



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH**zürich



# **SIRIUS: a GPU accelerated electronic-structure DFT library**

PerfLab seminar series at NHR@FAU

Anton Kozhevnikov, CSCS

February 11, 2025

# Table of Contents

1. Brief overview of CSCS and its activities in scientific software and libraries development
2. SIRIUS: domain-specific electronic structure library
3. A quick introduction to the most important backend libraries of SIRIUS
4. Q & A



**CSCS**

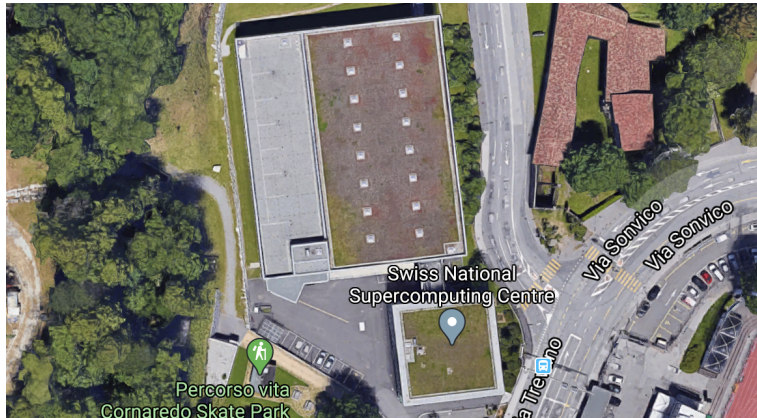
Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# Brief overview of CSCS and its activities in scientific software and libraries development

---

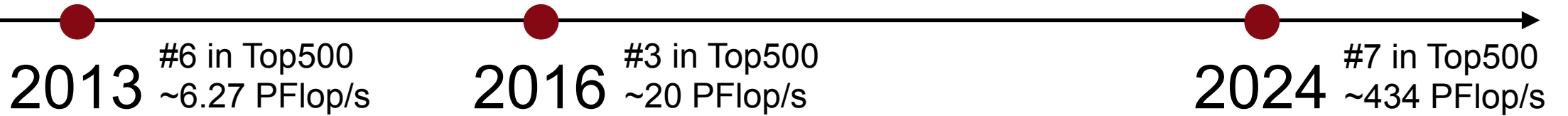
# Swiss National Supercomputing Centre – CSCS



CSCS develops and operates a high performance computing and data research infrastructure that supports world-class science in Switzerland



# Accelerated computing at CSCS



	K20X	P100	GH200
<b>Peak performance</b>	1.17 TFlop/s	4.8 TFlop/s	34 TFlop/s (67 with TC)
<b>GPU memory</b>	6 Gb	16 Gb	96 Gb
<b>GPU memory BW</b>	250 Gb/s	732 Gb/s	4000 Gb/s
<b>H-D transfer speed</b>	32 Gb/s	32 Gb/s	900 Gb/s

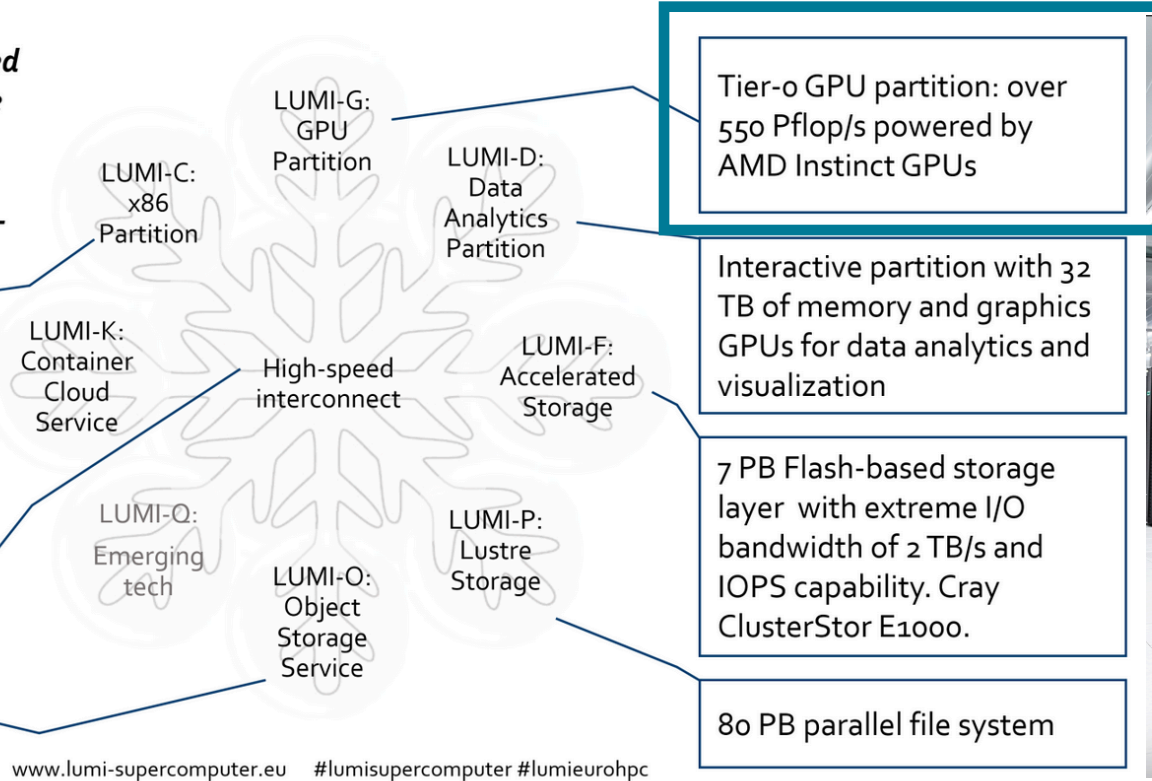
# ETH Zürich is a swiss partner of LUMI consortium

LUMI is a Tier-0 GPU-accelerated **supercomputer** that enables the convergence of **high-performance computing, artificial intelligence, and high-performance data analytics.**

- Supplementary CPU partition
- ~200,000 AMD EPYC CPU cores

Possibility for combining different resources within a single run. HPE Slingshot technology.

30 PB encrypted object storage (Ceph) for storing, sharing and staging data



Tier-0 GPU partition: over 550 Pflop/s powered by AMD Instinct GPUs

Interactive partition with 32 TB of memory and graphics GPUs for data analytics and visualization

