



DRIVING
THE EXASCALE
TRANSITION

Scientific software and libraries for electronic structure community

MaX webinar, 24 June 2020



Webinar is represented by



Anton Kozhevnikov
Scientific Software &
Libraries Group Lead
CSCS



Shoshana Jakobovits
Software Engineer
CSCS

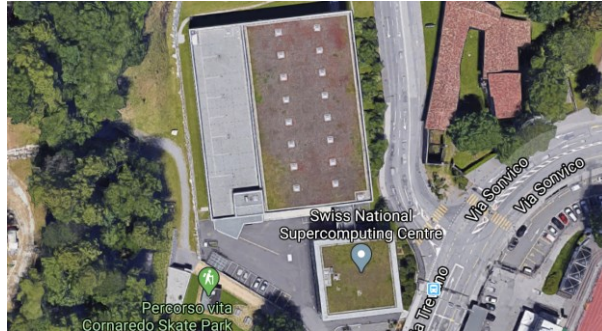
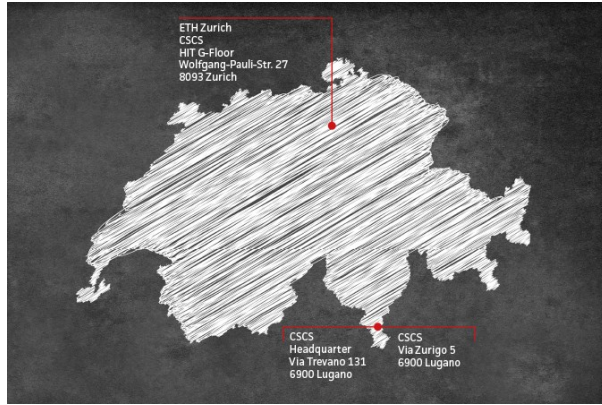


Marko Kabic
Software Engineer
CSCS



Simon Frasc
Software Engineer
CSCS

Swiss National Supercomputing Centre – CSCS



Piz Daint supercomputer at CSCS

2013 - first installation

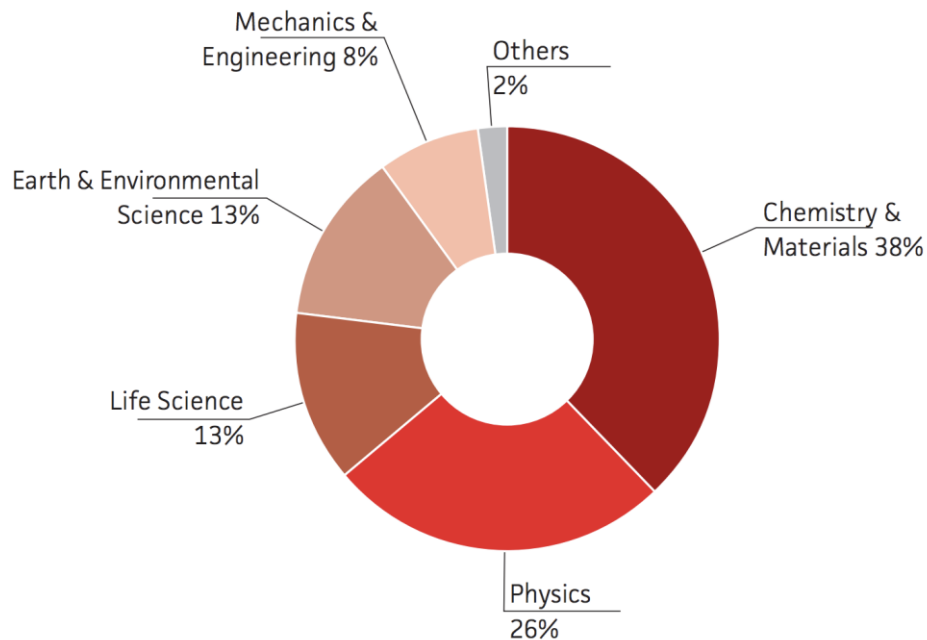
Cray XC30: 5272 nodes of 8-core Intel
SandyBridge@2.6GHz + NVIDIA K20X
Peak performance: 6.271 Petaflops

2016 - upgrade

Cray XC50: 5704 nodes of 12-core Intel
Haswell@2.6GHz + NVIDIA P100
Peak performance: 21.230 Petaflops



Material science codes at CSCS



- CP2K
 - Localized Gaussian basis set
 - Sparse matrix multiplication for $O(N)$ method
 - Dense eigen-solver for diagonalization-based SCF
 - Dense matrix multiplication for RPA calculation
 - FFTs
- Quantum ESPRESSO
 - Delocalized plane-wave basis set
 - FFTs
 - Davidson iterative subspace diagonalisation
 - dense eigen-solver
 - dense linear algebra

How CSCS can help community in porting scientific applications to novel architectures?

Porting scientific codes to GPUs

Scientific community applications are typically:

- monolithic all-in-one Fortran90
- MPI (with OpenMP) implementation
- ignorant of GPU

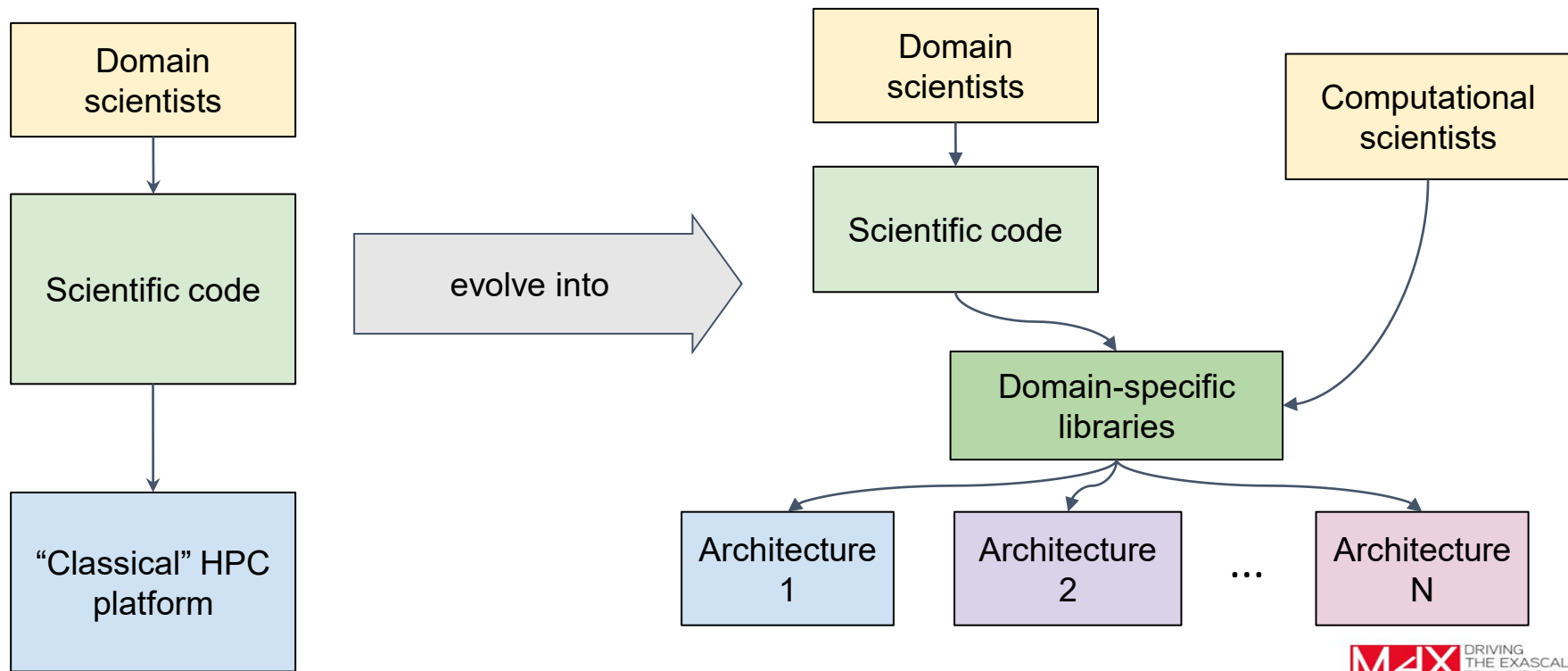
Usual steps of porting such applications to GPU:

- cleanup and refactor the code
- (probably) change the data layout
- fully utilize CPU threads (this helps to understand the compute-intensive kernels of the application)
- move compute-intensive kernels to GPU
 - OpenACC
 - OpenMP ≥ 4.5
 - CUDA with ISO_C_BINDING or Cuda-Fortran
 - OpenCL



Separation of work

CSCS vision: complexity of current and emerging HPC platforms and programming models should be reflected in the way we develop scientific software.



