CP2K Developers Meeting

2025/02/24

Topics

Part 1 CP2K Developers Meeting

– Best practices for development and contributions

– New and Ongoing Developments

– Current Issues with CP2K

– Next CP2K Release

– Planned Events in the Context of CP2K

Part 2 GPU-Development with CP2K

– Portable-CUDA Concept

– Other GPU Programming Topics

– ...

Part 1: CP2K Developers Meeting

Part 1: CP2K Developers Meeting

Best practices for development and contributions (HansP, MathieuT, OleS)

- Mandatory reviews
  - Currently, core developers are dispensed

PROs:	speed and ease of contribution; "no burden for real science"
CONs:	Dashboard may break; (mandatory tests are fine)

- More mandatory tests

PROs:	... More tests are better
CONs:	Higher cost (cloud)

Notes

- Opinions/thoughts?
  - Ole: common practise, but cost effort; not enough manpower to review every request
  - Hans/Ole: not really a shortage of reviews right now
  - Rocco: possibility to request reviews would be helpful
  - Ole: maybe for developers groups can be used
  - Ole: three tests automatically run (code formatting, sdbg) and are mandatory
  - Hans/Ole: reviewers/core developers can launch additional tests like parallel
  - Hans: maybe randomization
  - Ole: Google Cloud cost in the green again
  - Robert: maybe run tests on HPC
  - Ole: one CSCS test is up and running
  - Ole: github actions with HPC systems
  - **Policy: always add new tests for new functionalities/algorithms or major changes**

Migration from Makefile to CMake

What‘s left to do:

- libgrpp (PR 3966)
- Debug builds
- GPU builds
- Other architectures: ARM, macOS, (might drop i386)
- Exotic builds (coverage, conventions, sanitizers)

Roadmap:

- 2025.2 release: Declare Makefile as deprecated
- After 2025.2 release:
  - Remove Makefile from master branch
  - Remove DBCSR submodule
  - Simplify Toolchain
- 2026.1 release: Only ships with CMake

Migration from DBCSR to DBM

What‘s done:

- Dropped single precision matrices
- Dropped complex matrices
- Moved high level routines to cp_dbcsr_contrib.F

What‘s left to do:

- Maybe refactor cp_blacs_env to use Cartesian MPI Communicators.
- Maybe merge mp_cart_type with mp_comm_type.
- Implement symmetric matrices.
- Implement replication.
- Refactor arnoldi so we can drop dbcsr_get_data_p()
- Refactor hfx_energy_potential.F so we can drop dbcsr_dot_threadsafe().
- Implement reading/writting CSR files.
- Implement reading/writting our custom binary files.

New and Ongoing Developments in CP2K

Developments@CASUS

- Finite-Temperature Random Phase Approximation (Frederick)
  - testing/fixing in progress
- Migrate Multigrid code to C/GPU
  - Application for OpenHackathon@Jülich in April (Frederick, Johann, Jiři)
  - needs to write a new FFT backend (GPL vs BSD license)
- Tblite interface to the GFN2-xTB method (Johann)
- OpenPMD as alternative IO method for reading/writing cube files in CP2K (Franz)

CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

www.casus.science

Integrating openPMD output/input into CP2K

Current effort at adding openPMD-based data handling for natively-parallel binary output based on HDF5/ADIOS2 according to a F.A.I.R. scientific data standard

HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

TECHNISCHE UNIVERSITÄT DRESDEN

CBG Max Planck Institute of Molecular Cell Biology and Genetics

HELMHOLTZ Centre for Environmental Research

Uniwersytet Wrocławski

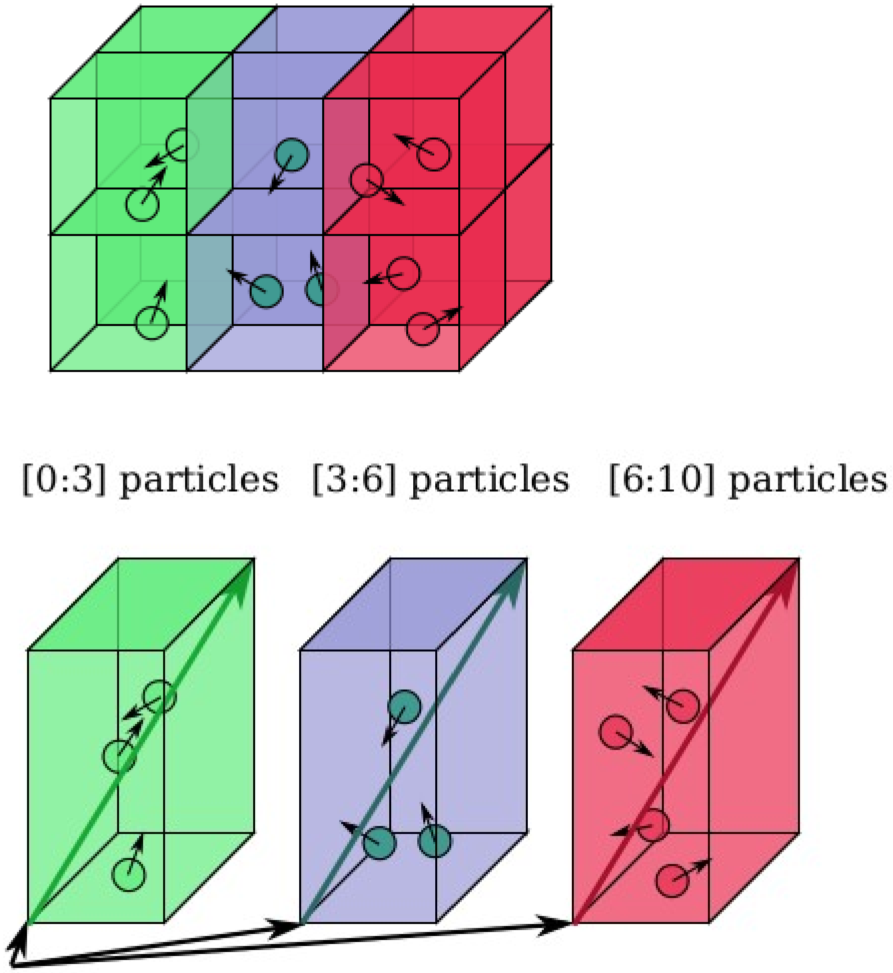
SPONSORED BY THE

Federal Ministry of Education and Research

STAATSMINISTERIUM FÜR WISSENSCHAFT KULTUR UND TOURISMUS

Freistaat SACHSEN

What is particle-mesh data?