7 PB Flash-based storage layer with extreme I/O bandwidth of 2 TB/s and IOPS capability. Cray ClusterStor E1000.

80 PB parallel file system



2019

# Platform for Advanced Scientific Computing - PASC

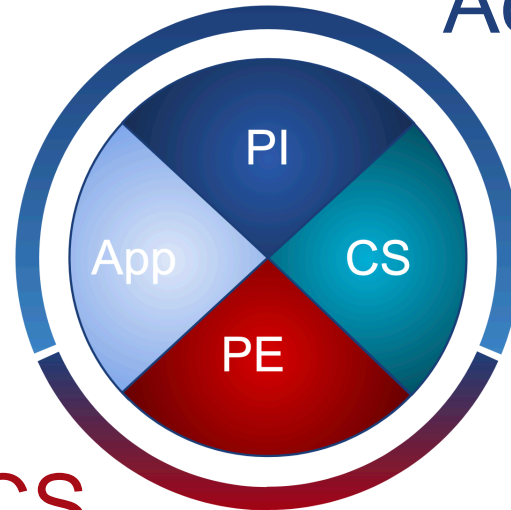
**PI**

scientific vision and goals

Academia

**Computational scientist**

translation of formulae to algorithms



CSCS

**Application scientist**

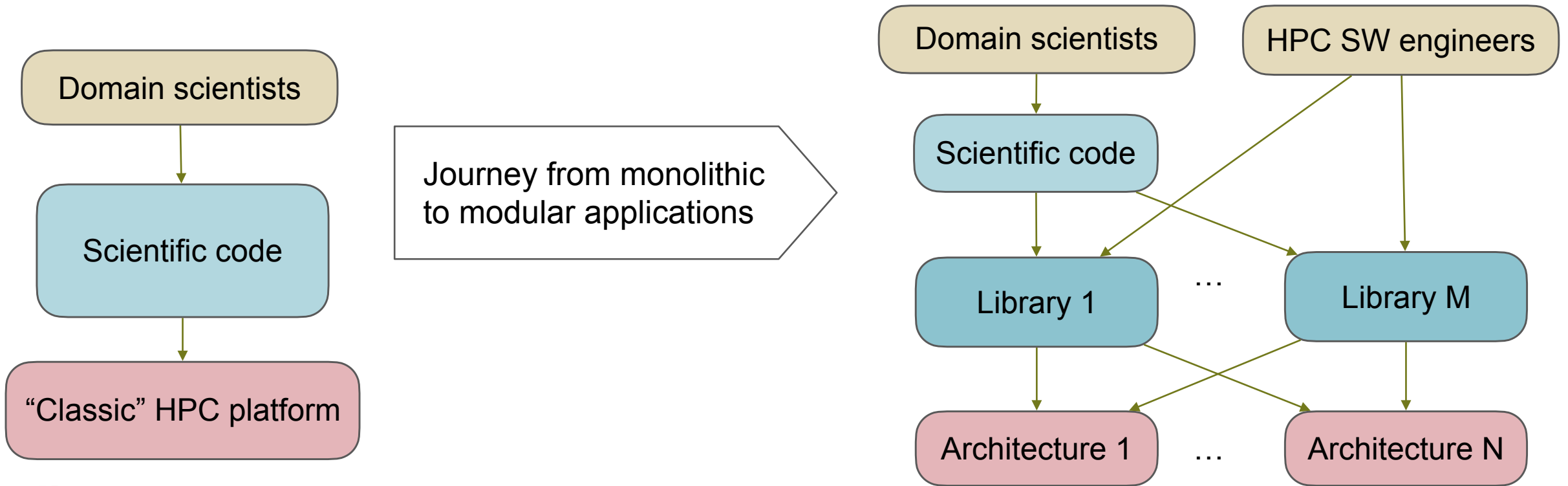
input cases, analysis and interpretation of results

**Performance and SW engineer**

Parallelization, hardware specific know-how (GPUs), development best practices

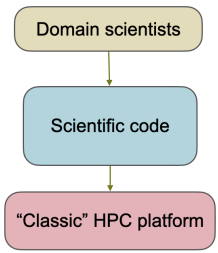
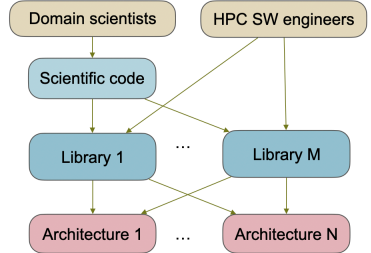
# Scientific software development at CSCS

CSCS vision: complexity of current and emerging HPC platforms and programming models should be reflected in the way we develop scientific software. Encapsulation of common, reusable components of the large scientific codes into domain specific libraries leads to a better software engineering of such codes.





# Pros and cons of monolithic and modular applications

	 <pre> graph TD     A[Domain scientists] --&gt; B[Scientific code]     B --&gt; C["Classic" HPC platform]         </pre>	 <pre> graph TD     DS[Domain scientists] --&gt; SC[Scientific code]     HSW[HPC SW engineers] --&gt; L1[Library 1]     HSW --&gt; LM[Library M]     SC --&gt; L1     SC --&gt; LM     L1 --&gt; A1[Architecture 1]     L1 --&gt; AN[Architecture N]     LM --&gt; A1     LM --&gt; AN         </pre>
Pros	<ul style="list-style-type: none"> <li>• usually works well on your “home” cluster</li> <li>• few dependencies</li> <li>• full control of the source code</li> </ul>	<ul style="list-style-type: none"> <li>• separation of concerns</li> <li>• reusability of code</li> <li>• less own code to maintain</li> </ul>
Cons	<ul style="list-style-type: none"> <li>• supporting multiple architectures is close to impossible</li> <li>• hard to maintain and on-board new developers</li> <li>• often a hacky build process on other platforms and environments</li> </ul>	<ul style="list-style-type: none"> <li>• long-term support of libraries</li> <li>• lag in bug fixes</li> <li>• requesting new features takes time</li> <li>• need to install many dependencies, which is hard to do manually</li> </ul>

# Common traits for CSCS software development

- Use C++ as a programming language
- Use CUDA/ROCM or Kokkos as a GPU programming model
- Use **CMake** as a build system
- Use **spack** to build dependencies
- CI/CD pipeline that runs both on Github/Gitlab VMs and at CSCS

Each of our libraries:

- supports CMake/spack
- provides C/Fortran API, examples and documentation



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# **SIRIUS: domain-specific electronic structure library**

---

# “Delta DFT” effort

## DFT METHODS

## Reproducibility in density functional theory calculations of solids

Kurt Lejaeghere,<sup>1\*</sup> Gustav Bihlmayer,<sup>2</sup> Torbjörn Björkman,<sup>3,4</sup> Peter Blaha,<sup>5</sup> Stefan Blügel,<sup>2</sup> Volker Blum,<sup>6</sup> Damien Caliste,<sup>7,8</sup> Ivano E. Castelli,<sup>9</sup> Stewart J. Clark,<sup>10</sup> Andrea Dal Corso,<sup>11</sup> Stefano de Gironcoli,<sup>11</sup> Thierry Deutsch,<sup>7,8</sup> John Kay Dewhurst,<sup>12</sup> Igor Di Marco,<sup>13</sup> Claudia Draxl,<sup>14,15</sup> Marcin Dułak,<sup>16</sup> Olle Eriksson,<sup>13</sup> José A. Flores-Livas,<sup>12</sup> Kevin F. Garrity,<sup>17</sup> Luigi Genovese,<sup>7,8</sup> Paolo Giannozzi,<sup>18</sup> Matteo Giantomassi,<sup>19</sup> Stefan Goedecker,<sup>20</sup> Xavier Gonze,<sup>19</sup> Oscar Grånäs,<sup>13,21</sup> E. K. U. Gross,<sup>12</sup> Andris Gulans,<sup>14,15</sup> François Gygi,<sup>22</sup> D. R. Hamann,<sup>23,24</sup> Phil J. Hasnip,<sup>25</sup> N. A. W. Holzwarth,<sup>26</sup> Diana Iuşan,<sup>13</sup> Dominik B. Jochym,<sup>27</sup> François Jollet,<sup>28</sup> Daniel Jones,<sup>29</sup> Georg Kresse,<sup>30</sup> Klaus Koepnik,<sup>31,32</sup> Emine Küçükbenli,<sup>9,11</sup> Yaroslav O. Kvashnin,<sup>13</sup> Inka L. M. Loch,<sup>13,33</sup> Sven Lubeck,<sup>14</sup> Martijn Marsman,<sup>30</sup> Nicola Marzari,<sup>9</sup> Ulrike Nitzsche,<sup>31</sup> Lars Nordström,<sup>13</sup> Taisuke Ozaki,<sup>34</sup> Lorenzo Paulatto,<sup>35</sup> Chris J. Pickard,<sup>36</sup> Ward Poelmans,<sup>1,37</sup> Matt I. J. Probert,<sup>25</sup> Keith Refson,<sup>38,39</sup> Manuel Richter,<sup>31,32</sup> Gian-Marco Rignanese,<sup>19</sup> Santanu Saha,<sup>20</sup> Matthias Scheffler,<sup>15,40</sup> Martin Schlipf,<sup>22</sup> Karlheinz Schwarz,<sup>5</sup> Sangeeta Sharma,<sup>12</sup> Francesca Tavazza,<sup>17</sup> Patrik Thunström,<sup>41</sup> Alexandre Tkatchenko,<sup>15,42</sup> Marc Torrent,<sup>28</sup> David Vanderbilt,<sup>23</sup> Michiel J. van Setten,<sup>19</sup> Veronique Van Speybroeck,<sup>1</sup> John M. Wills,<sup>43</sup> Jonathan R. Yates,<sup>29</sup> Guo-Xu Zhang,<sup>44</sup> Stefaan Cottenier<sup>1,45\*</sup>

The widespread popularity of density functional theory has given rise to an extensive range of dedicated codes for predicting molecular and crystalline properties. However, each code implements the formalism in a different way, raising questions about the reproducibility of such predictions. We report the results of a community-wide effort that compared **15 solid-state codes**, using 40 different potentials or basis set types, to assess the quality of the Perdew-Burke-Ernzerhof equations of state for 71 elemental crystals. We conclude that predictions from recent codes and pseudopotentials agree very well, with pairwise

Science, Volume 351, Issue 6280, Mar 2016

Code	Basis	Electron treatment
Wien2k	LAPW+lo	Full-potential
FLEUR	LAPW+lo	Full-potential
Exciting	LAPW+lo	Full-potential
Elk	LAPW+lo	Full-potential
FHI-aims	Numeric atom-centered orbitals	Full-potential
FPLO	Local orbitals	Full-potential
RSPT	Linear Muffin-Tin Orbitals	Full-potential
Abinit	Plane-waves	Pseudopotential
Quantum ESPRESSO	Plane-waves	Pseudopotential
VASP	Plane-waves	Pseudopotential
GPAW	Plane-waves	Pseudopotential
CASTEP	Plane-waves	Pseudopotential
DACAPO	Plane-waves	Pseudopotential
BigDFT	Daubechies wavelet	Pseudopotential
OpenMX	Pseudo-atomic localized basis functions	Pseudopotential