DBCSR library

Shoshana Jakobovits

COSMA library

Marko Kabic



Motivation

Yet another matrix multiplication?

Motivation

Efforts to achieve communication-optimality:

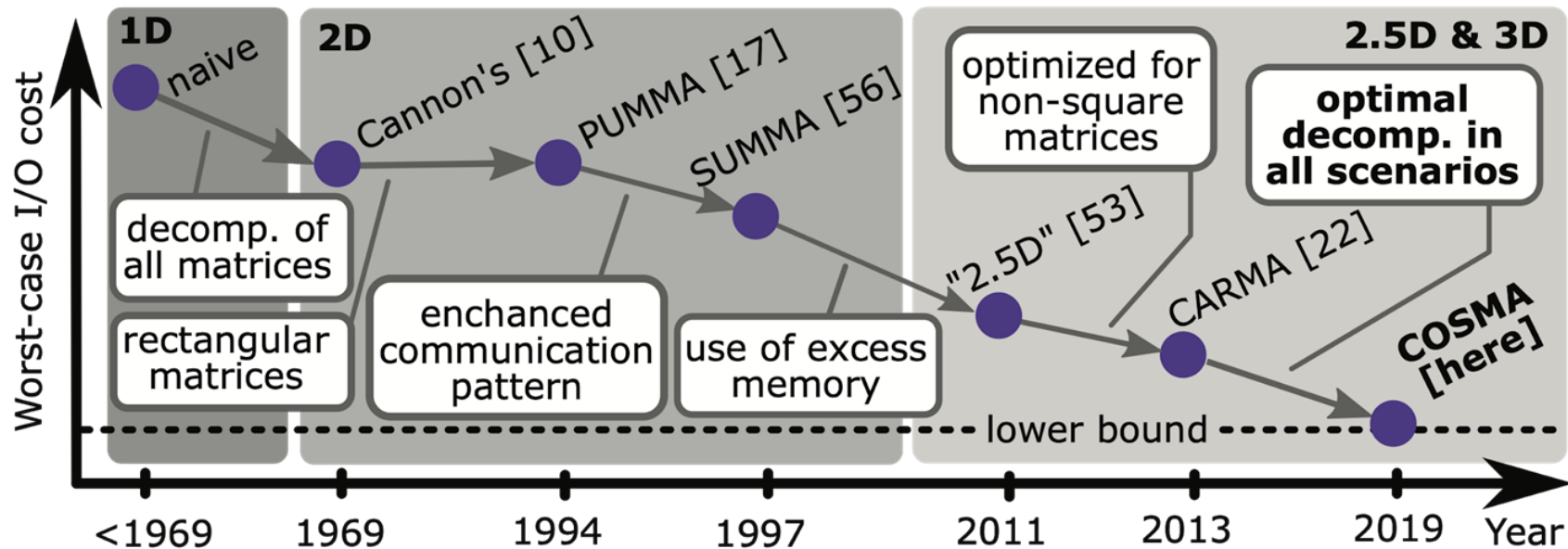


Figure 2: Illustratory evolution of MMM algorithms reaching the I/O lower bound.

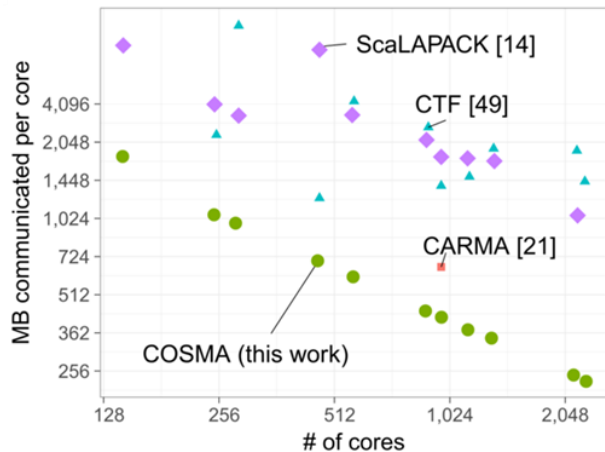
Evaluation

EVALUATION

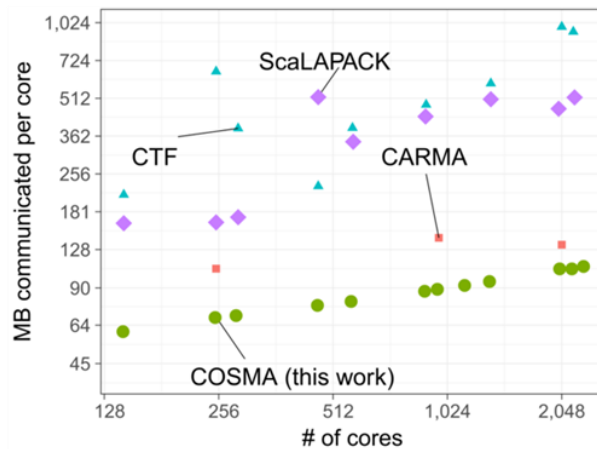
Total communication volume for “largeK”



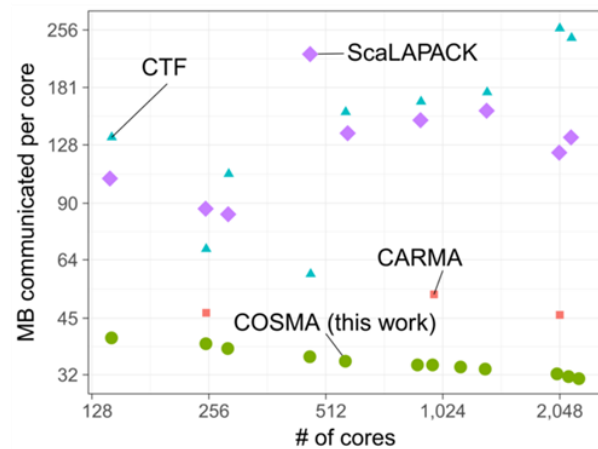
matrices



(a) Strong scaling, $n = m = 17,408$, $k = 3,735,552$



(b) Limited memory, $m = n = 979p^{\frac{1}{3}}$, $k = 1.184p^{\frac{2}{3}}$



(c) Extra memory, $m = n = 979p^{\frac{2}{9}}$, $k = 1.184p^{\frac{4}{9}}$

↓ lower = better

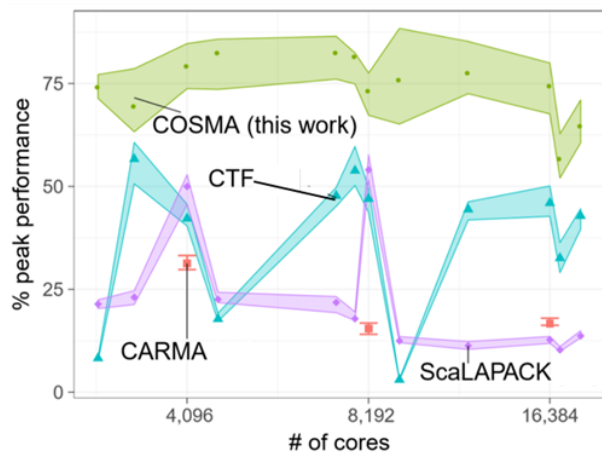
Evaluation

EVALUATION

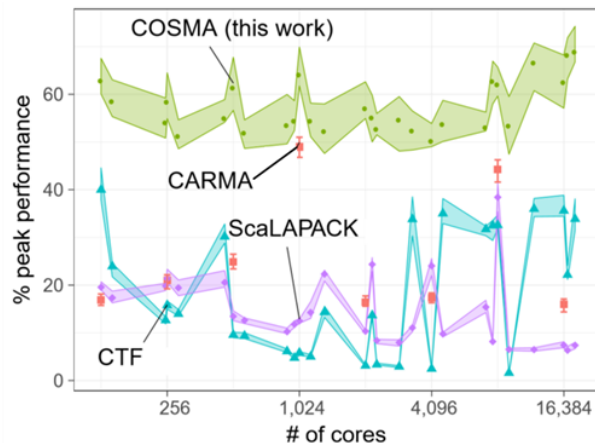
% of achieved peak performance for “largeK”