Mesh

n-dimensional space,
divided into discrete cells

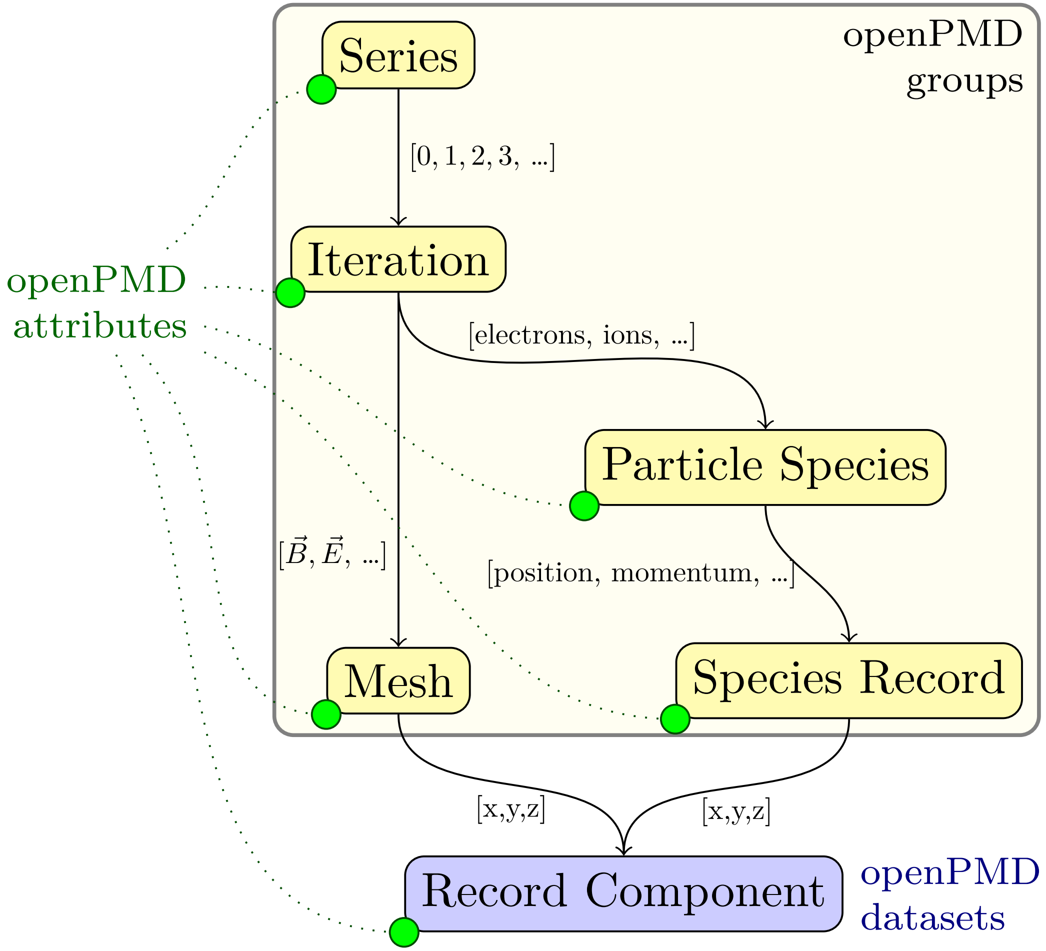
- e.g. temperature:
  store a scalar number per cell
- e.g. electrical fields:
  store a 3D vector per cell

Particles

A list of discrete objects,
located on the mesh

- for each particle: list its position
- optionally: list charge, weight, …

openPMD hierarchy

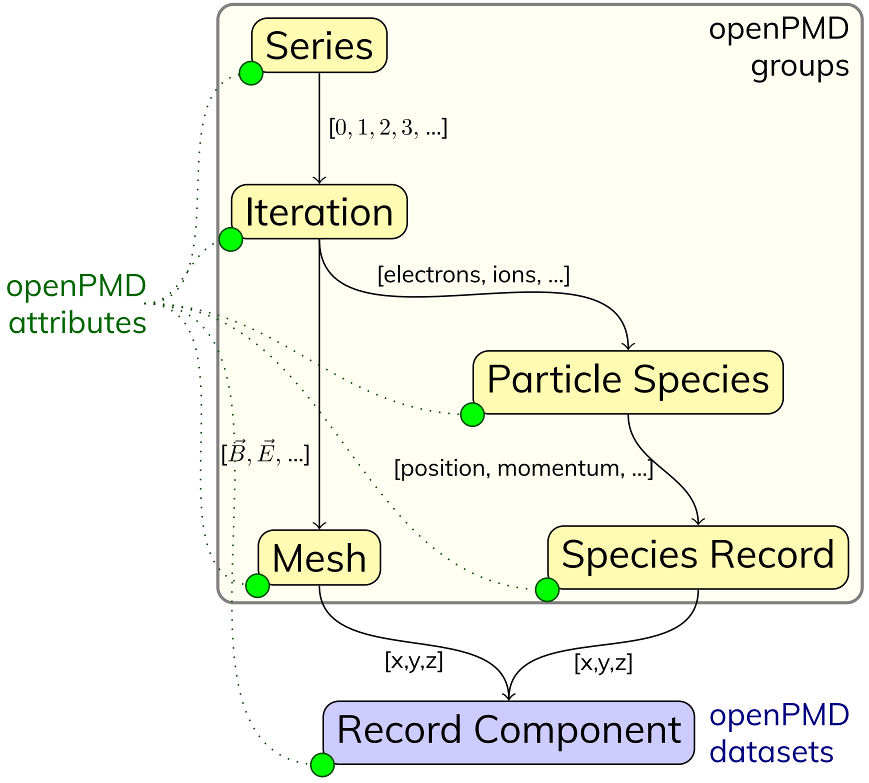


- **Structure** for series & snapshots encoded as either:
  - **files** (one file per iteration)
  - **groups** (reuse files)
  - **variables** (reuse files & variables in ADIOS2)

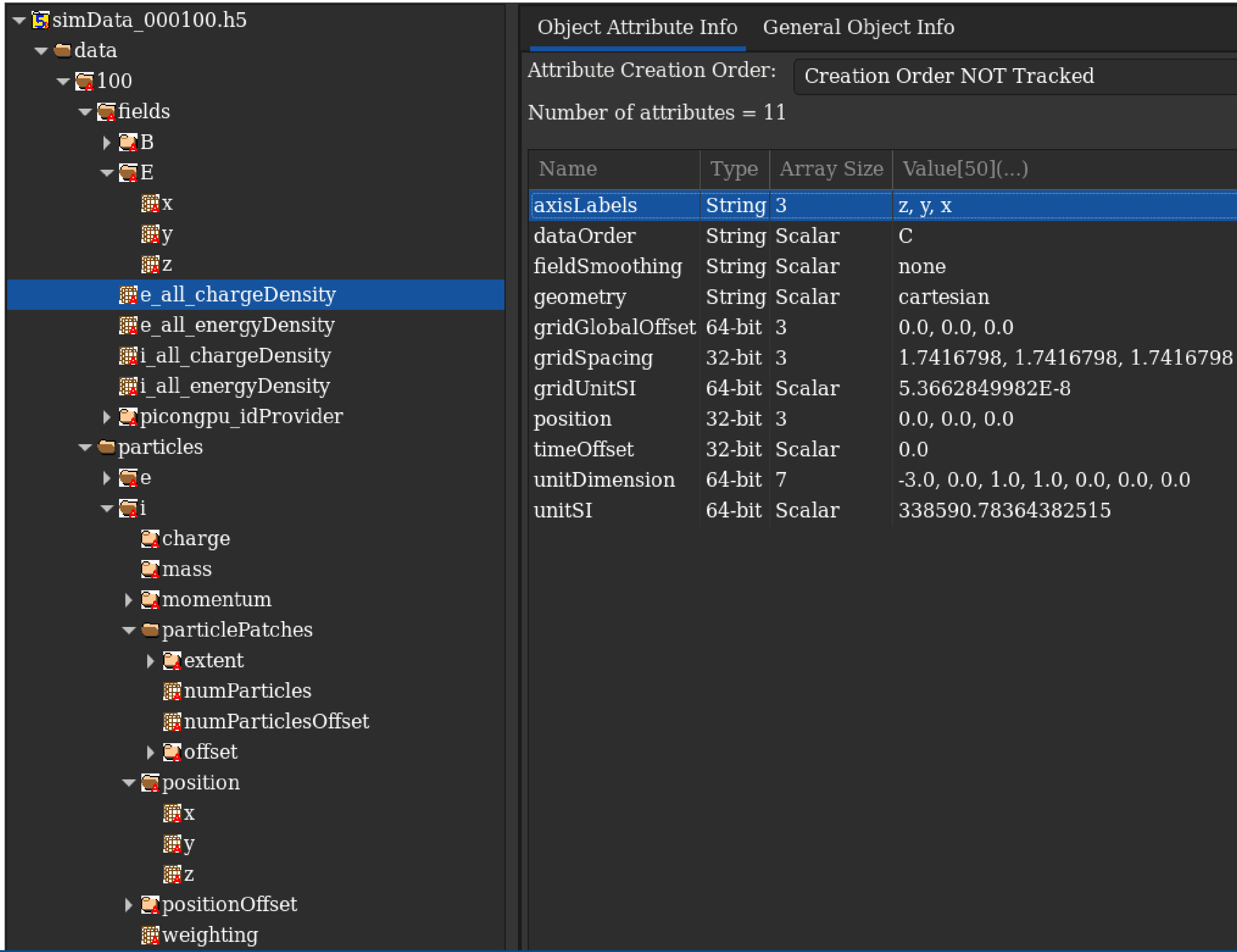
- Records for **physical observables**
  constants, mixed precision, complex numbers

- **Attributes:** unit conversion, description, relations, mesh geometry, authors, env. info, ...