# Pseudopotential plane-wave method

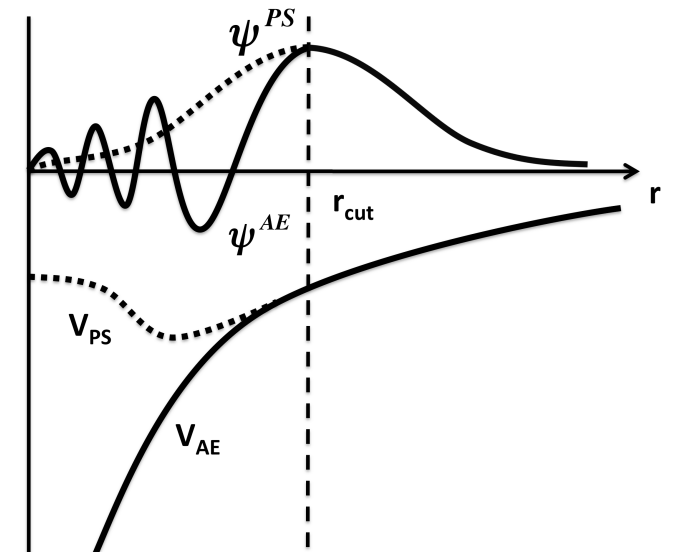
- Unit cell is mapped to a regular grid
- All functions are expanded in plane-waves
- Atomic potential is replaced by a pseudopotential  $\hat{V}_{PS} = V_{loc}(\mathbf{r}) + \sum_{\alpha} \sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$

Basis functions:

$$\varphi_{\mathbf{G}+\mathbf{k}}(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} e^{i(\mathbf{G}+\mathbf{k})\mathbf{r}}$$

Potential and density:

$$V(\mathbf{r}) = \sum_{\mathbf{G}} V(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} \quad \rho(\mathbf{r}) = \sum_{\mathbf{G}} \rho(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}}$$



# Pseudopotential plane-wave method

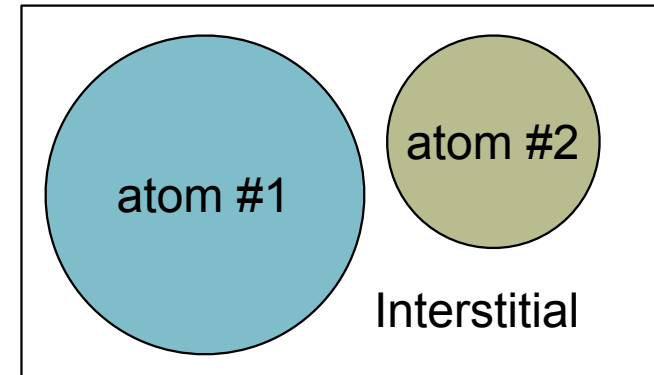
- Approximation to atomic potential
- Core states are excluded
- Number of basis functions:  $\sim 1000$  / atom
- Number of valence states:  $\sim 0.001 - 0.01\%$  of the total basis size
- Efficient iterative subspace diagonalization schemes exist
- Atomic forces can be easily computed
- Stress tensor can be easily computed

# Full-potential linearized augmented plane-wave method

- Unit cell is partitioned into “muffin-tin” spheres and interstitial region
- Inside MT spheres spherical harmonic expansion is used
- In the interstitial region functions are expanded in plane-waves

Basis functions:

$$\varphi_{\mathbf{G}+\mathbf{k}}(\mathbf{r}) = \begin{cases} \sum_{\ell m} \sum_{\nu=1}^{O_{\ell}^{\alpha}} A_{\ell m \nu}^{\alpha}(\mathbf{G} + \mathbf{k}) u_{\ell \nu}^{\alpha}(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \text{MT}\alpha \\ \frac{1}{\sqrt{\Omega}} e^{i(\mathbf{G}+\mathbf{k})\mathbf{r}} & \mathbf{r} \in \text{I} \end{cases}$$



Potential and density:

$$V(\mathbf{r}) = \begin{cases} \sum_{\ell m} V_{\ell m}^{\alpha}(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \text{MT}\alpha \\ \sum_{\mathbf{G}} V(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} & \mathbf{r} \in \text{I} \end{cases} \quad \rho(\mathbf{r}) = \begin{cases} \sum_{\ell m} \rho_{\ell m}^{\alpha}(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \text{MT}\alpha \\ \sum_{\mathbf{G}} \rho(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} & \mathbf{r} \in \text{I} \end{cases}$$

# Full-potential linearized augmented plane-wave method

- No approximation to atomic potential
- Core states are included
- Number of basis functions:  $\sim 100$  / atom
- Number of valence states:  $\sim 15-20\%$  of the total basis size
- Large condition number of the overlap matrix
- Full diagonalization of dense matrix is required (iterative subspace diagonalization schemes are not efficient)
- Atomic forces can be easily computed
- Stress tensor can't be easily computed (N-point numerical scheme is required)



## Common features of the FP-LAPW and PP-PW methods

- Definition of the unit cell (atoms, atom types, lattice vectors, symmetry operations, etc.)
- Definition of the reciprocal lattice, plane-wave cutoffs,  $\mathbf{G}$  vectors,  $\mathbf{G}+\mathbf{k}$  vectors
- Definition of the wave-functions as a 2D array of basis expansion coefficients
- FFT driver
- Generation of the charge density on the regular grid
- Generation of the XC-potential
- Symmetrization of the density, potential and occupancy matrices
- Low-level numerics (spherical harmonics, Bessel functions, Gaunt coefficients, spline interpolation, Wigner D-matrix, linear algebra wrappers, etc.)

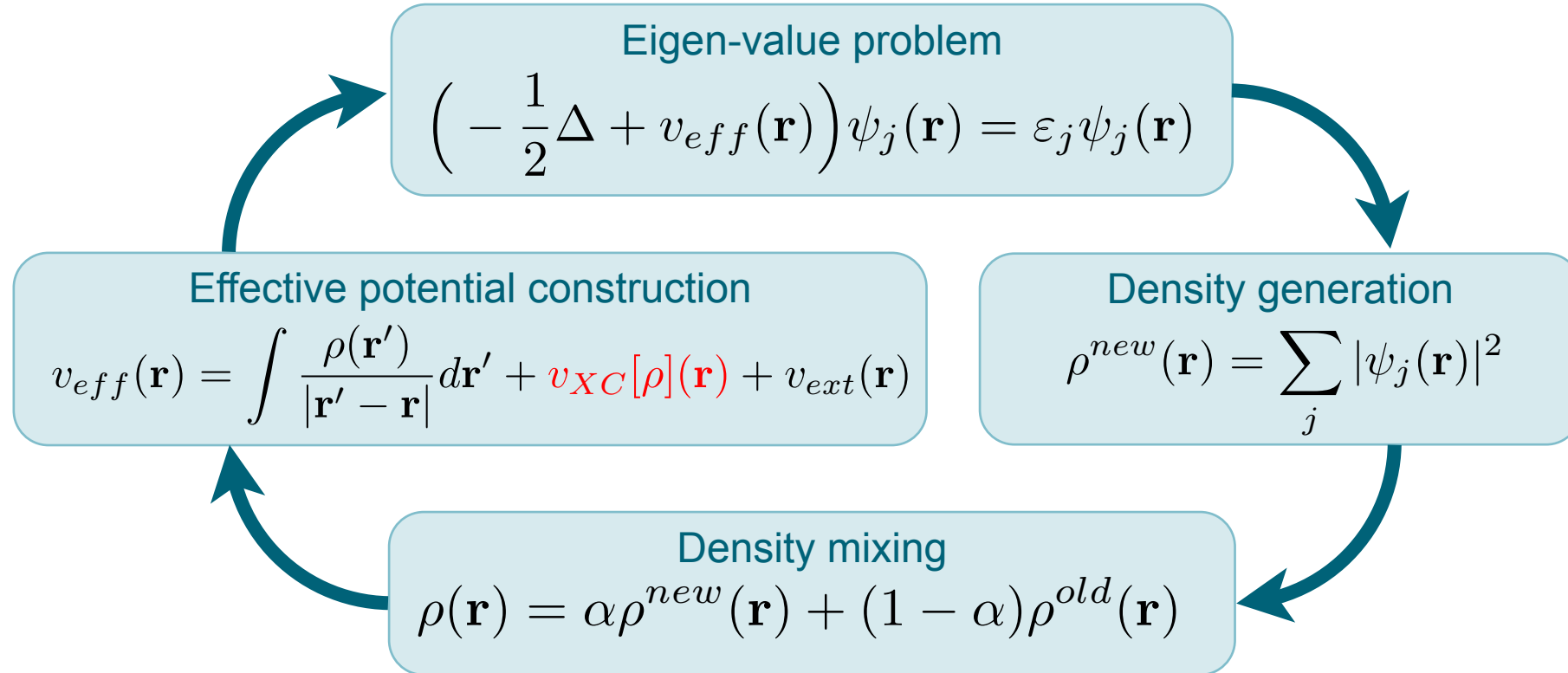
## Motivation for a common plane-wave DFT library

- 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 repetition of work!