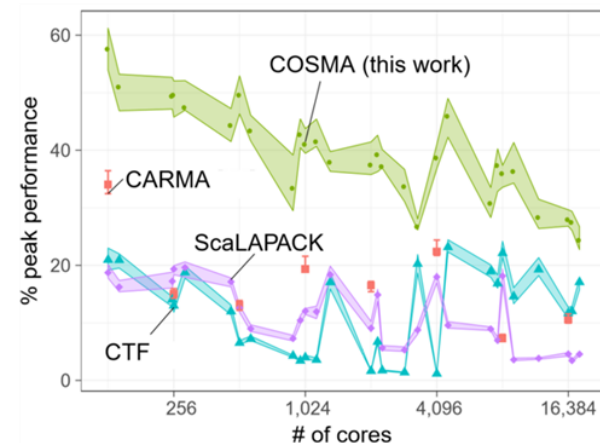
↑ higher = better



(a) Strong scaling, $n = m = 17,408$, $k = 3,735,552$



(b) Limited memory, $m = n = 979p^{\frac{1}{3}}$, $k = 1.184p^{\frac{2}{3}}$



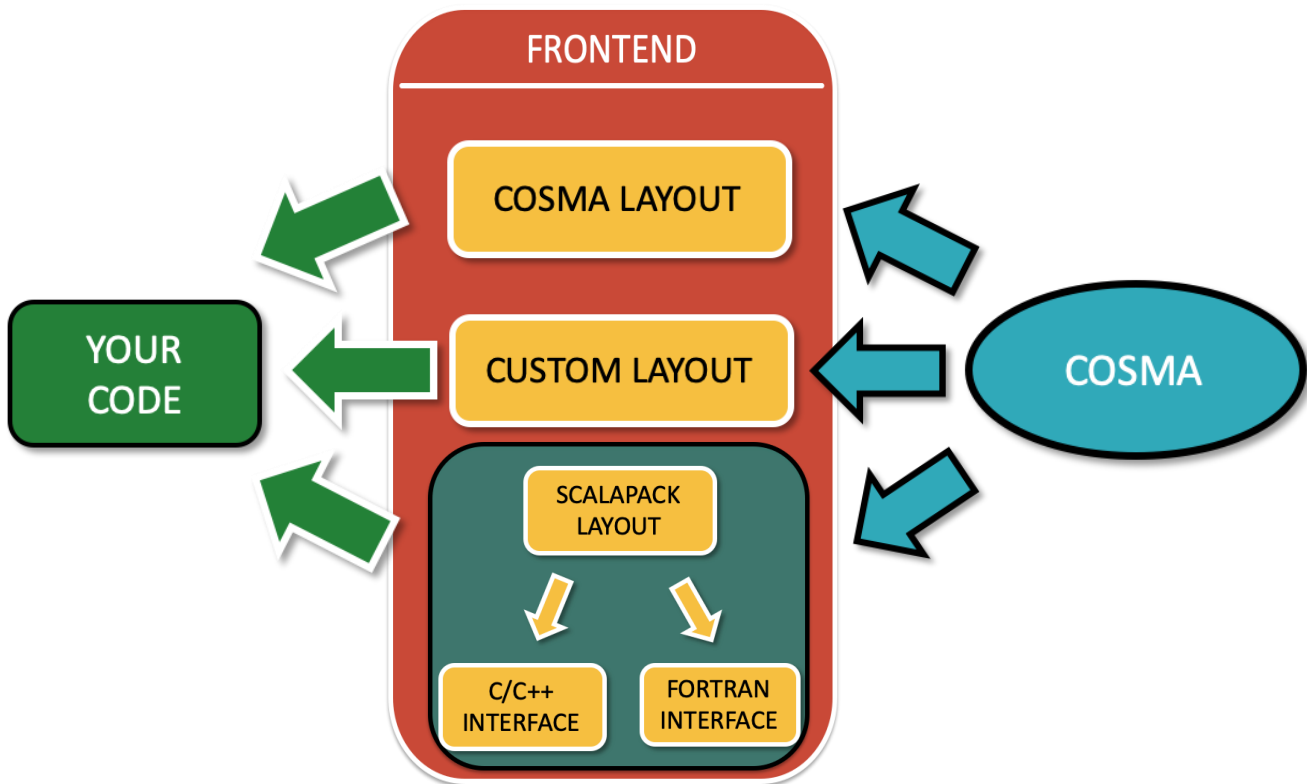
(c) Extra memory, $m = n = 979p^{\frac{2}{9}}$, $k = 1.184p^{\frac{4}{9}}$