Example dataset: HDF5 backend



Sample data created with PIConGPU



```
simData_000100.h5
  data
    100
      fields
        B
        E
          x
          y
          z
        e_all_chargeDensity
        e_all_energyDensity
        i_all_chargeDensity
        i_all_energyDensity
        picongpu_idProvider
      particles
        e
        i
          charge
          mass
          momentum
          particlePatches
            extent
            numParticles
            numParticlesOffset
            offset
          position
            x
            y
            z
          positionOffset
          weighting
```

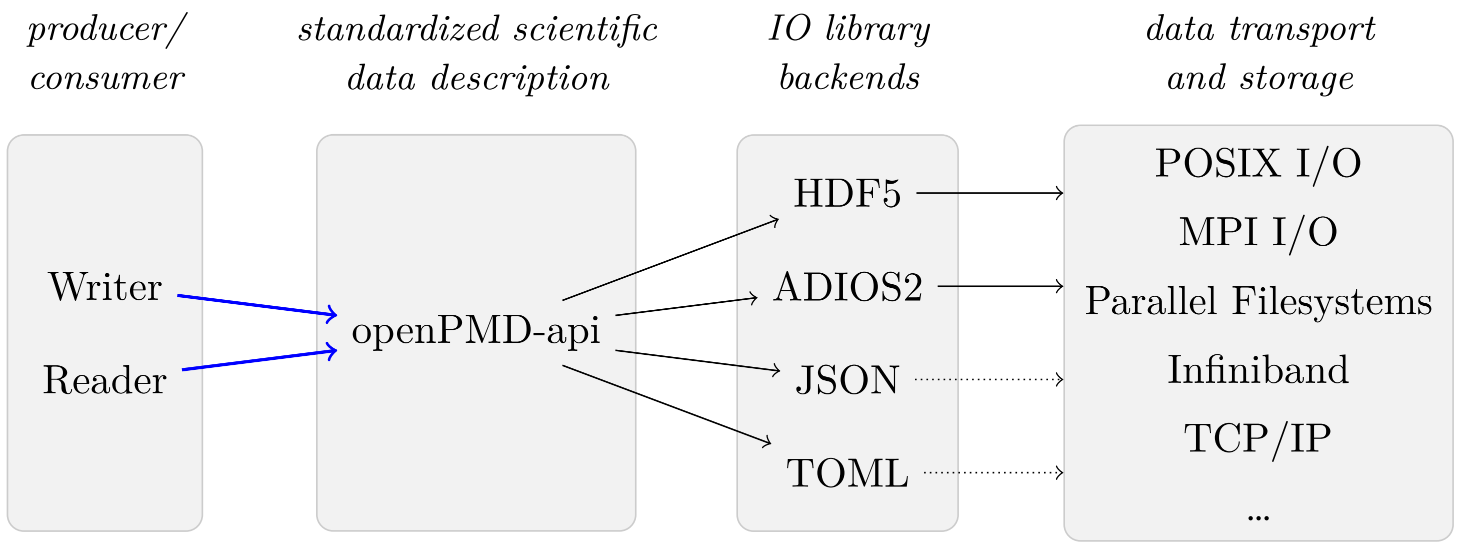
Object Attribute Info | General Object Info

Attribute Creation Order: Creation Order NOT Tracked

Number of attributes = 11

Name	Type	Array Size	Value[50](...)
axisLabels	String	3	z, y, x
dataOrder	String	Scalar	C
fieldSmoothing	String	Scalar	none
geometry	String	Scalar	cartesian
gridGlobalOffset	64-bit	3	0.0, 0.0, 0.0
gridSpacing	32-bit	3	1.7416798, 1.7416798, 1.7416798
gridUnitSI	64-bit	Scalar	5.3662849982E-8
position	32-bit	3	0.0, 0.0, 0.0
timeOffset	32-bit	Scalar	0.0
unitDimension	64-bit	7	-3.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0
unitSI	64-bit	Scalar	338590.78364382515

openPMD-api – open stack for scientific I/O



producer/ consumer

standardized scientific data description

IO library backends

data transport and storage

Writer

Reader

openPMD-api

HDF5

ADIOS2

JSON

TOML

POSIX I/O

MPI I/O

Parallel Filesystems

Infiniband

TCP/IP

...

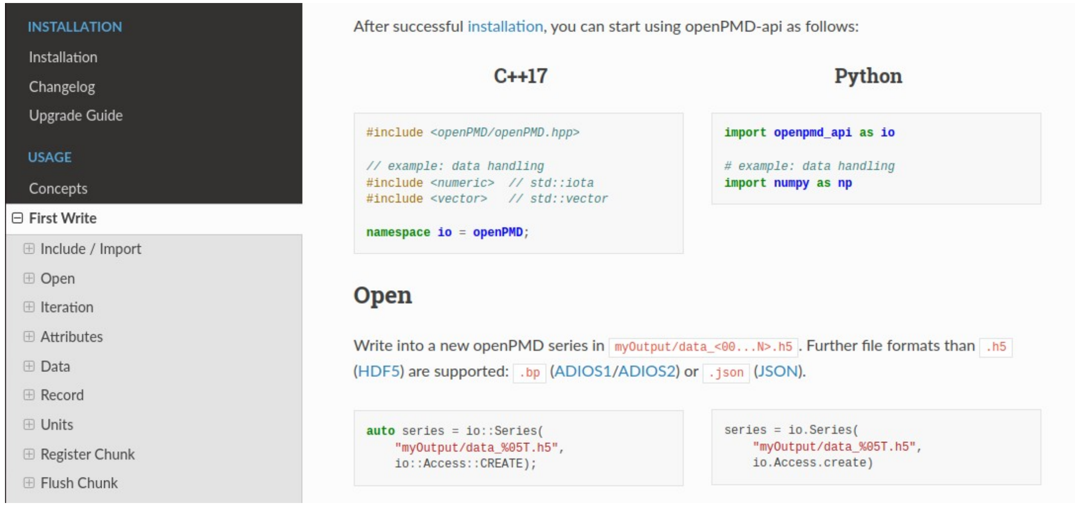
```python
import openpmd_api as io

# pick and configure backend via JSON/TOML or inferred from filename extension
adios_config = """
  backend = "adios2"
  [[adios2.dataset.operators]]
  type = "blosc" # activate compression
"""
mode = io.Access.create
series = io.Series("simOutput.h5",   mode
                   """{"hdf5": {"vfd": {"type": "subfiling"}}}""")
series = io.Series("simOutput.bp5",  mode, adios_config)
series = io.Series("simOutput.sst",  mode, "@./or/load/config/from/file.json")
series = io.Series("simOutput.json", mode)
```

- MPI support at all levels
- Implemented in C++17
- Bindings in C++17, Python and (dev version only) Julia
- Specify backend at runtime:
  I/O library, transport, compression, streaming, aggregation, …

Reference Implementation in C++ & Bindings: Python and Julia

CASUS
CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

Online Documentation: openpmd-api.readthedocs.io



After successful installation, you can start using openPMD-api as follows:

C++17

```
#include <openPMD/openPMD.hpp>

// example: data handling
#include <numeric>  // std::iota
#include <vector>   // std::vector

namespace io = openPMD;
```