It is beneficial to develop a common DFT functionality and establish interfaces with various electronic-structure codes.

# Where to draw the line?



## Output:

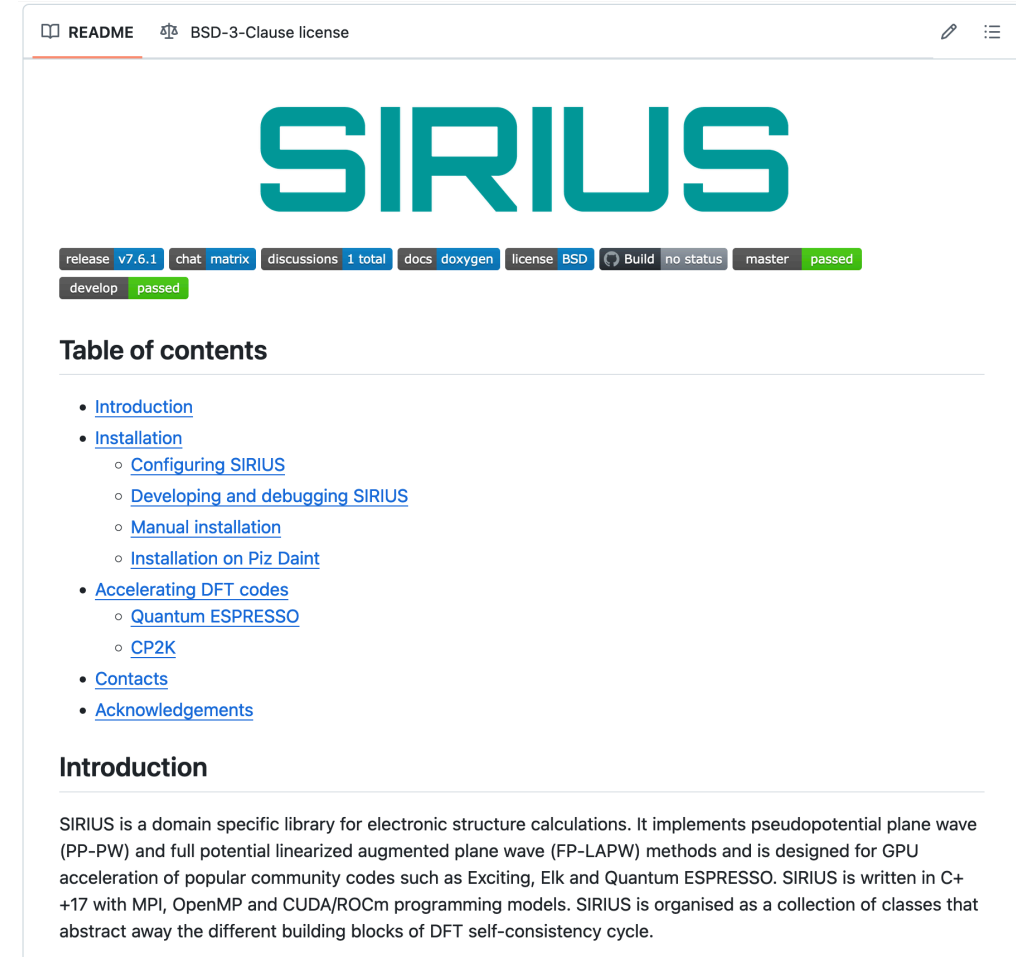
wave-functions  $\psi_j(\mathbf{r})$  and eigen energies  $\varepsilon_j$

charge density  $\rho(\mathbf{r})$  and magnetization  $\mathbf{m}(\mathbf{r})$

total energy  $E_{tot}$ , atomic forces  $\mathbf{F}_\alpha$  and stress tensor  $\sigma_{\alpha\beta}$

# SIRIUS library

programming model	<ul style="list-style-type: none"><li>• C++17 with OpenMP</li><li>• MPI</li><li>• CUDA/ROCM</li></ul>
build system	<ul style="list-style-type: none"><li>• CMake</li><li>• Spack</li></ul>
methods	<ul style="list-style-type: none"><li>• FP-LAPW (APW, LAPW + any combination of local orbitals)<ul style="list-style-type: none"><li>• ZORA, IORA</li></ul></li><li>• PP-PW (NC, USPP, PAW)<ul style="list-style-type: none"><li>• DFT+U, DFT+U+V</li></ul></li></ul>
core functionality	<ul style="list-style-type: none"><li>• DFT ground state, energy, forces, stress tensor</li><li>• spin-orbit</li><li>• non-collinear magnetism</li><li>• local lattice relaxation (VC-SQNM module)</li></ul>
bindings	<ul style="list-style-type: none"><li>• Fortran 90 (ISO_C_BINDING)</li><li>• Pybind11 (at different granularity)</li><li>• Julia</li></ul>
data formats	<ul style="list-style-type: none"><li>• JSON</li><li>• XML</li><li>• HDF5</li></ul>



The screenshot shows the GitHub README for the SIRIUS library. At the top, it displays the project name 'SIRIUS' in a large, teal, stylized font. Below the name, there are several status bars: 'release v7.6.1', 'chat matrix', 'discussions 1 total', 'docs doxygen', 'license BSD', 'Build no status', and 'master passed'. A 'develop passed' bar is also visible. The 'Table of contents' section lists the following links: Introduction, Installation (with sub-links for Configuring SIRIUS, Developing and debugging SIRIUS, Manual installation, and Installation on Piz Daint), Accelerating DFT codes (with sub-links for Quantum ESPRESSO and CP2K), Contacts, and Acknowledgements. The 'Introduction' section begins with the text: '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++17 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.'

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

# Fortran API example

```
! initialize the library
call sirius_initialize(call_mpi_init=.true.)

! create simulation context using a specified communicator
call sirius_create_context(MPI_COMM_WORLD, handler)

call sirius_import_parameters(handler, &
    '{"parameters" : {"electronic_structure_method" : "pseudopotential"},&
    "control" : {"verbosity" : 1, "verification" : 0}}')

! atomic units are used everywhere
! plane-wave cutoffs are provided in a.u.^-1
call sirius_set_parameters(handler, pw_cutoff=40.d0, gk_cutoff=7.d0)

lat_vec = 0.d0
do i = 1, 3
    lat_vec(i,i) = 7.260327248
enddo
! disturb the lattice a little bit
lat_vec(1,3) = 0.001

call sirius_set_lattice_vectors(handler, lat_vec(:, 1), lat_vec(:, 2), lat_vec(:, 3))

call sirius_add_atom_type(handler, "Sr", fname="Sr.UPC")
call sirius_add_atom_type(handler, "V", fname="V.UPF")
call sirius_add_atom_type(handler, "O", fname="O.UPF")
```

# Interoperability with the host code (high level overview)

Host code interacts with SIRIUS via API  
(C and Fortran90 bindings are provided)

Example:

```
! create context of simulation
CALL sirius_create_context(intra_image_comm, sctx,&
    &fcomm_k=inter_pool_comm,&
    &fcomm_band=intra_pool_comm, error_code=ierr)
IF (ierr .NE. 0) THEN
    STOP 'error in sirius_create_context()'
END IF
```

Once the simulation parameters are set up, host code calls SIRIUS to find the ground state and get back total energy, lattice stress and atomic forces.

Host code performs the lattice relaxation step and finds new lattice parameters and atomic positions.