Portability and Usability

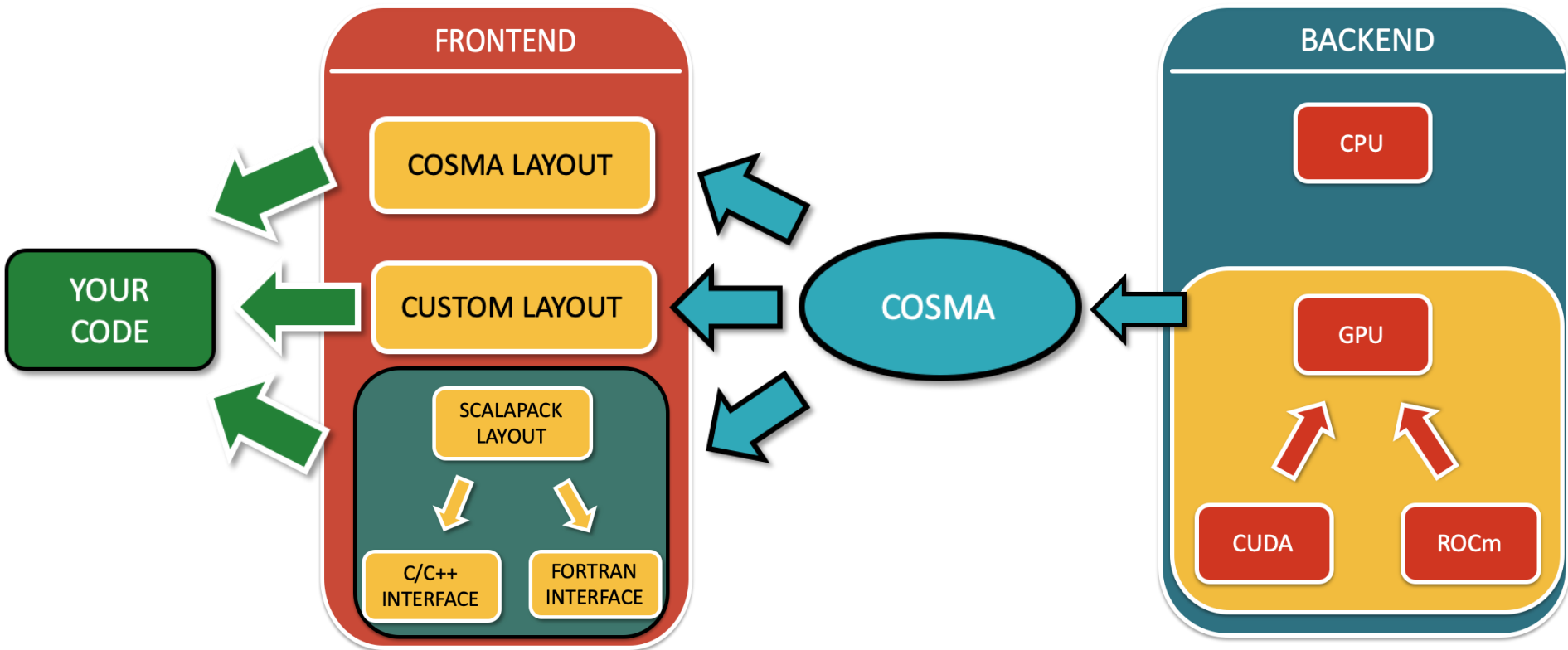
YOUR
CODE

COSMA

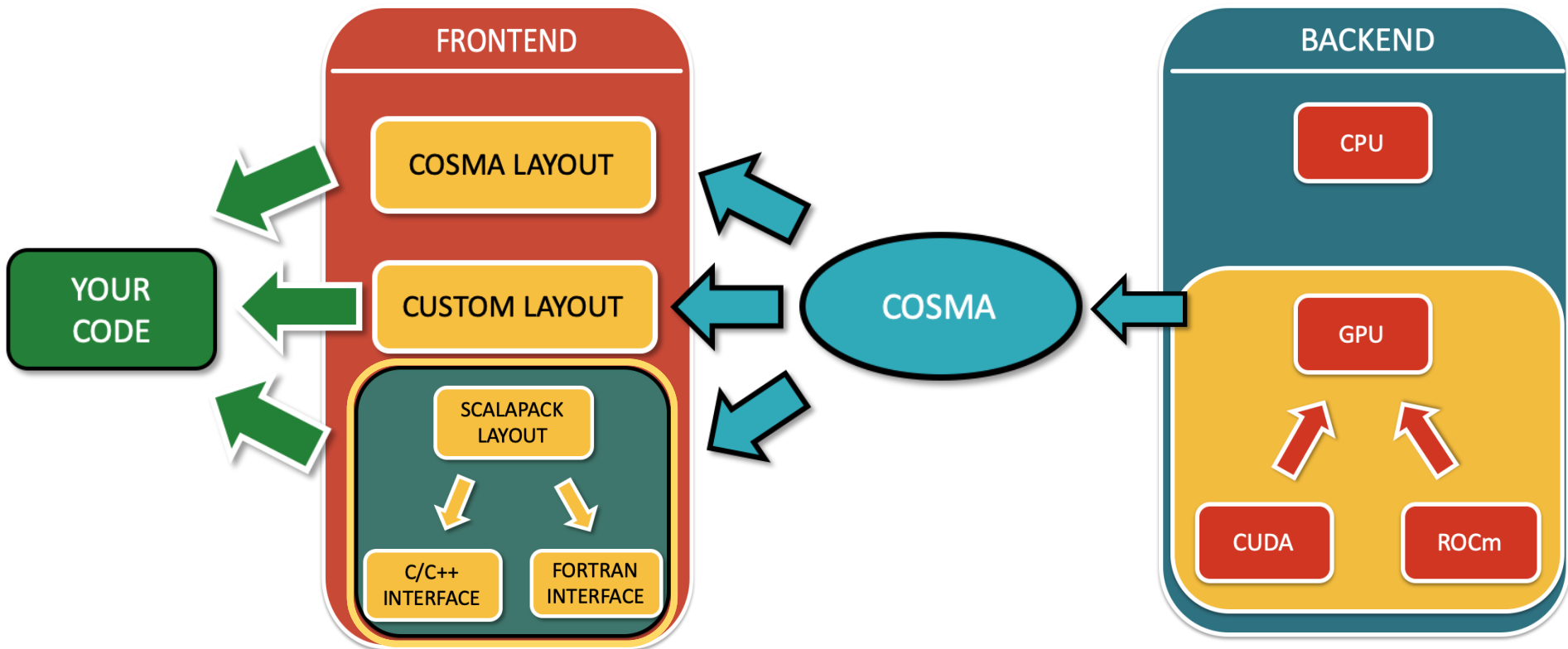
Portability and Usability



Portability and Usability



Portability and Usability



SCALAPACK-wrapper

- Follow the 30 seconds tutorial: <https://github.com/eth-cscs/COSMA#using-cosma-in-30-seconds>

SCALAPACK-wrapper

- Follow the 30 seconds tutorial: <https://github.com/eth-cscs/COSMA#using-cosma-in-30-seconds>

COMPILE COSMA

```
→ git clone --recursive  
  https://github.com/eth-cscs/COSMA cosma  
  && cd cosma  
→ mkdir build && cd build  
→ cmake -DCOSMA_BLAS=CUDA  
        -DCOSMA_SCALAPACK=MKL  
        -DCMAKE_INSTALL_PREFIX=<install-dir>  
        ..  
→ make -j 8  
→ make install
```


SCALAPACK-wrapper

- Follow the 30 seconds tutorial: <https://github.com/eth-cscs/COSMA#using-cosma-in-30-seconds>

COMPILE COSMA

```
→ git clone --recursive
  https://github.com/eth-cscs/COSMA cosma
  && cd cosma
→ mkdir build && cd build
→ cmake -DCOSMA_BLAS=CUDA
        -DCOSMA_SCALAPACK=MKL
        -DCMAKE_INSTALL_PREFIX=<install-dir>
        ..
→ make -j 8
→ make install
```

LINK TO COSMA

```
# link to COSMA, before any SCALAPACK
→ LIBS += -L<install-dir>/lib64
        -lcosma_pwgemm
        -lcosma -lgrid2grid
        -lTiled-MM
        -lcublas -lcudart -lrt

# include headers
→ INCS += -I<install-dir>/include
```

SCALAPACK-wrapper



used in CP2K

COMPILE COSMA

```
→ git clone --recursive
  https://github.com/eth-cscs/COSMA cosma
  && cd cosma
→ mkdir build && cd build
→ cmake -DCOSMA_BLAS=CUDA
        -DCOSMA_SCALAPACK=MKL
        -DCMAKE_INSTALL_PREFIX=<install-dir>
        ..
→ make -j 8
→ make install
```

LINK TO COSMA

```
# link to COSMA, before any SCALAPACK
→ LIBS += -L<install-dir>/lib64
        -lcosma_pwgemm
        -lcosma -lgrid2grid
        -lTiled-MM
        -lcublas -lcudart -lrt

# include headers
→ INCS += -I<install-dir>/include
```

SCALAPACK-wrapper



used in CP2K

COMPILE COSMA

```
→ git clone --recursive
  https://github.com/eth-cscs/COSMA cosma
  && cd cosma
→ mkdir build && cd build
→ cmake -DCOSMA_BLAS=CUDA
        -DCOSMA_SCALAPACK=MKL
        -DCMAKE_INSTALL_PREFIX=<install-dir>
        ..
→ make -j 8
→ make install
```



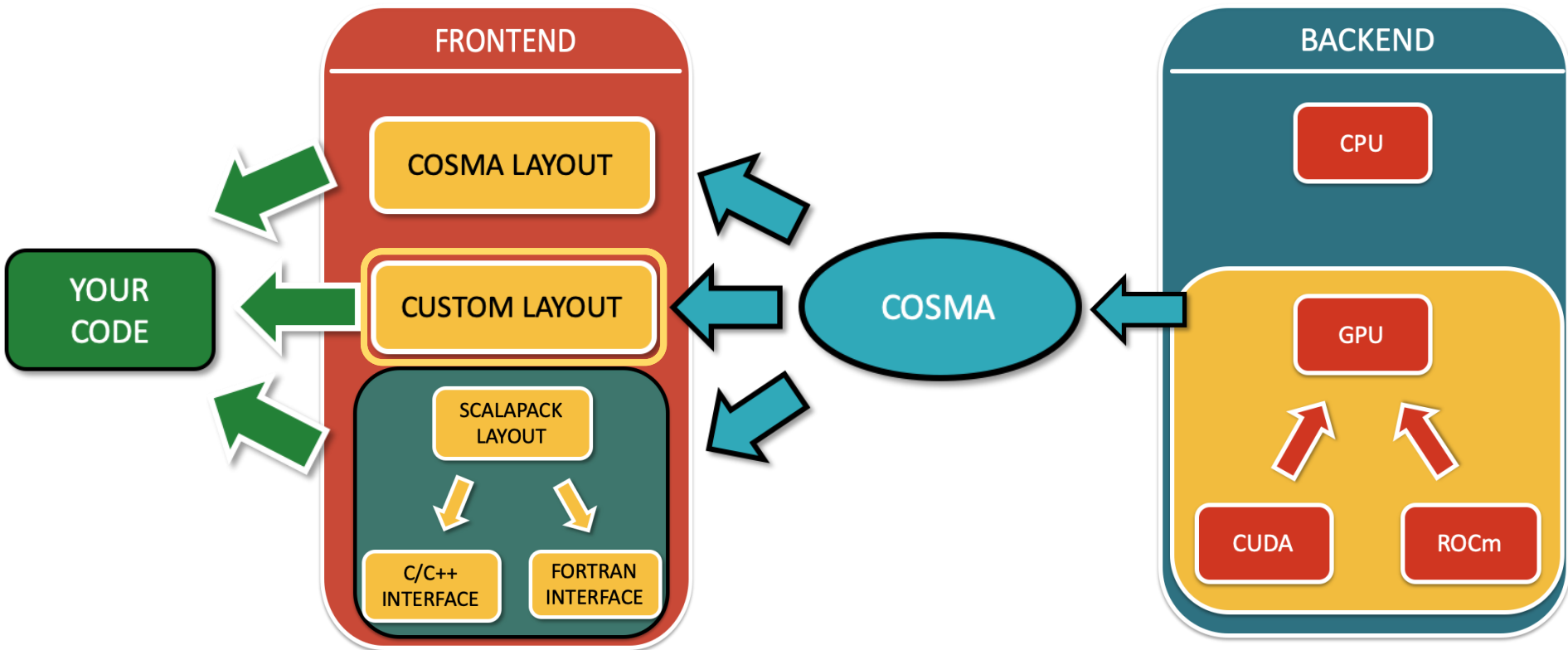
user code untouched!