Python

```
import openpmd_api as io

# example: data handling
import numpy as np
```

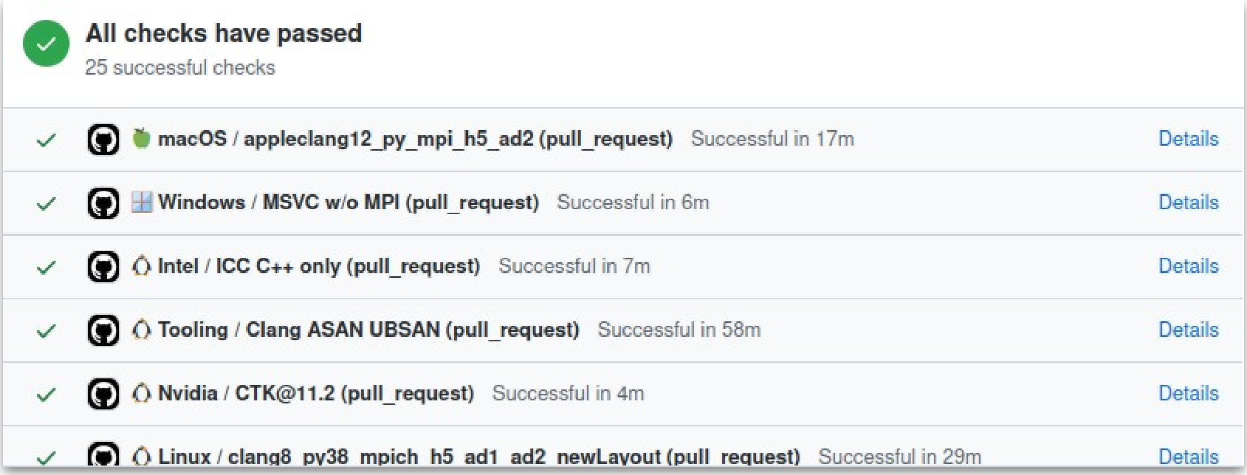
Open

Write into a new openPMD series in `myOutput/data_<00...N>.h5`. Further file formats than `.h5` (HDF5) are supported: `.bp` (ADIOS1/ADIOS2) or `.json` (JSON).

```
auto series = io::Series(
    "myOutput/data_%05T.h5",
    io::Access::CREATE);
```

```
series = io.Series(
    "myOutput/data_%05T.h5",
    io.Access.create)
```

Open-Source Development & Tests: github.com/openPMD/openPMD-api



All checks have passed
25 successful checks

✓ macOS / appleclang12_py_mpi_h5_ad2 (pull_request) Successful in 17m Details
✓ Windows / MSVC w/o MPI (pull_request) Successful in 6m Details
✓ Intel / ICC C++ only (pull_request) Successful in 7m Details
✓ Tooling / Clang ASAN UBSAN (pull_request) Successful in 58m Details
✓ Nvidia / CTK@11.2 (pull_request) Successful in 4m Details
✓ Linux / clang8_py38_mpich_h5_ad1_ad2_newLayout (pull_request) Successful in 29m Details

Rapid and easy installation on any platform:

```
python3 -m pip install
        openpmd-api
```

```
brew tap openpmd/openpmd
brew install openpmd-api
```

```
cmake -S . -B build
cmake --build build
    --target install
```

```
conda install
    -c conda-forge
    openpmd-api
```

```
spack install
      openpmd-api
```

```
module load openpmd-api
```

A Huebl, F Poeschel, F Koller, J Gu, et al.
"openPMD-api: C++ & Python API for Scientific I/O with openPMD" (2018) DOI:10.14278/rodare.27

openPMD powered Projects and Users

Documents:

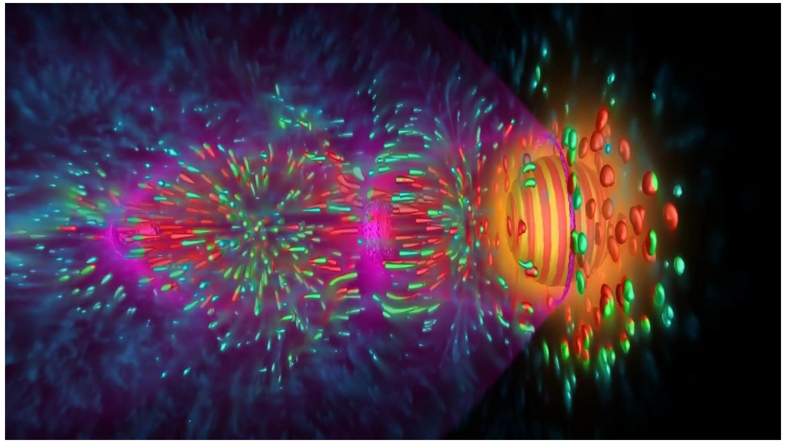
- **openPMD standard** (1.0.0, 1.0.1, 1.1.0)
  the underlying file markup and definition
  A Huebl et al., doi: 10.5281/zenodo.33624

Language Binding:

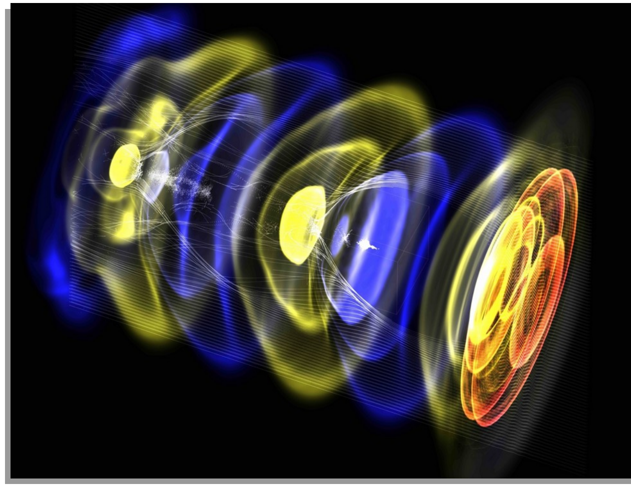
- **openPMD-api** (HZDR, CASUS, LBNL)
  reference API for openPMD data handling
  maintainers: A Huebl, J Gu, F Poeschel et al.

Scientific Simulations:

- **PIConGPU** (HZDR)
  electro-dynamic particle-in-cell code
  maintainers: R Widera, S Bastrakov, A Debus et al.
- **WarpX** (LBNL, LLNL)
  electro-dynamic/static particle-in-cell code
  maintainers: JL Vay, D Grote, R Lehe, A Huebl et al.
- **FBPIC** (LBNL, DESY)
  spectral, fourier-bessel particle-in-cell code
  maintainers: R Lehe, M Kirchen et al.
- **SimEx Platform** (EUCALL, European XFEL)
  simulation of advanced photon experiments
  maintainer: C Fortmann-Grote



PIConGPU+ISAAC on Summit
2nd prize Helmholtz Imaging
Best Scientific Image Contest 2022
Image credit: Felix Meyer/HZDR



WarpX
PI: Jean-Luc Vay/LBNL

see also: https://github.com/openPMD/openPMD-projects

openPMD powered Projects and Users

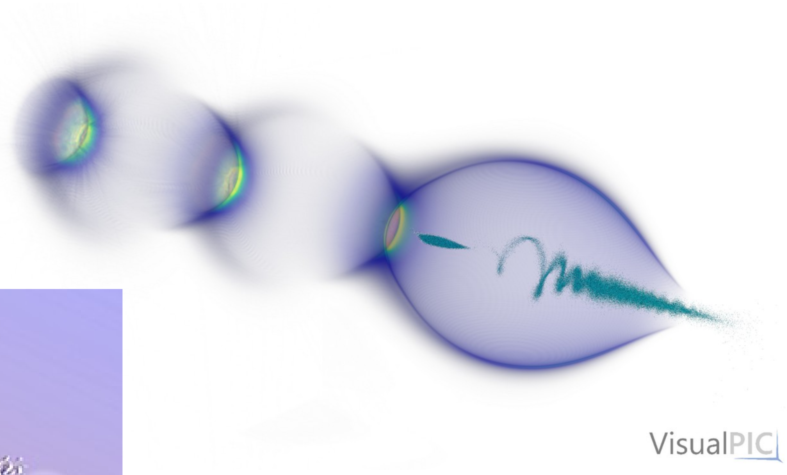
Documents:

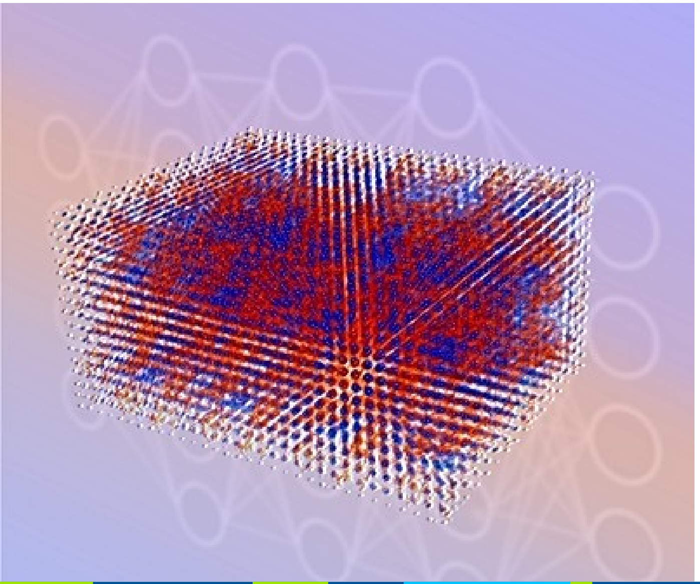
- **openPMD standard** (1.0.0, 1.0.1, 1.1.0)
  the underlying file markup and definition
  A Huebl et al., doi: 10.5281/zenodo.33624

Language Binding:

- **openPMD-api** (HZDR, CASUS, LBNL)
  reference API for openPMD data handling
  maintainers: A Huebl, J Gu, F Poeschel et al.

HiPACE++ → VisualPIC
Credit: M.Thévenet
& A. Ferran Pousa (DESY)



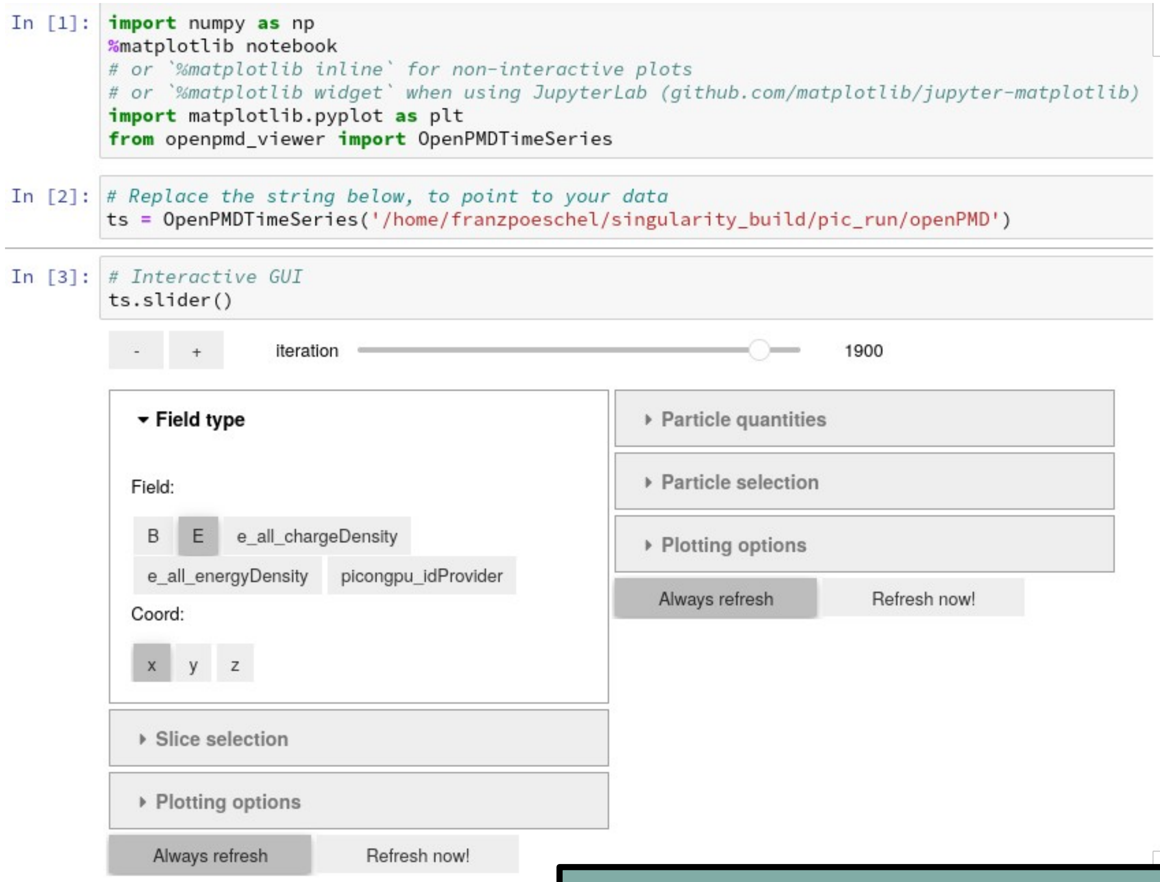


MALA → ParaView
Credit: A. Cangi (CASUS)

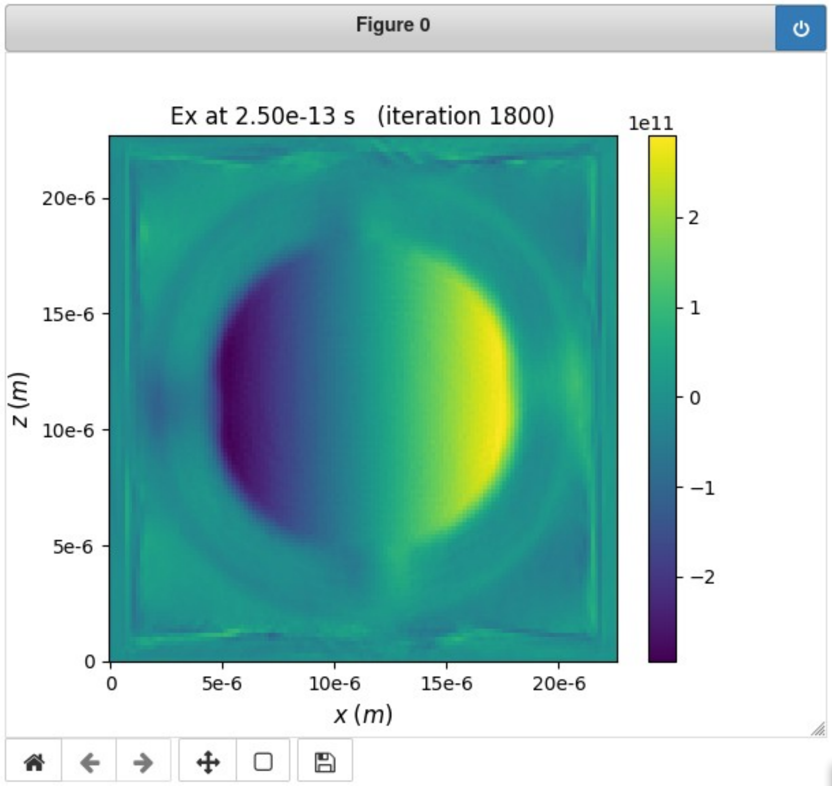
- **Wake-T** (DESY)
  fast particle-tracking code for plasma-based accelerators
  maintainer: A Ferran Pousa
- **HiPACE++** (DESY, LBNL)
  3D GPU-capable quasi-static PIC code for plasma accel.
  maintainers: M Thevenet, S Diederichs, A Huebl
- **Bmad** (Cornell)
  library for charged-particle dynamics simulations
  maintainers: D Sagan et al.
- **MALA** (CASUS, SNL)
  ML models that replace DFT calculations in materials science
  maintainers: Attila Cangi & Sivasankaran Rajamanickam
- and more…