## SIRIUS setup phase

Create, set and initialize `Simulation_context` instance

- set lattice vectors, atom types and atomic positions
- set pseudopotential or LAPW basis description
- set plane-wave cutoffs and other simulation parameters
- set XC potential type

Create and initialize `K_point_set` instance

Create and initialize `DFT_ground_state` instance

## SIRIUS execution phase

Run `DFT_ground_state` and compute total energy, stress and forces components

## SIRIUS update phase

Update lattice vectors and atomic positions and recompute dependent variables

# GPU backend

- wrappers for cublas and rocblas linear algebra functions
- GPU kernels are written in CUDA/ROCm and compiled separately by nvcc/hip
- main C++ code calls GPU kernels using `extern "C"` interface
- math primitive functions and types are substituted at compile time

```
#if defined(SIRIUS_CUDA)
using acc_complex_double_t = cuDoubleComplex;
#define make_accDoubleComplex make_cuDoubleComplex
#define accCadd cuCadd
#define accCmul cuCmul
...
#define ACC_DYNAMIC_SHARED(type, var) extern __shared__ type var[];

#elif defined(SIRIUS_ROCM)
using acc_complex_double_t = hipDoubleComplex;
#define make_accDoubleComplex make_hipDoubleComplex
#define accCadd hipCadd
#define accCmul hipCmul
...
#define ACC_DYNAMIC_SHARED(type, var) HIP_DYNAMIC_SHARED(type, var)
#endif
```



# CUDA / ROCm kernels

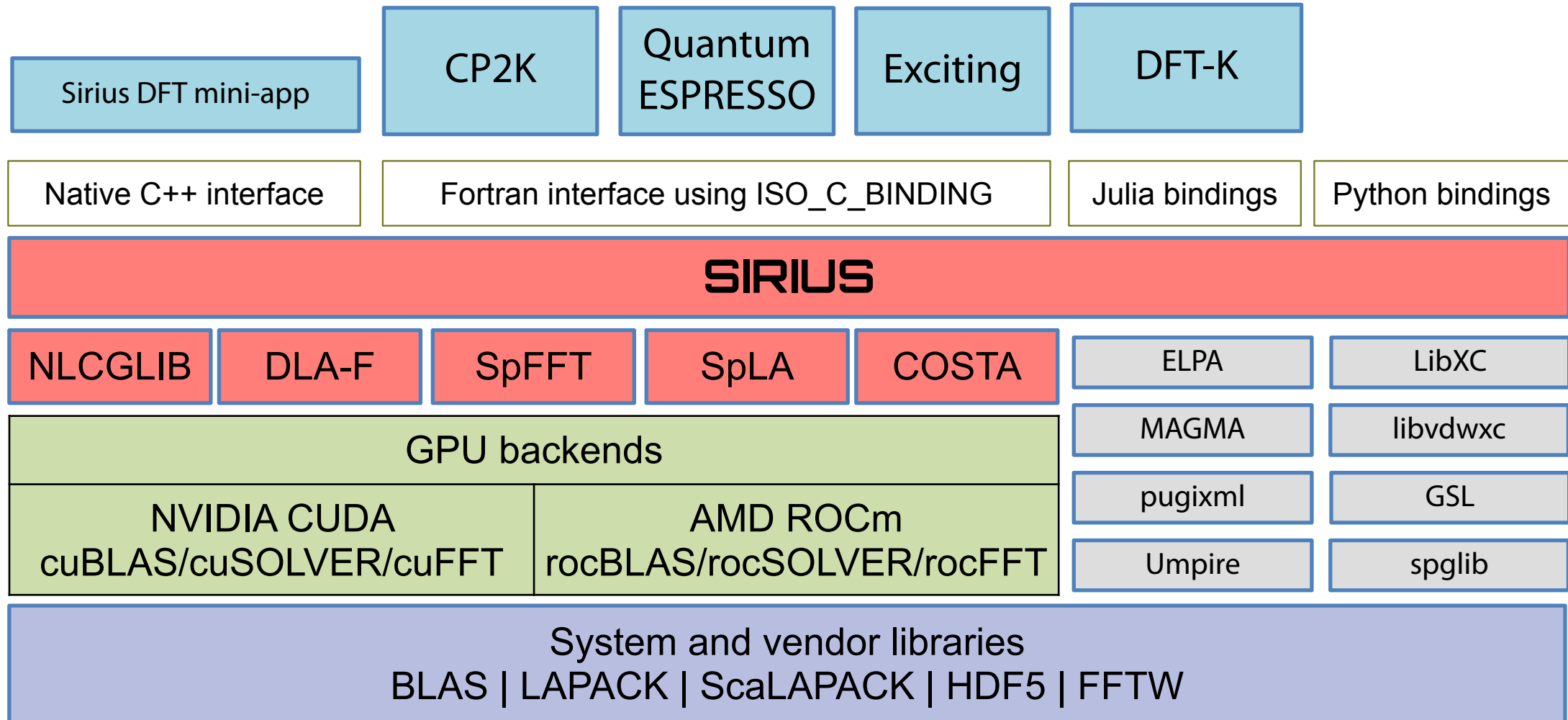
```
/* CUDA runtime calls and definitions */
#ifdef __CUDA
#define accLaunchKernel(kernelName, numblocks, numthreads, memperblock, streamId, ...) \
    do { \
        kernelName<<<numblocks, numthreads, memperblock, streamId>>>(__VA_ARGS__); \
    } while (0)
#endif
/* ROCM runtime calls and definitions */
#ifdef __ROCM
#define accLaunchKernel(...) \
    do { \
        hipLaunchKernelGGL(__VA_ARGS__); \
    } while (0)
#endif

__global__ void add_pw_ekin_gpu_kernel(int num_gvec__, double alpha__, double const* pw_ekin__, acc_complex_double_t const* phi__,
                                     acc_complex_double_t const* vphi__, acc_complex_double_t* hphi__)
{
    int ig = blockIdx.x * blockDim.x + threadIdx.x;
    if (ig < num_gvec__) {
        acc_complex_double_t z1 = accCadd(vphi__[ig], make_accDoubleComplex(alpha__ * pw_ekin__[ig] * phi__[ig].x,
                                                                            alpha__ * pw_ekin__[ig] * phi__[ig].y));
        hphi__[ig] = accCadd(hphi__[ig], z1);
    }
}

/// Update the hphi wave functions.
extern "C" void add_pw_ekin_gpu(int num_gvec__, double alpha__, double const* pw_ekin__, acc_complex_double_t const* phi__,
                               acc_complex_double_t const* vphi__, acc_complex_double_t* hphi__)
{
    dim3 grid_t(64);
    dim3 grid_b(num_blocks(num_gvec__, grid_t.x));

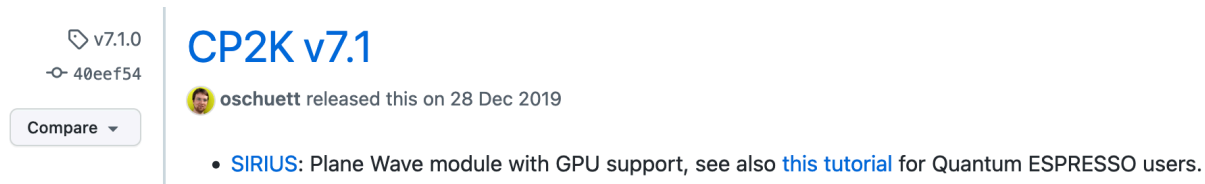
    accLaunchKernel((add_pw_ekin_gpu_kernel), dim3(grid_b), dim3(grid_t), 0, 0, num_gvec__, alpha__, pw_ekin__, phi__, vphi__, hphi__);
}
```

# Software stack



# CP2K

- Starting from v7.1 CP2K supports interface to SIRIUS. Details are available in “CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations” // *J. Chem. Phys.* 152, 194103 (2020)



A screenshot of a software release page for CP2K v7.1. The page shows the version number 'v7.1.0' and a commit hash '40eef54'. It identifies the user 'oschuett' as the releaser on '28 Dec 2019'. A 'Compare' button is visible. A bullet point highlights the 'SIRIUS: Plane Wave module with GPU support, see also [this tutorial](#) for Quantum ESPRESSO users.'

## B. SIRIUS: Plane wave density functional theory support

CP2K supports computations of the electronic ground state, including forces and stresses,<sup>419,420</sup> in a PW basis. The implementation relies on the quantum engine SIRIUS.<sup>421</sup>

- Possible use cases:
  - Extend CP2K with plane-wave DFT capabilities; use CP2K native or UPF pseudopotentials
  - Run FP-LPAW calculations using CP2K input file without a need to switch to a different code and a different input file. For example, get a reference full-potential total energy in “Delta-DFT” benchmarks within CP2K

