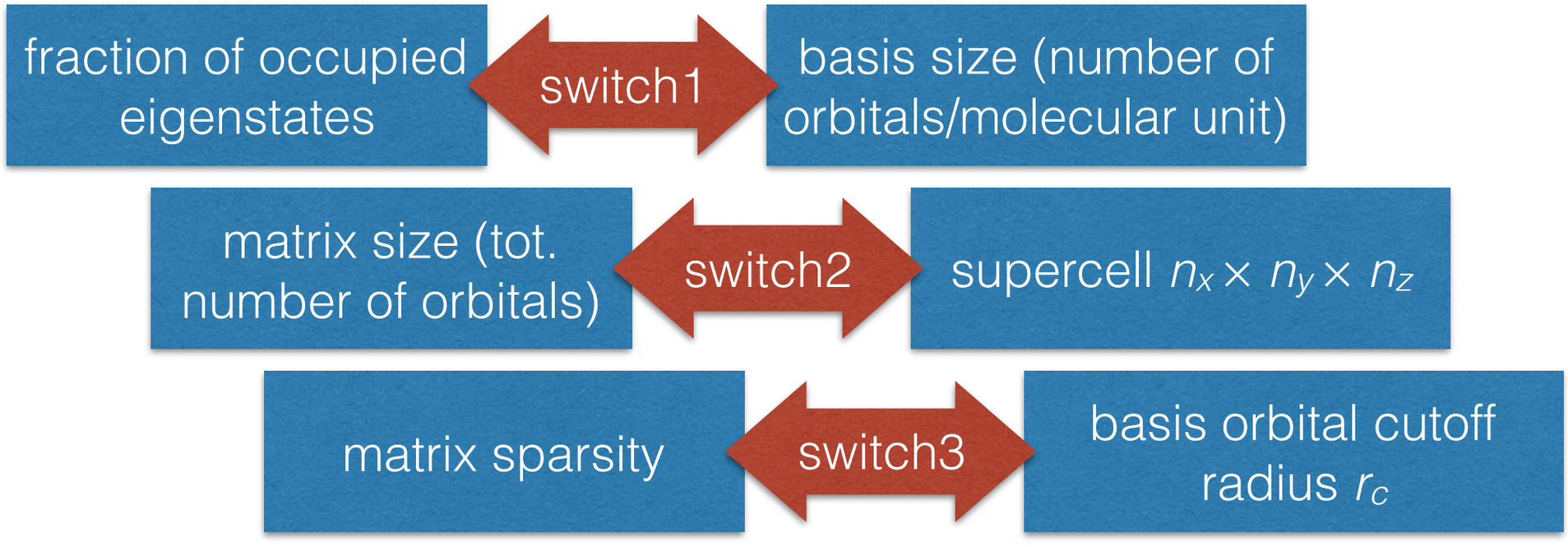
ToMatO

- Funny acronym: Toy Matrices On-the-fly
- F90 code
- Single subroutine call to generate test case
- Returns {*H*, *S*} in MatrixSwitch format
- MPI parallel
- Input data: “template” text files for different test systems (we can build up a database over time)

API

```
subroutine tomato_TB(template_basedir(*), system_label(*), &
                     switch1, frac_occ, basis_size, &
                     switch2, matrix_size, supercell(3), &
                     switch3, sparsity, orb_r_cut, &
                     num_occ_states, &
                     gamma_point, k_point(3), &
                     defect, defect_perturbation, &
                     H, S, m_storage(5), &
                     build_matrix)
```

`character` `logical` `integer` `double precision` `type(matrix)`



Calling example (1)

```
program example
  use MatrixSwitch

  implicit none

  character(5) :: m_storage

  integer :: basis_size, matrix_size, supercell(3), num_occ_states

  double precision :: frac_occ, sparsity, orb_r_cut, k_point(3)

  type(matrix) :: H, S

  m_storage='sdden'
  frac_occ=0.24d0
  matrix_size=1200
  sparsity=0.80d0
  k_point=(/0.0d0, 0.0d0, 0.0d0/)

  call tomato_TB('data', 'Si', &
                 .true., frac_occ, basis_size, &
                 .true., matrix_size, supercell, &
                 .true., sparsity, orb_r_cut, &
                 num_occ_states, &
                 .false., k_point, &
                 .false., 0.0d0, &
                 H, S, m_storage, &
                 .true.)

  call m_deallocate(S)
  call m_deallocate(H)

end program example
```

Calling example (2)

```
program example
  use MatrixSwitch

  implicit none

  character(5) :: m_storage

  integer :: basis_size, matrix_size, supercell(3), num_occ_states
                                                    sity, orb_r_cut, k_point(3)
```

On input:

```
frac_occ    = 0.24          basis_size not set
matrix_size = 1200          supercell  not set
sparsity    = 0.80          orb_r_cut  not set

num_occ_states not set
```

```
  call tomato_TB('data', 'Si', &
                 .true., frac_occ, basis_size, &
                 .true., matrix_size, supercell, &
                 .true., sparsity, orb_r_cut, &
                 num_occ_states, &
                 .false., k_point, &
                 .false., 0.0d0, &
                 H, S, m_storage, &
                 .true.)

  call m_deallocate(S)
  call m_deallocate(H)

end program example
```

On output:

```
frac_occ    = 0.25          basis_size = 8
matrix_size = 1280          supercell  = (4, 4, 5)
sparsity    = 0.8146        orb_r_cut  = 0.60

num_occ_states = 320
```

Template files (1)

- Each physical system requires at least one template file containing unfolded {*H*, *S*} elements for the unit cell

Si_40_38.template

```
#      16712
  1   1   0  -1   0 -3.138E-03  1.608E-04
  1   5   0  -1   0  4.663E-03 -2.471E-04
  1   6   0  -1   0 -4.663E-03  2.471E-04
  1   7   0  -1   0  3.029E-05  0.000E+00
  1  17   0  -1   0 -2.450E-05  0.000E+00
  1  18   0  -1   0  8.520E-03 -4.766E-04
  1  19   0  -1   0 -2.441E-03  1.376E-04
  1  20   0  -1   0  2.450E-05  0.000E+00
  1  21   0  -1   0  4.227E-03 -2.383E-04
  1  32   0  -1   0 -4.332E-03  2.673E-04
  1  33   0  -1   0  3.308E-06  0.000E+00
  1  34   0  -1   0  1.016E-02 -6.212E-04
  1  35   0  -1   0  2.800E-03 -1.691E-04
  1  36   0  -1   0 -3.772E-05  0.000E+00
  1  37   0  -1   0  1.069E-02 -6.548E-04
```

- By providing templates for a range of basis sizes and orbitals cutoffs, tomato can pick the best one to match the requested sparsity and fraction of occupied orbitals

- tomato can also select a subset of basis orbitals in the template file, so that only the biggest basis needs to be stored

Template files (2)

- One physical system: ~50 MB
- So far (from SIESTA):

  ★ Bulk silicon (3D)

  ★ Diamond (3D)

  ★ BN monolayer (2D)

  ★ Graphene (2D)

  ★ Polyethylene chain (1D)

Future work

- Improve the estimation of the sparsity
- Improve the functionality for breaking crystalline symmetry
- Add more template files (different DFT codes)
- Write tomato test code for the ELSI interface
- Plane-wave toy systems

Thank you for your attention!