see also: https://github.com/openPMD/openPMD-projects

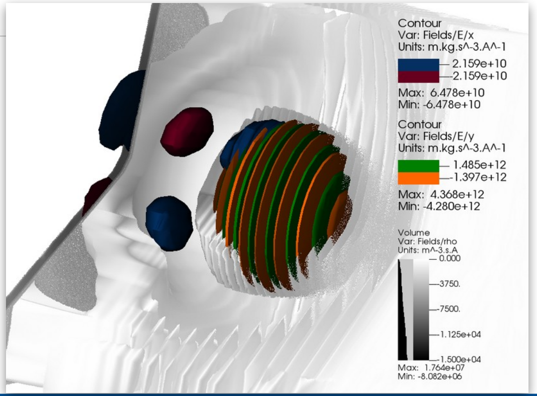
Analysis and Visualization



```
In [1]: import numpy as np
        %matplotlib notebook
        # or `%matplotlib inline` for non-interactive plots
        # or `%matplotlib widget` when using JupyterLab (github.com/matplotlib/jupyter-matplotlib)
        import matplotlib.pyplot as plt
        from openpmd_viewer import OpenPMDTimeSeries

In [2]: # Replace the string below, to point to your data
        ts = OpenPMDTimeSeries('/home/franzpoeschel/singularity_build/pic_run/openPMD')

In [3]: # Interactive GUI
        ts.slider()
```



openPMD/openPMD-viewer



Standardization of data

→ integration into modern scientific compute workflows

RAPIDS pandas openPMD DASK ParaView yt VisIt

Current status and Todo

Done:

- Added a Fortran module openPMD.F to bind to the C++ API via C
- Modified cp_output_handling.F module to support creation of openPMD files instead of Cube
- Challenges:
  - openPMD has an internal structure, hence one openPMD file corresponds to multiple Cube files
    → Need to distinguish callsites
  - Streaming support requires a workflow where IO handles stay open in the background
  - Representation for nested Iterations

To do:

- Actually write n-dimensional output
  (in realspace_grid_cube.F)
- Parallel output (best-case scenario: trivial)
- Input reading from openPMD
- Runtime configuration via input files
- Conditional compilation
  (openPMD as an optional dependency)
- Add openPMD output to modules other than src/qs_scf_post_gpw.F
  Should be simple once the main logic stands.
- Testing, tooling (e.g. conversion Cube ←→ openPMD)

Thomas Kühne/CASUS

- Sigma-RPA (Görling) implementation nearly done, benchmark tests currently ongoing
- MACE-potential meeting -> periodic RPA calculations (CP2K) needed to train networks
- HPC events:
  - NVIDIA GPU-event
  - Jülich recently
- (cusolvermp generalized eigenvalue solver)
- Caution for ELPA: make sure that you use
  - CPU: 2-stage solvers
  - GPU:1-stage solvers
  - see also https://manual.cp2k.org/trunk/CP2K_INPUT/FORCE_EVAL/PW_DFT/CONTROL.html and benchmark for your case!
- Announcement:
  - PostDoc position open in Stefan Grimme‘s group

„Traditional“ Diagonalization

$$\mathbf{K}\mathbf{c} = \mathbf{S}\mathbf{c}\epsilon$$

$$\boldsymbol{K}\,\mathbf{c} = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}\,\boldsymbol{c}\,\epsilon$$

$$(\boldsymbol{U}^{\mathrm{T}})^{-1}\,\boldsymbol{K}\,\boldsymbol{U}^{-1}\,\boldsymbol{c}' = \boldsymbol{c}'\,\epsilon$$

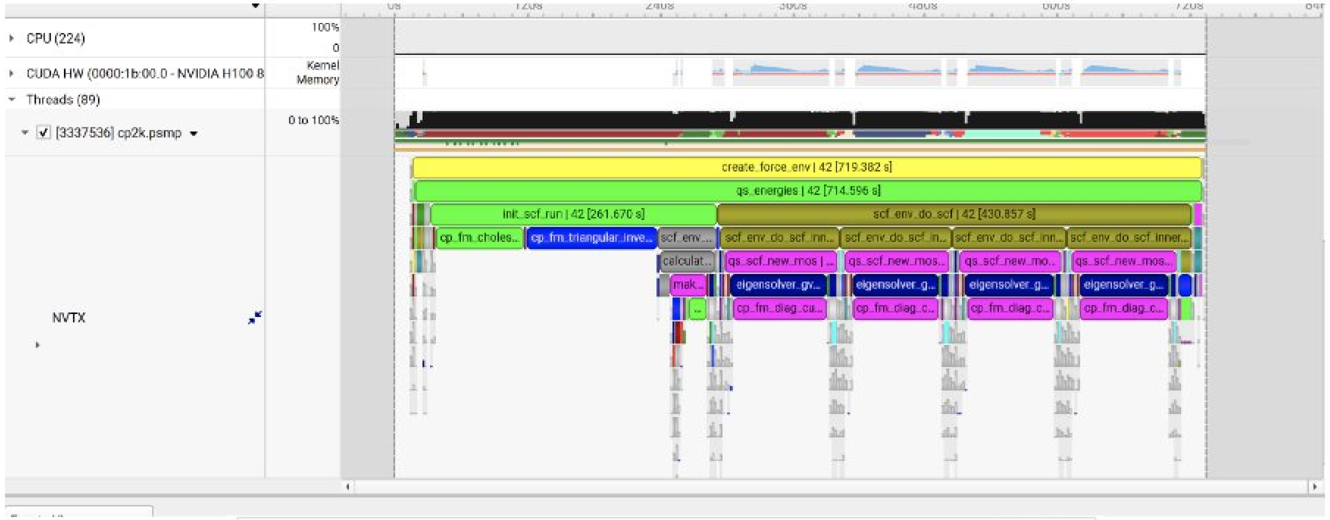
(`pdsygst`)

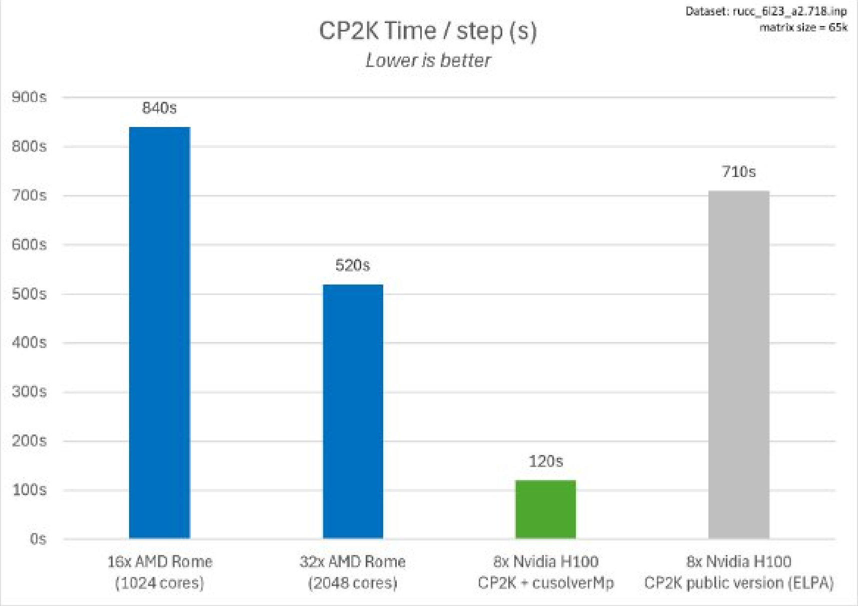
$$\boldsymbol{K}'\,\boldsymbol{c}' = \boldsymbol{c}'\,\epsilon$$