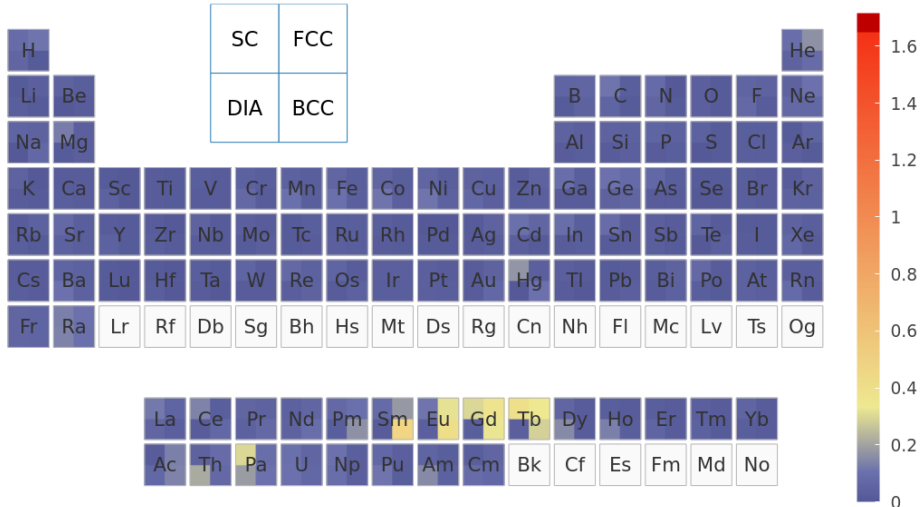
# Equation of states with CP2K/Sirius

In collaboration with Hossein Mirhosseini, PhD

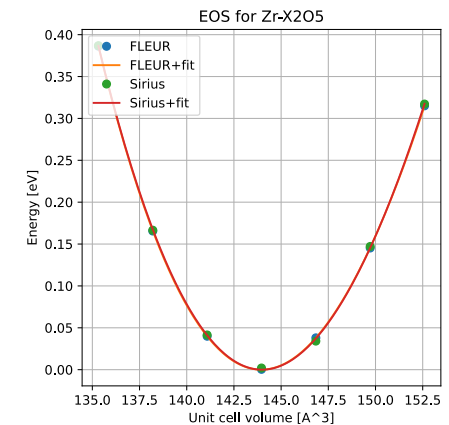
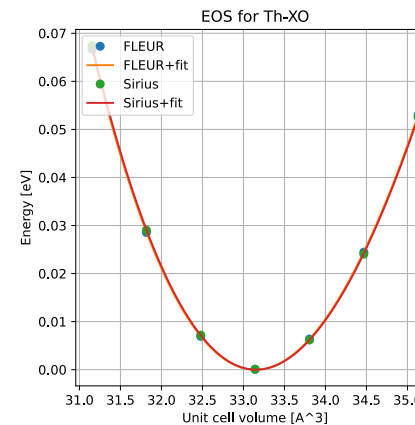
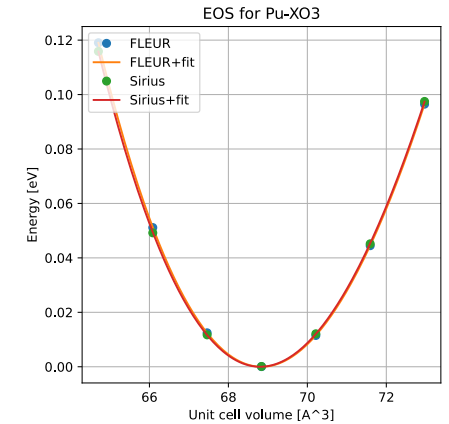
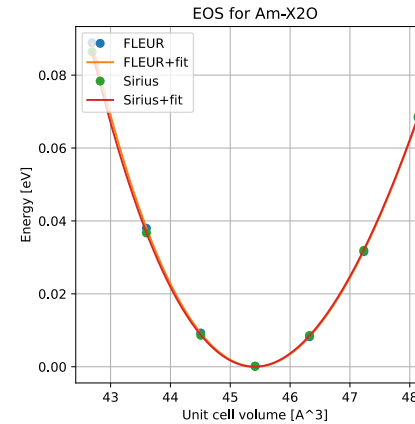
Center for Advanced Systems Understanding (CASUS)  
 Helmholtz-Zentrum Dresden-Rossendorf e.V. (HZDR)  
 Conrad-Schiedt-Straße 20, 02826 Görlitz  
<https://www.casus.science>

## Unaries

$\nu$  for SIRIUS@PF-LAPW vs. all-electron average



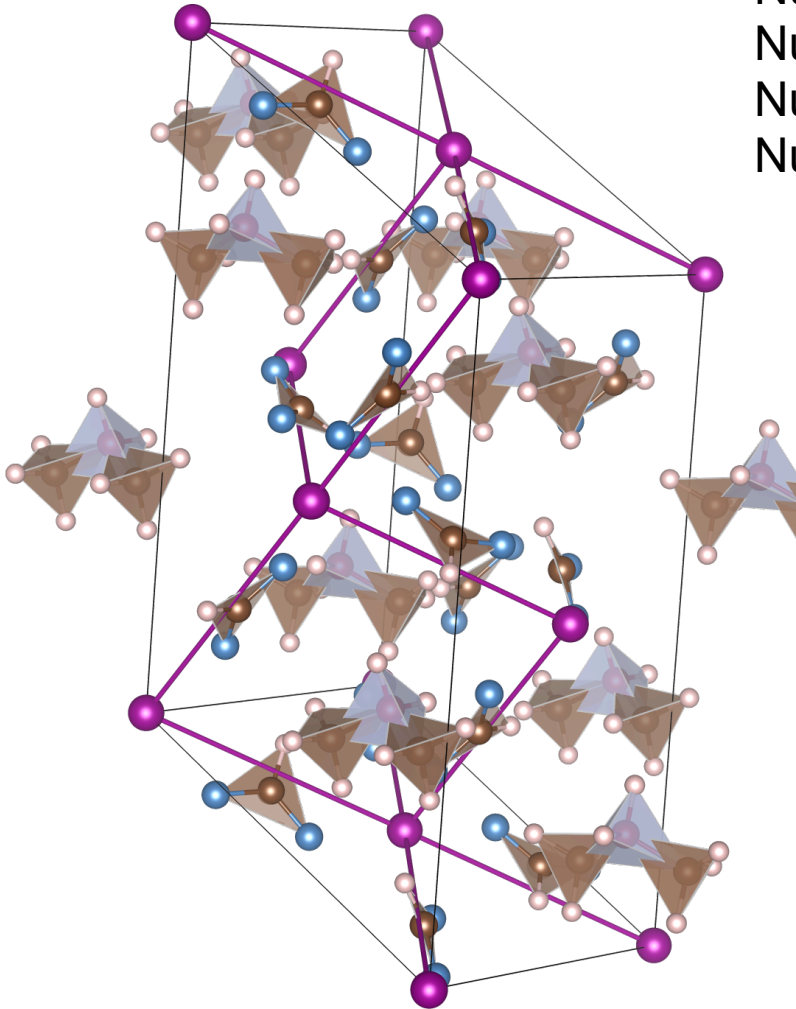
## Oxides



## Exciting code

- PoC interface with SIRIUS was implemented in Exciting long time ago
- **Finally!** SIRIUS bindings were integrated in the main Exciting branch
- **WIP:** benchmark and tweak
  
- Use cases:
  - Enable a GPU backed for Exciting (for both NVIDIA and AMD GPU cards)
  - Enable distributed Hamiltonian setup and diagonalization
  - Enable simulations of large unit cells (>200 atoms)

# Exciting benchmark of Mn-MOF



Number of atoms: **96**  
Number of k-points: **24**  
Number of basis functions: **~16300**  
Number of bands: **~180**

	Exciting	Exciting/SIRIUS	
	Intel Broadwell nodes	NVIDIA GH200 nodes	
Number of sockets / node	2	4	
Node performance (TFlop/s)	~1.2	~200	<b>1:166</b>
Total number of nodes	12	1	
One SCF iteration time (sec.)	1000.4	91.065	
Computational cost of a single SCF iteration (node-hours)	3.33	0.025	<b>1:133</b>



# Embedded DFT

- If your project requires a “black box” DFT solver, SIRIUS can help!

## Missing theoretical evidence for conventional room temperature superconductivity in low enthalpy structures of carbonaceous sulfur hydrides

Moritz Gubler,<sup>1</sup> José A. Flores-Livas,<sup>2,3</sup> Anton Kozhevnikov,<sup>4</sup> and Stefan Goedecker<sup>1</sup>

<sup>1</sup>*Department of Physics, University of Basel, Klingelbergstrasse 82, CH-4056 Basel, Switzerland*

<sup>2</sup>*Dipartimento di Fisica, Università di Roma La Sapienza, Piazzale Aldo Moro 5, I-00185 Roma, Italy*