LINK TO COSMA

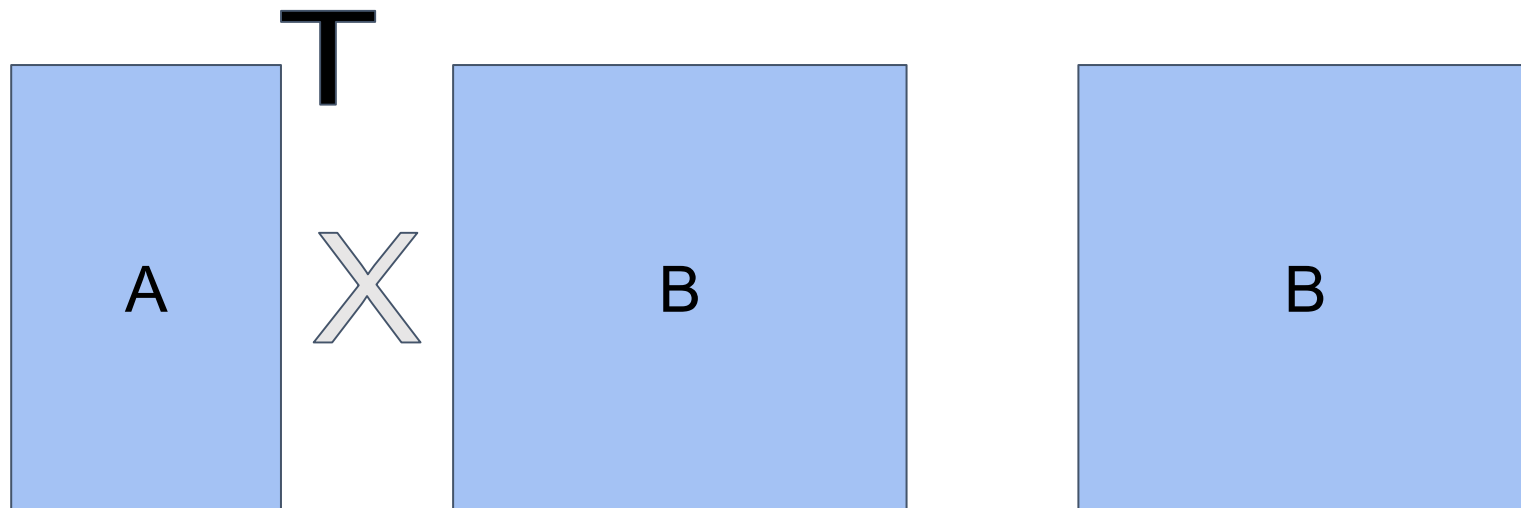
```
# link to COSMA, before any SCALAPACK
→ LIBS += -L<install-dir>/lib64
        -lcosma_pwgemm
        -lcosma -lgrid2grid
        -lTiled-MM
        -lcublas -lcudart -lrt

# include headers
→ INCS += -I<install-dir>/include
```

Portability and Usability



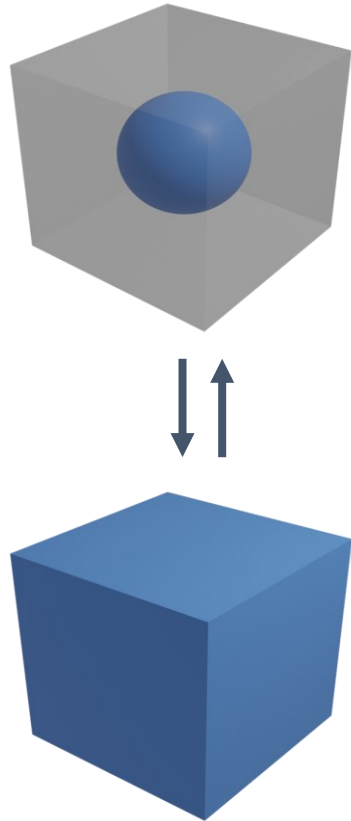
Portability and Usability



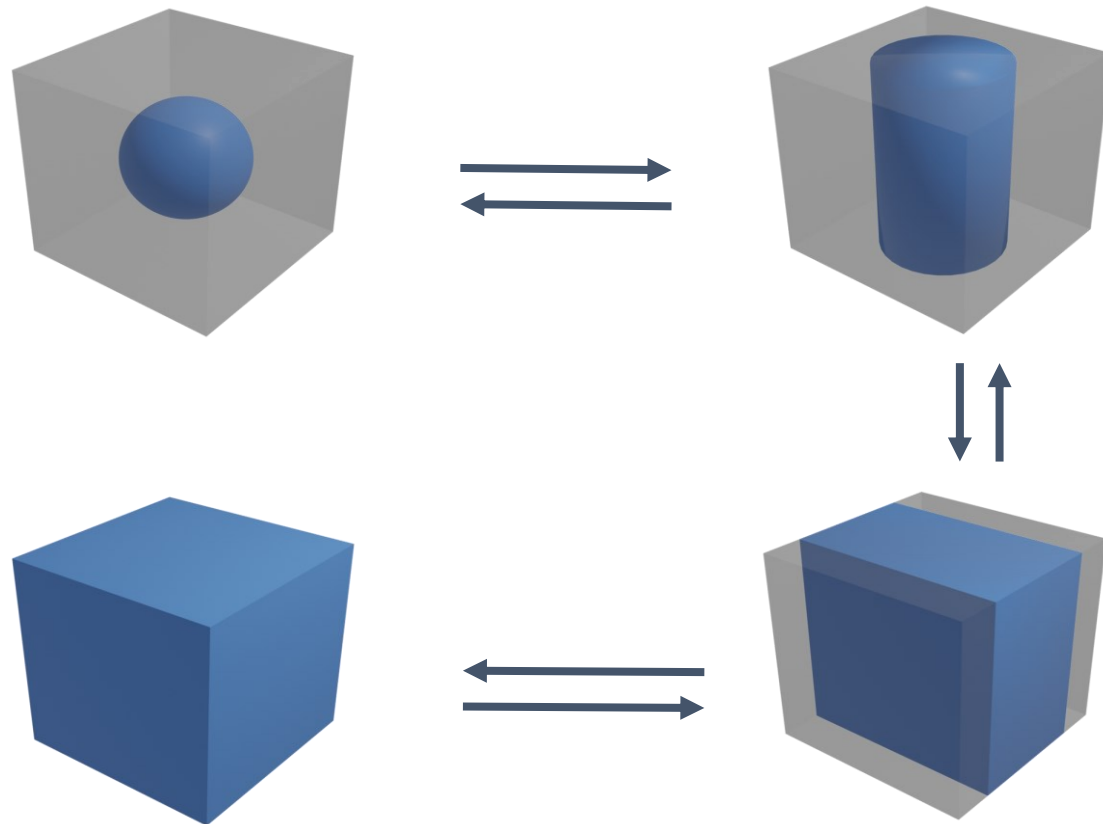
SpFFT library

Simon Frasch

SpFFT



SpFFT



SpFFT library

Design goals:

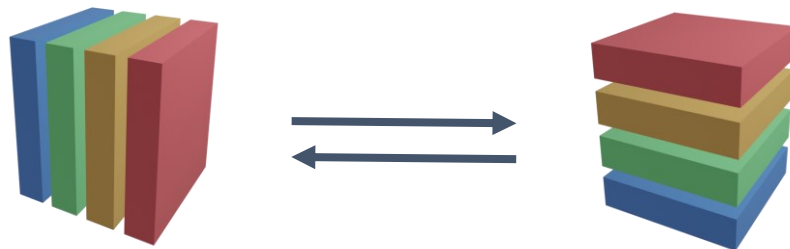
- Distributed 3D FFT computation with sparse input
- Resource reuse for transforms of different sizes
- Support for shifted indexing with centered zero-frequency
- Full use of Hermitian symmetry for complex-to-real transforms

Implementation:

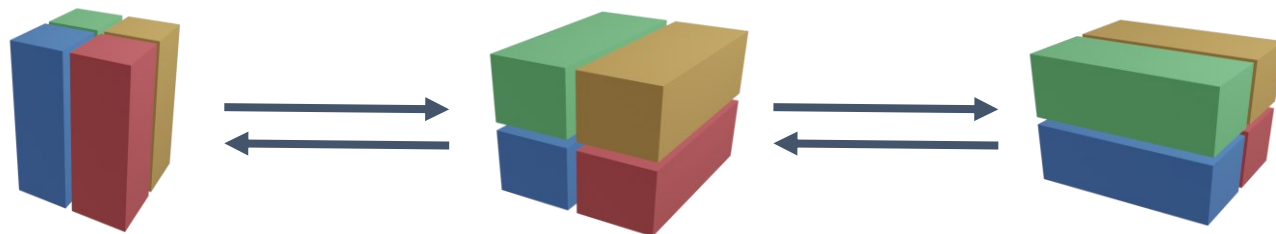
- Written in C++11
- Only mandatory dependency: Library providing a FFTW 3.x interface
- Optional parallelization and acceleration with:
 - OpenMP
 - MPI
 - CUDA or ROCm

SpFFT - Data Decomposition

Slab decomposition:

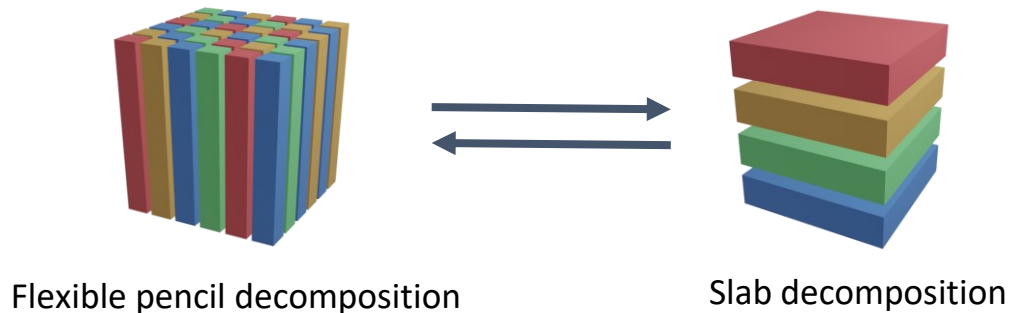


Pencil decomposition:



SpFFT - Data Decomposition

SpFFT uses a mixed decomposition:



Advantages:

- Less constraints on the distribution of sparse input data
- Better suited for GPU acceleration

Disadvantages:

- Distribution of dense data is limited by the size of one dimension

SpFFT - Data exchange

Three MPI exchange methods are supported:

- **MPI_Alltoall**
 - Fixed message sizes
 - Typically best optimized for large number of ranks
- **MPI_Alltoallv**
 - Adapts to non-uniform data distribution with variable message sizes
- **MPI_Alltoallw**
 - Manual packing / unpacking of data before exchange can be avoided by using custom data types for each message

Additional features:

- Optional use of conversion to / from single precision for MPI exchange step
- CUDA aware MPI with GPUDirect to avoid data transfer between host and device

SpFFT - Interface

The interface is based on two constructs:

Grid

- Allocates memory for transforms up to a given maximum size
- Transforms of different sizes can be executed on the same grid, allowing for memory reuse

Transform

- Associated to a reference counted grid
- Created with frequency (Miller) indices of sparse input data
- For GPU acceleration: Accepts host and device pointers. Output can be selected to be placed on host or device memory.

Note: No CUDA or ROCm API is exposed to the user.

SpFFT - Example

```
! Create grid, which allocates necessary resources
spfft_grid_create_distributed(grid, dffts%nr1, dffts%nr2, dffts%nr3,&
                             dffts%nsp(dffts%myype+1), dffts%my_nr3p,&
                             SPFFT_PU_HOST, 1, MPI_COMM_WORLD, SPFFT_EXCH_BUFFERED)

! Create transform on grid with Miller indices
spfft_transform_create(transform, grid, SPFFT_PU_HOST, SPFFT_TRANS_C2C,&
                       dffts%nr1, dffts%nr2, dffts%nr3, dffts%my_nr3p,&
                       size(mill)/3, SPFFT_INDEX_TRIPLETS, mill)

! Grids are reference counted. Can be safely destroyed, since resources are only freed
! after associated transforms have been destroyed as well.
spfft_grid_destroy(grid)

! Memory for storing real space data is provided by transform.
! Must be translated from a C pointer.
spfft_transform_get_space_domain(transform, SPFFT_PU_HOST, psic_ptr)
call c_f_pointer(psic_ptr, psic, [n])

! Transform nbnd times forward and backwards.
do ib = 1, nbnd, 1
    spfft_transform_backward(transform, psi(:,ib), SPFFT_PU_HOST)

    ! Work on real space data here...

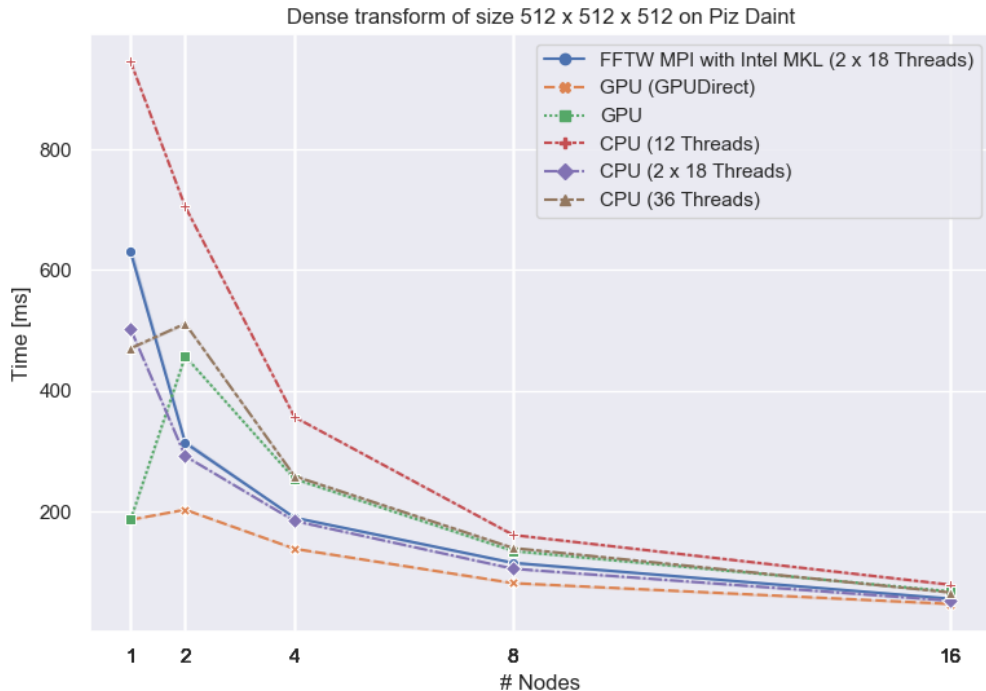
    spfft_transform_forward(transform, SPFFT_PU_HOST, hpsi(:,ib), SPFFT_FULL_SCALING)
enddo

! All resources are freed by destroying the only / last transform
spfft_transform_destroy(transform)
```

SpFFT - Benchmark

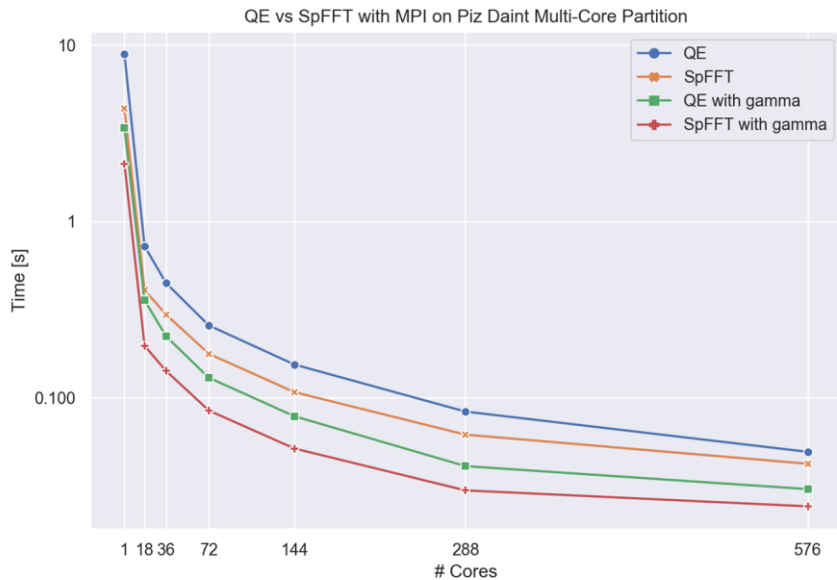
Benchmark:

- FFTW is executed with transposed output (only one internal MPI call necessary)
- **GPU node:** Intel Xeon E5-2690 v3 (12 cores), Nvidia Tesla P100
- **Multi-Core node:** 2x Intel Xeon E5-2695 v4 (2 x 18 cores)



SpFFT vs QE

Timings with single thread per rank:



Multi-thread scaling with two MPI ranks:



Size: 243 x 384 x 576
Density: 34%

SIRIUS library

Anton Kozhevnikov

Motivation for a common plane-wave DFT library

- Many similar full-potential LAPW codes (Exciting, Elk, FLEUR, Wien2k)
- Many similar pseudopotential PW codes (Quantum ESPRESSO, Abinit, VASP)
- Core DFT functionality is the same (compute total energy, magnetic moments, stress tensor, forces)
- A lot of common functionality between FP-LAPW and PP-PW methods

Accelerating and writing architecture backends for individual DFT codes is a waste of resources.