(`pdsyevx` or `pdsyevd`)

$$\begin{aligned} \mathbf{c} &= \mathbf{U}^{-1}\mathbf{c}' \quad \text{or} \\ \mathbf{c} &= \mathbf{S}^{-1/2}\mathbf{c}' \end{aligned}$$

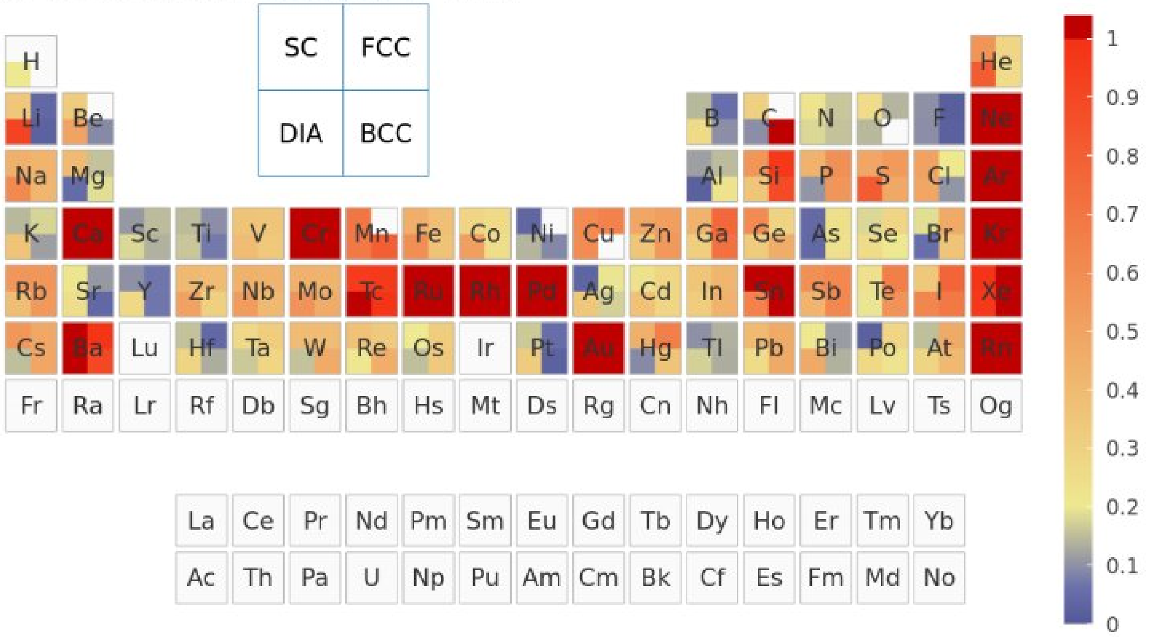
„Traditional“ Diagonalization



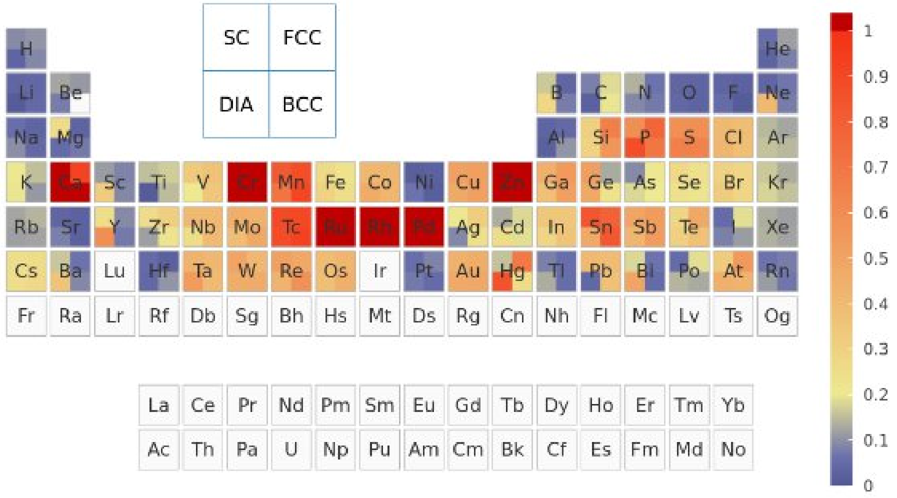


Thomas Kühne
CASUS

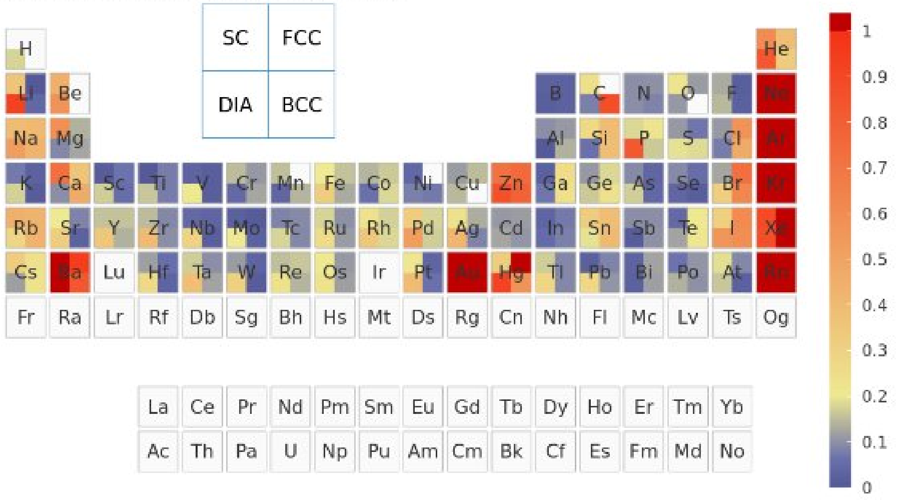
ε for CP2K-GTH-UZH vs. SIRIUS-FP-LAPW



ε for SIRIUS-GTH-UZH vs. SIRIUS-FP-LAPW

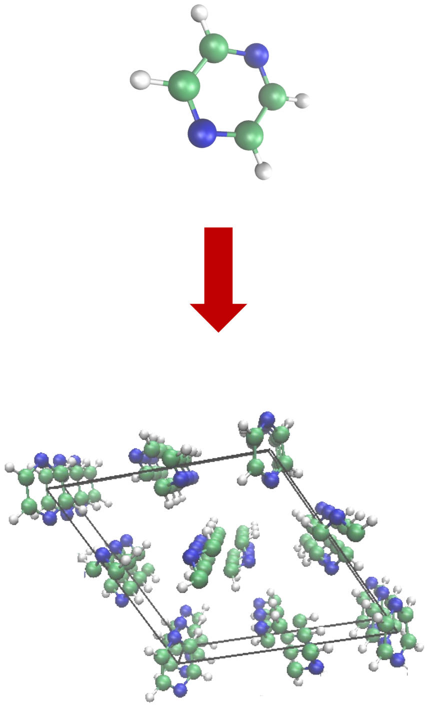


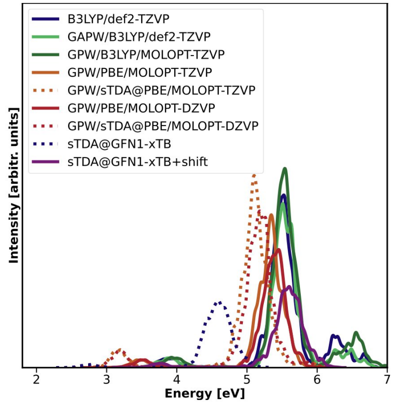
ε for CP2K-GTH-UZH vs. SIRIUS-GTH-UZH

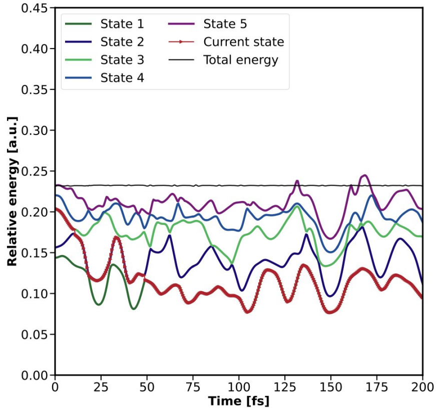


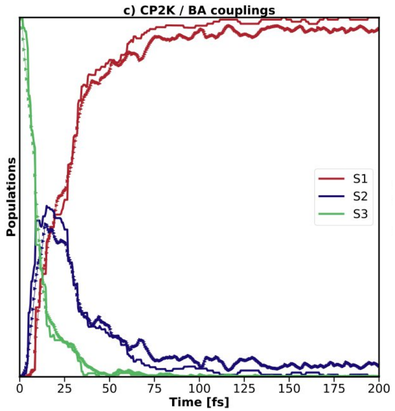
Ongoing developments in Kiel

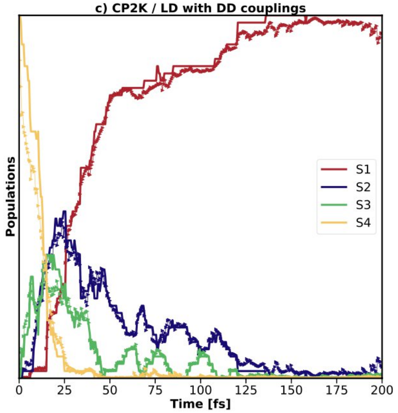
- **Non-adiabatic molecular dynamics** relying on semi-empirical or fast numerical time derivative couplings or local diabatization
- **Smeared occupation** for time-dependent density functional theory ansätze to capture static correlation (based on different distribution functions)
- **Simplified Bethe-Salpeter equation** and multipole expansions for GFN1-xTB (CRC proposal)

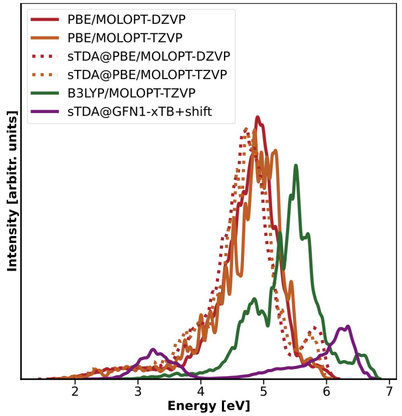


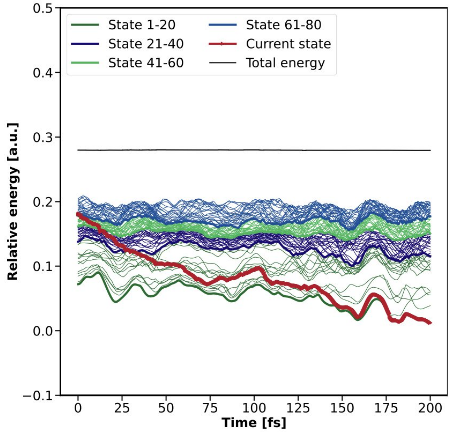


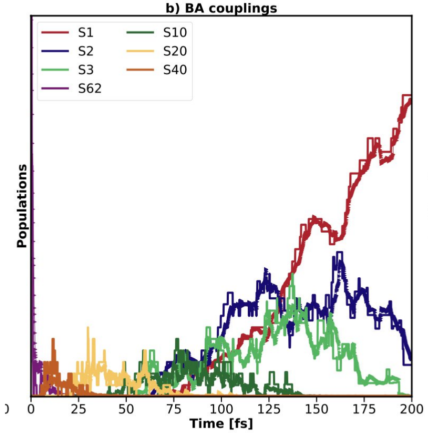


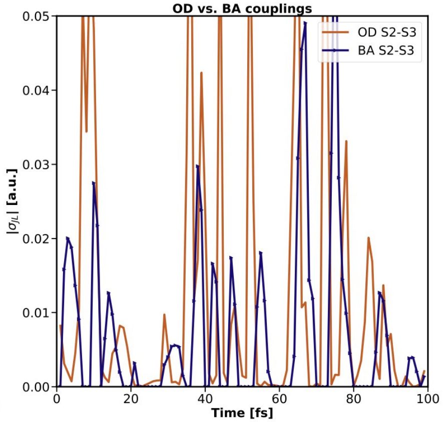












Current Issues in CP2K

- PR #4000: significant perf. overhead if F2K8+ compliant behaviour like copy/assignment (re-)allocation
  - Issue appeared with IFX but after fix, GNU had significant benefit too
  - Related to structures with allocatable components
- OpenMP workshare incorrect in almost all compiler (https://github.com/cp2k/dbcsr/issues/857#issuecomment-2511098676)

Next CP2K Release

- Schedule: Summer 2025
  - Makefile deprecation
  - CMake support

Planned Events in the Context of CP2K

- International Summer School on CP2K-GROMACS for Multiscale Atomistic Simulation
  - tentative date: 4 days in KW 40 (29.9. - 2.10.) at Uni-Paderborn (depends on availability of room)
  - tentative programs: lectures + hands-on exercises + posters from participants
  - tentative schedule:
    - day 1: GROMACS and MD Simulation