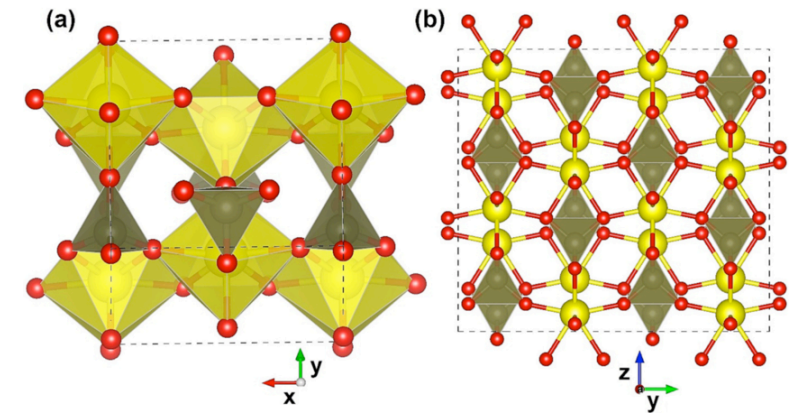
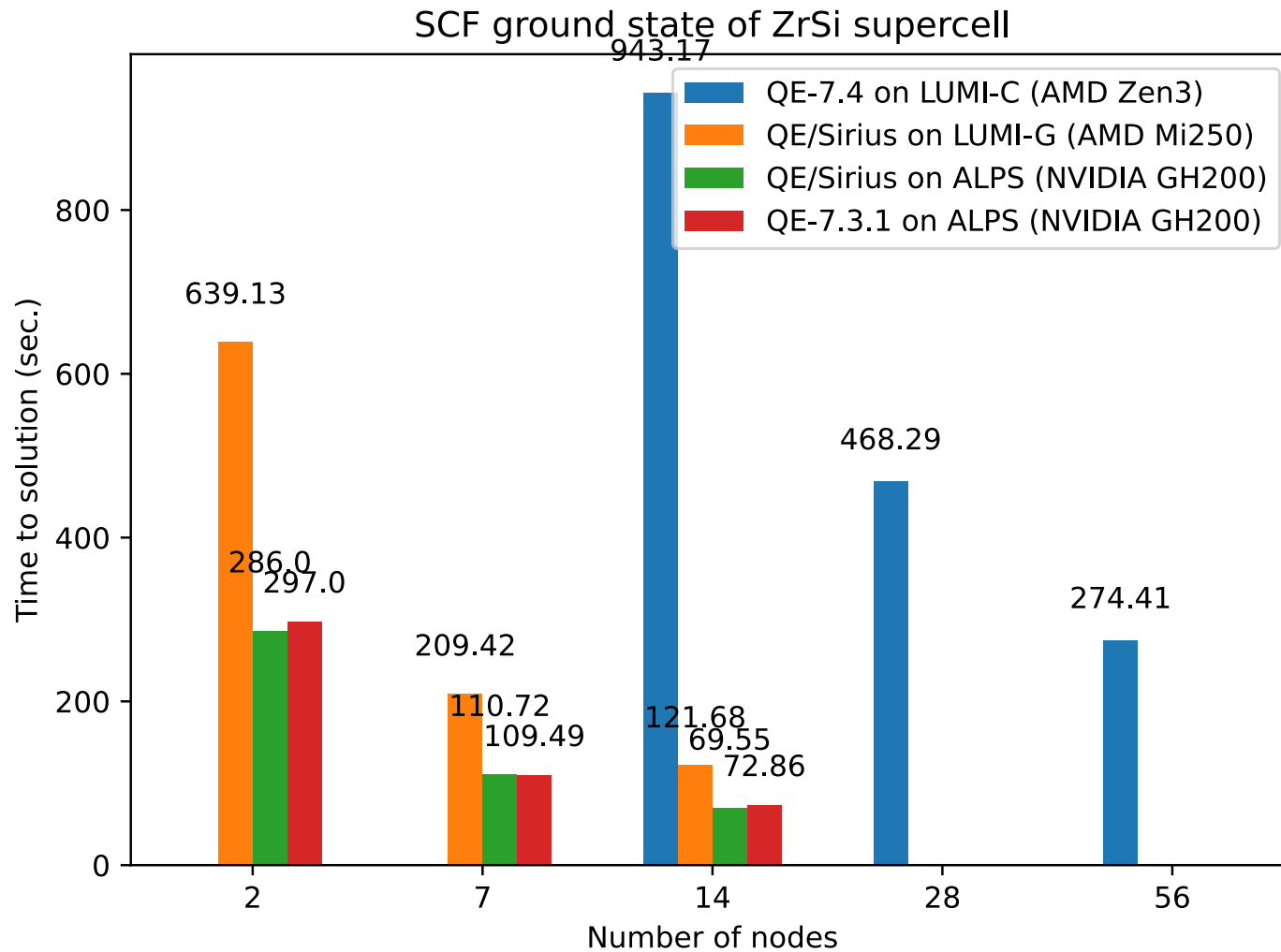
<sup>3</sup>*RIKEN Center for Emergent Matter Science, 2-1 Hirosawa, Wako, 351-0198, Japan*

<sup>4</sup>*CSCS, Swiss National Supercomputing Centre, Via Trevano 131, CH-6900 Lugano, Switzerland*

To elucidate the geometric structure of the putative room temperature superconductor, carbonaceous sulfur hydride, at high pressure, we present the results of an extensive computational structure search of bulk C-S-H at 250 gigapascals. Using the minima hopping structure prediction method coupled to the GPU accelerated Sirius library, more than 17,000 local minima with different stoichiometries in large simulation cells were investigated. Only 24 stoichiometries are favourable against elemental decomposition, all of them are carbon doped H<sub>3</sub>S crystals. The absence of van Hove singularities or similar peaks in the electronic density of states of more than 3.000 candidate phases rules out conventional superconductivity in C-S-H at room-temperature.

Submitted to “Physical Review Materials.”

# QE/Sirius



Number of atoms: **144**

Number of k-points: **112**

Number of basis functions: **~60000**

Number of bands: **~700**

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

We can run on LUMI-G and Frontier!

# Where Sirius is used

**Roadmap on electronic structure codes in the exascale era.**

DOI: 10.1088/1361-651X/acdf06

**All-electron APW+lo calculation of magnetic molecules with the SIRIUS domain-specific package**

DOI: 10.1063/5.0139497

**CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations**

DOI: 10.1063/5.0007045

**The ternary phase diagram of nitrogen doped lutetium hydrides can not explain its claimed high Tc superconductivity**

DOI: 10.1088/1367-2630/ad0e1a

**Efficient variable cell shape geometry optimization**

<https://doi.org/10.1016/j.jcpx.2023.100131>

**Expansion of the Materials Cloud 2D Database**

DOI: 10.1021/acsnano.2c11510

**How to verify the precision of density-functional-theory implementations via reproducible and universal workflows**

<https://doi.org/10.1038/s42254-023-00655-3>

**Materials Cloud three-dimensional crystals database (MC3D)**

DOI: 10.24435/materialscloud:rw-t0

**Trends in Atomistic Simulation Software Usage**

DOI: 10.33011/livecoms.3.1.1483



**CSCS**

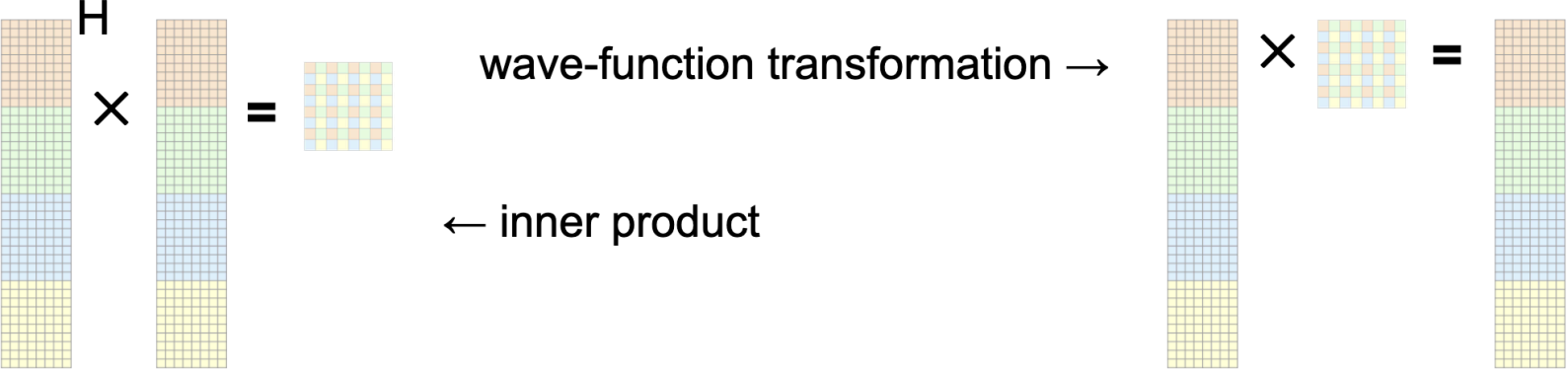
Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# A quick introduction to the most important backend libraries of SIRIUS