It is much more efficient to focus on the development of a common DFT functionality and create interfaces to various electronic-structure codes.

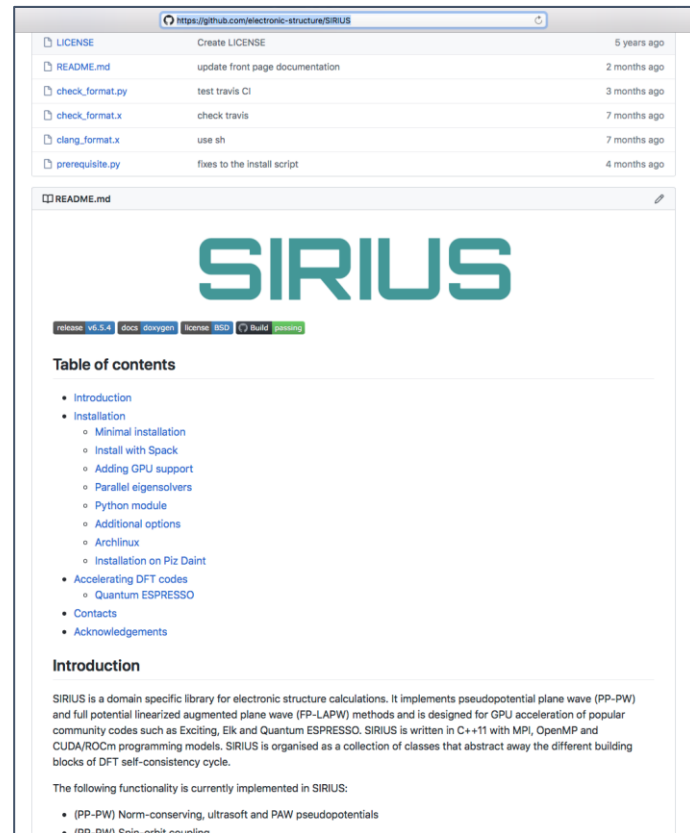
SIRIUS library

SIRIUS is a domain specific library organized as a collection of C++ classes that abstract away the different building blocks of PW and LAPW codes. The library is written in C++11 with MPI, OpenMP and CUDA/ROCm programming models.

<https://github.com/electronic-structure/SIRIUS>

<https://electronic-structure.github.io/SIRIUS-doc/>

```
git clone --recursive https://github.com/electronic-structure/SIRIUS.git
mkdir SIRIUS/build
cd SIRIUS/build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/local
make -j install
```



The screenshot shows the GitHub repository for SIRIUS. The top part displays a list of files and their commit dates:

| File | Commit Message | Time Ago |
|-----------------|---------------------------------|--------------|
| LICENSE | Create LICENSE | 5 years ago |
| README.md | update front page documentation | 2 months ago |
| check_format.py | test travis CI | 3 months ago |
| check_format.x | check travis | 7 months ago |
| clang_format.x | use sh | 7 months ago |
| prerequisite.py | fixes to the install script | 4 months ago |

Below this is the README.md file, which features the SIRIUS logo and a navigation bar with links for release v6.5.4, docs, doxygen, license, BSD, and Build passing. The main content includes a Table of contents with the following items:

- Introduction
- Installation
 - Minimal installation
 - Install with Spack
 - Adding GPU support
 - Parallel eigensolvers
 - Python module
 - Additional options
 - Archlinux
 - Installation on Piz Daint
- Accelerating DFT codes
 - Quantum ESPRESSO
- Contacts
- Acknowledgements

The Introduction section states: "SIRIUS is a domain specific library for electronic structure calculations. It implements pseudopotential plane wave (PP-PW) and full potential linearized augmented plane wave (FP-LAPW) methods and is designed for GPU acceleration of popular community codes such as Exciting, Elk and Quantum ESPRESSO. SIRIUS is written in C++11 with MPI, OpenMP and CUDA/ROCm programming models. SIRIUS is organised as a collection of classes that abstract away the different building blocks of DFT self-consistency cycle."

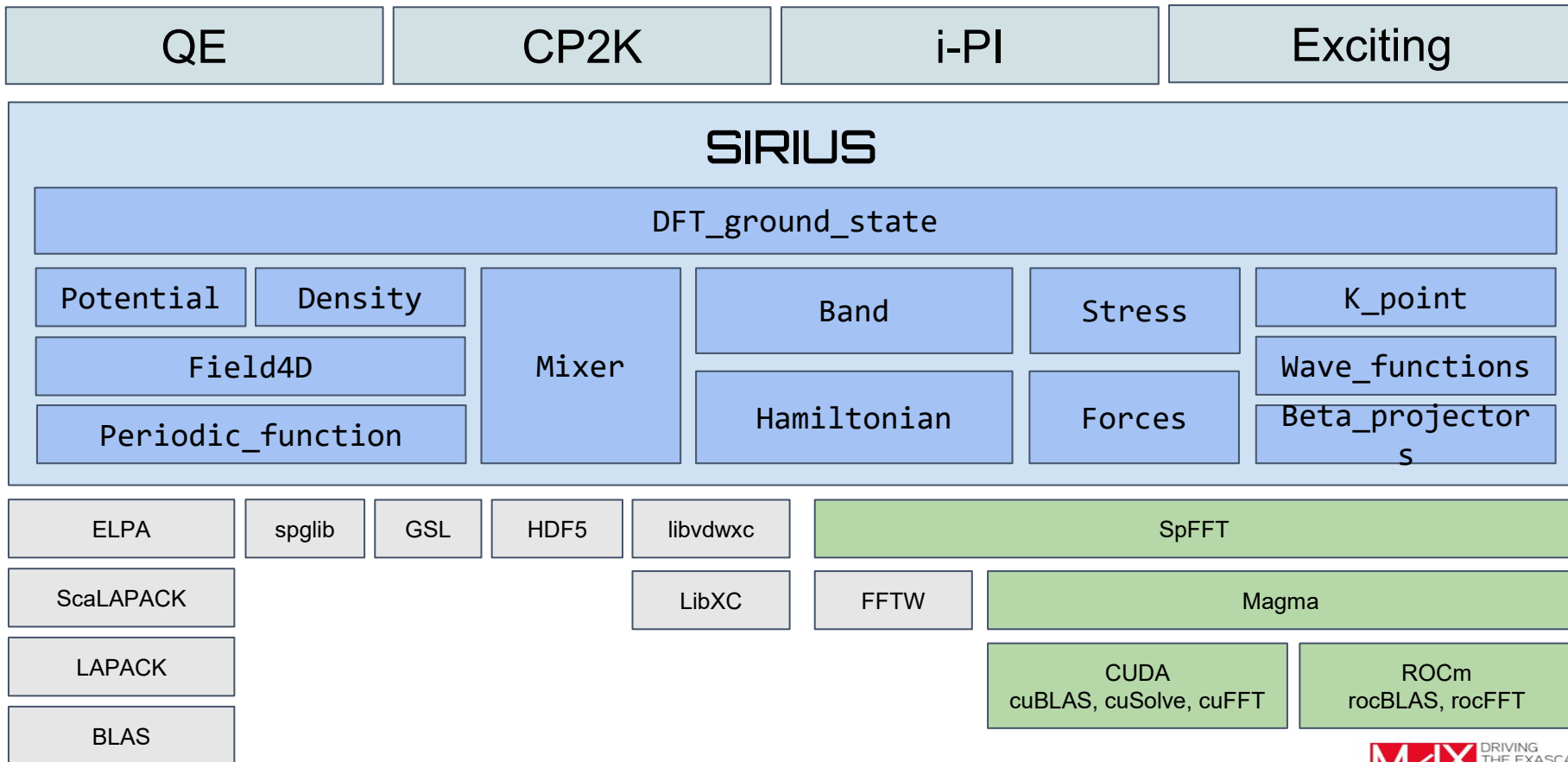
The following functionality is currently implemented in SIRIUS:

- (PP-PW) Norm-conserving, ultrasoft and PAW pseudopotentials
- (PP-PW) Spin-orbit coupling

Supported functionality

| Features | | |
|---|--|---|
| Pseudopotential | Full-potential | Common |
| <ul style="list-style-type: none">● NC/US/PAW pseudopotentials● Collinear and non-collinear magnetism● Hubbard U correction● Spin-orbit coupling● Stress tensor● Atomic forces● Verification of S-operator matrix● Iterative Davidson and exact diagonalization solvers● Orbital transformation (wave-function optimisation) method | <ul style="list-style-type: none">● L(A)PW+lo method with arbitrary number of local orbitals● Collinear and non-collinear magnetism with second variational approach● Iterative Davidson and exact diagonalization solvers● Spin-orbit coupling● Atomic forces | <ul style="list-style-type: none">● Python frontend● Symmetrization of lattice-periodic functions and on-site matrices (using symmetries from <i>spglib</i>)● Generation of k-point mesh using <i>spglib</i>● Run-time control of the eigenvalue solvers (Lapack / MAGMA / ScaLAPACK / ELPA)● Run-time control of the BLAS provider (CPU BLAS / cuBlas / cuBlasXt) |

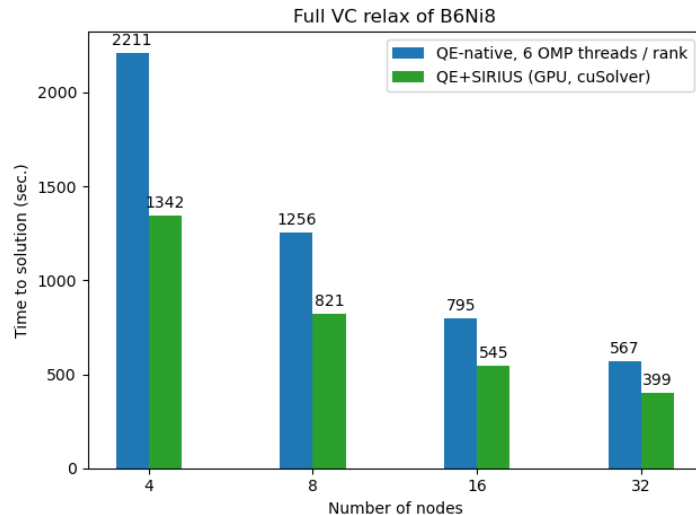
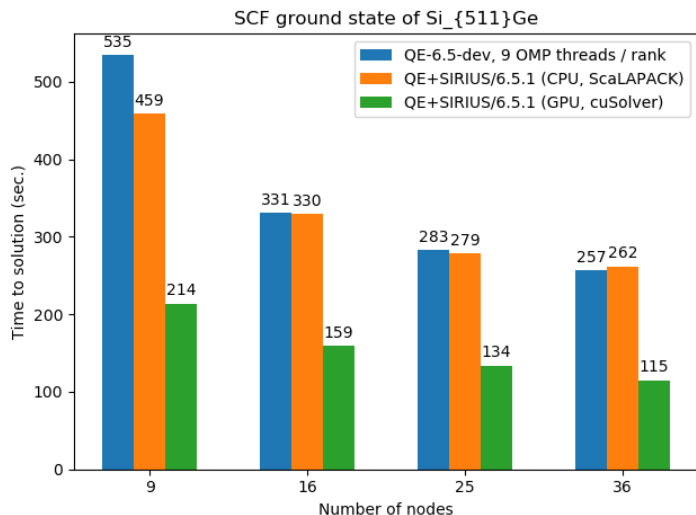
SIRIUS library design



GPU acceleration of Quantum ESPRESSO

<https://github.com/electronic-structure/q-e-sirius>

- Always in sync with main QE repository
- Used in production at CSCS



PW functionality in CP2K

cp2k / cp2k

Watch 42 Star 217 Fork 14

Code Issues 71 Pull requests 12 Actions Projects 1 Wiki Security 0 Insights

Releases Tags

Latest release

v7.1.0
40eef54

Compare

CP2K v7.1

oschuett released this on 28 Dec 2019 · 400 commits to master since this release

- **SIRIUS**: Plane Wave module with GPU support, see also [this tutorial](#) for Quantum ESPRESSO users.
- xTB: Tight-binding module based on [doi:10.1021/acs.jctc.7b00118](https://doi.org/10.1021/acs.jctc.7b00118).
- RPA / GW / MP2: migrated to DBCSR tensors.
- HELIUM: New canonical worm algorithm based on [doi:10.1103/PhysRevE.74.036701](https://doi.org/10.1103/PhysRevE.74.036701).
- XAS_TDP: X-ray absorption spectra simulations using linear-response TDDFT.
- NEGF: Contact-specific temperature, correct shift and scale factors.
- S-ALMO: Major refactoring, added wide variety of options.
- CDFT: Cleanup and bug fixing.
- FPGA interface for pw FFT.
- Updated libraries: DBCSR, ELPA, libint, libxc, libxsmm.
- The cp2k_shell was integrated into the main binary, simply call cp2k with `-s` or `--shell`.
- Development moved from SVN to Git.

LiF with UPF or GTH pseudopotentials

```
&FORCE_EVAL
  METHOD SIRIUS
  &PW_DFT
    &PARAMETERS
      ELECTRONIC_STRUCTURE_METHOD pseudopotential
      SMEARING_WIDTH 0.025
      USE_SYMMETRY true
      GK_CUTOFF 6.0
      PW_CUTOFF 20.00
      ENERGY_TOL 1e-6
      NGRIDK 2 2 2
    &END PARAMETERS
  &END PW_DFT
  &DFT
    &XC
      ...
    &END XC
  &END DFT
&END FORCE_EVAL
&SUBSYS
...
&END SUBSYS
&GLOBAL
...
&END GLOBAL
```

```
&SUBSYS
  &CELL
    A [bohr] 0.0 3.80402 3.80402
    B [bohr] 3.80402 0.0 3.80402
    C [bohr] 3.80402 3.80402 0.0
  &END CELL
  &COORD
    SCALED
    Li 0.0 0.0 0.0
    F 0.5 0.5 0.5
  &END COORD
  &KIND Li
    POTENTIAL UPF "Li.pz-s-kjpaw_psl.0.2.1.UPF.json"
  &END KIND
  &KIND F
    POTENTIAL GTH-LDA-q11
  &END KIND
&END SUBSYS
```

CP2K/SIRIUS output example

Charges and magnetic moments

total charge : 10.000000

Energy

valence_eval_sum : -4.33328910
<rho|V^{XC}> : -8.47838592
<rho|E^{XC}> : -7.00804097
<mag|B^{XC}> : 0.00000000
<rho|V^{H}> : 17.65751990
one-electron contribution : -13.47059366 (Ha), -26.94118732 (Ry)
hartree contribution : 8.82875995
xc contribution : -7.00804097
ewald contribution : -20.48430223
PAW contribution : -4.52435252
Total energy : -36.65852943 (Ha), -73.31705886 (Ry)

band gap (eV) : 9.07701505

Efermi : 0.22500000

iteration : 15, RMS 1.424969998675E-09, energy difference : 6.240623875442E-07

converged after 16 SCF iterations!

ENERGY| Total FORCE_EVAL (SIRIUS) energy (a.u.): -36.658529429377616

CP2K input reference

Section PW_DFT

DFT calculation using plane waves basis can be set in this section. The backend called SIRIUS, computes the basic properties of the system, such as ground state, forces and stresses tensors which can be used by cp2k afterwards. The engine has all these features build-in, support of pseudo-potentials and full-potentials, spin-orbit coupling, collinear and non collinear magnetism, Hubbard correction, all exchange functionals supported by libxc and Van der Waals corrections (libvdwxc). [Edit on GitHub]

Section path: CP2K_INPUT / FORCE_EVAL / PW_DFT

This section cannot be repeated.

Subsections

- CONTROL
- ITERATIVE_SOLVER
- MIXER
- PARAMETERS

Full documentation is available here:

https://manual.cp2k.org/cp2k-7_1-branch/CP2K_INPUT/FORCE_EVAL/PW_DFT.html



DRIVING THE EXASCALE TRANSITION

THANKS

Q&A