    - day 2: CP2K, Intro to QM/MM, CP2K/GROMACS QM/MM Simulation
    - day 3: IC-QM/MM and Post-DFT in CP2K
    - day 4: GROMACS on GPUs and MiMiC

Planned Events in the Context of CP2K

- next LUMI hackathon (Oslo): https://lumi-supercomputer.eu/events/lumi-hackathon-spring2025/
- next-to-next hackathon (CSCS): November
- Juelich Mimic Summer school: 1st week of June, CECAM+Psi-k https://www.cecam.org/workshop-details/multiscale-molecular-dynamics-with-mimic-optimizing-the-performance-on-modern-supercomputers-1397

Part 2: GPU Development in CP2K

1. Portable CUDA Concept (Ole Schütt)
2. Other GPU Programming Topics

Portable CUDA Concept (Ole Schütt)

- Use the subset of CUDA that's also supported by HIP.
- Use our offload API to switch between runtimes.
- Full code sharing between Nvidia and AMD.
- Partial code sharing with OpenCL / Intel.
- Partial code sharing with CPU.
- Simple, robust, and future proof.
- Successfully in use for grid, DBM, and pw.

Why doesn't anyone else advertise this? (speculations)

- It's boring. CUDA has been around since 2006.
- GPU manufacturers prefer solutions with vendor lock-in.
- Computing centers don't like to admit that GPUs require large rewrites.

Comments

- https://x-dev.pages.jsc.fz-juelich.de/models/
- Hipfly (header based translation) approach where one can keep their CUDA code and translate to HIP at compile time: https://github.com/amd/HPCTrainingExamples/tree/main/hipifly/vector_add
- Do concurrent: depends a lot on the compiler support

Other GPU Programming Topics (HansP)

- __OFFLOAD_UNIFIED_MEMORY

  PROs

  - Enables more GPU usage (finer granularity)?

  CONs

  - Less explicit/general compared to assuming descrete memory spaces
  - Unclear performance status wrt level of hardware support

  Optional

  - Asks to fold host and device pointers (code and data structures affected)
  - Currently, H2D and D2H are no-ops, only “host“-pointers are allocated

- How about GPU-CPU hybrid computations?
  - For example, DBM could use both CPU and GPU...

Comments

- Ongoing work for MI300A, grid code already ported to unified memory