---

# SpLA

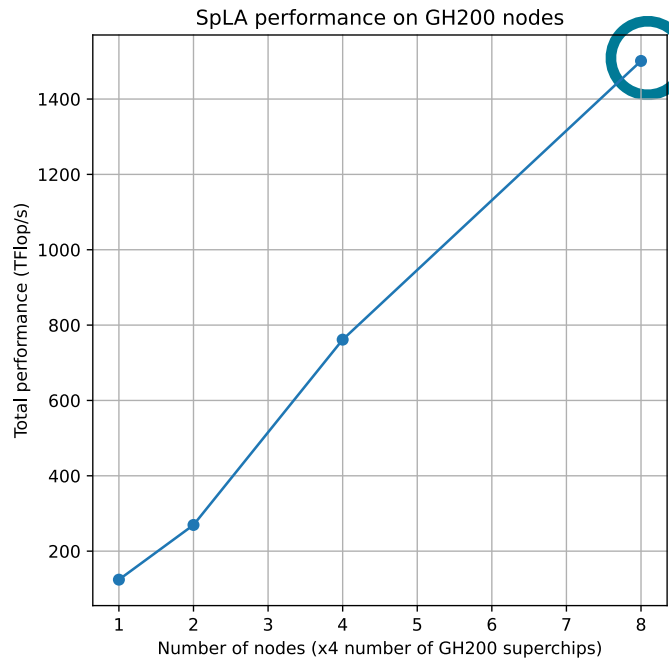
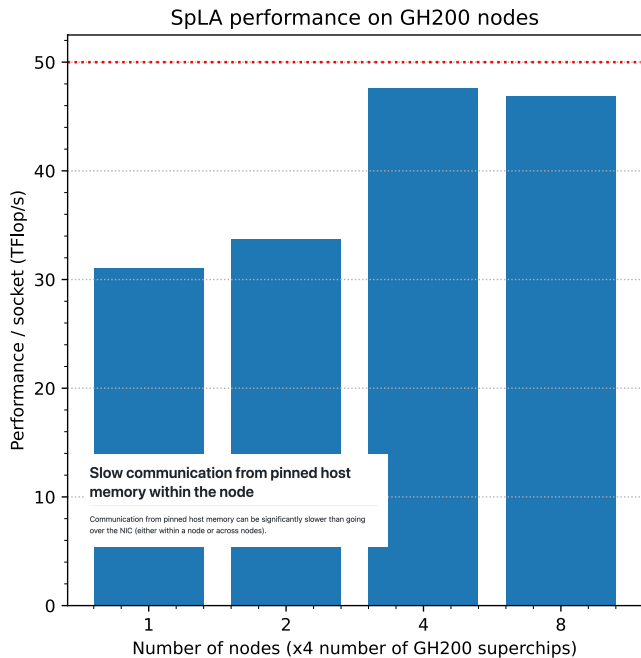
Short description	<p>Specialized parallel linear algebra operations. The library is designed to work with particular matrix distribution, corresponding to wave-function storage in plane-wave codes.</p>  <p>← inner product</p>
Key features	<ul style="list-style-type: none"><li>• MPI and OpenMP parallel</li><li>• Effective MPI ring communication avoiding <i>allreduce</i></li><li>• Accepts any combination of pointers (host or device)</li><li>• Backends for Nvidia and AMD GPUs via CUDA and ROCm</li></ul>
Use cases	<ul style="list-style-type: none"><li>• Iterative subspace methods</li><li>• Distributed tall-and-skinny matrix multiplication (for example, Green's function construction)</li></ul>
URL	<p><a href="https://github.com/eth-cscs/spla">https://github.com/eth-cscs/spla</a></p>

# SpLA benchmark

Compute inner product for 2 spins, 8000 bands and ~700K plane-waves

$$O_{ij} = \sum_{\sigma} \sum_{\mathbf{G}} \phi_i^{\sigma*}(\mathbf{G}) \phi_j^{\sigma}(\mathbf{G})$$

~1.5 PFlop/s



NEWS RELEASE 10-NOV-2008

## DOE's Oak Ridge supercomputer now world's fastest for open science

Business Announcement

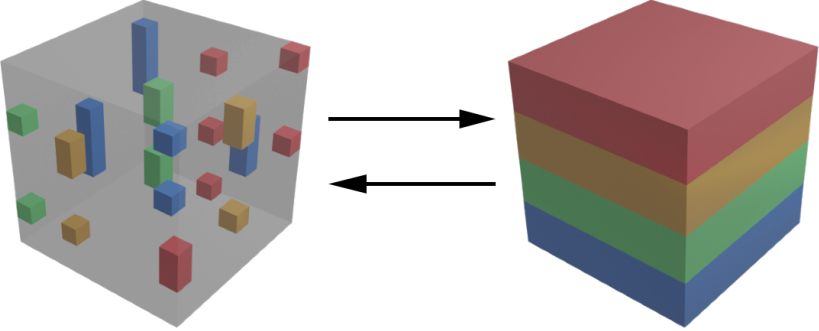
DOE/US DEPARTMENT OF ENERGY

OAK RIDGE, Tenn. -- The latest upgrade to the Cray XT Jaguar supercomputer at the Department of Energy's (DOE's) Oak Ridge National Laboratory (ORNL) has increased the system's computing power to a peak 1.64 "petaflops," or quadrillion mathematical calculations per second, making Jaguar the world's first petaflop system dedicated to open research. Scientists have already used the newly upgraded Jaguar to complete an unprecedented superconductivity calculation that achieved a sustained performance of more than 1.3 petaflops.

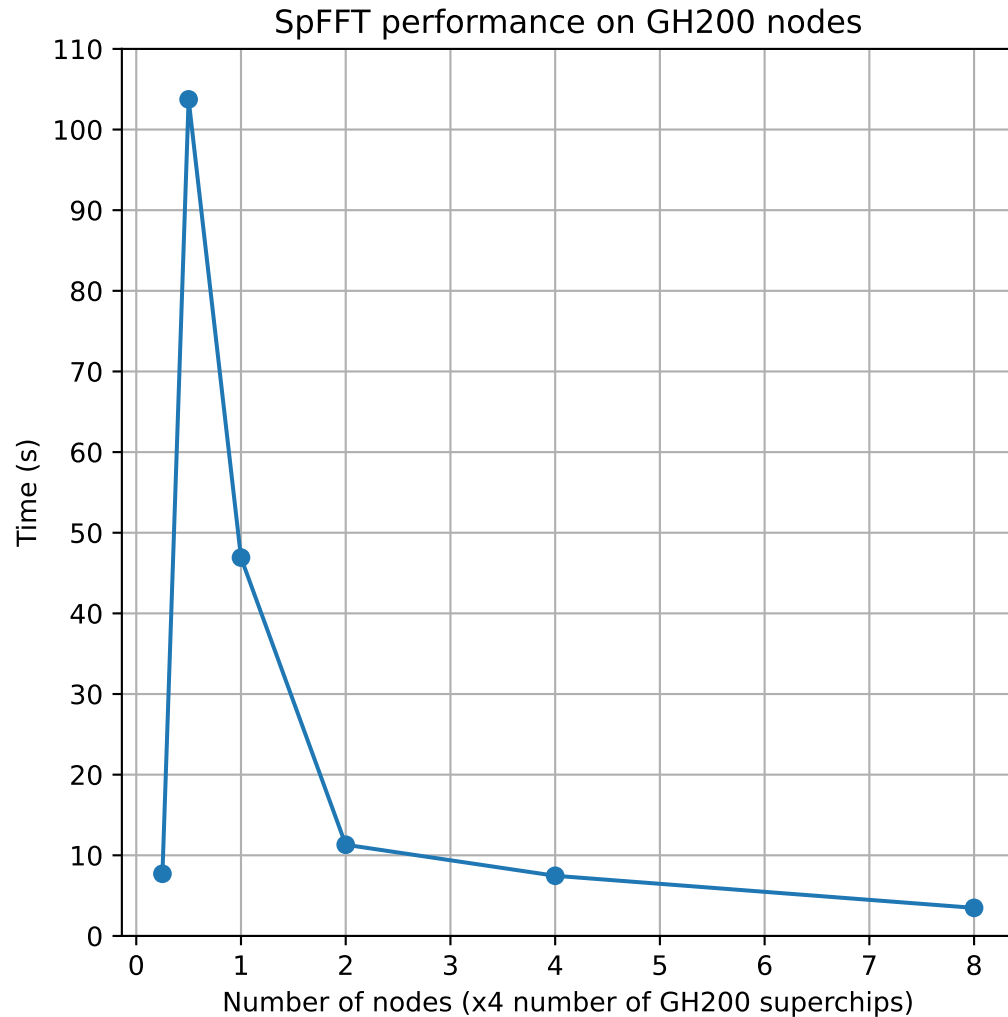




# SpFFT

Short description	SpFFT - A 3D FFT library for sparse frequency domain data 
Key features	<ul style="list-style-type: none"><li>• MPI and OpenMP parallel backends for Nvidia and AMD GPUs via CUDA and ROCm</li><li>• Slab decomposition in space domain and pencil decomposition in frequency domain (1D-2D FFT split, single MPI_Alltoall data exchange)</li><li>• Gamma-point support</li><li>• Unified interface for calculations on CPUs and GPUs</li></ul>
Use cases	Plane-wave codes where FFT is a bottleneck
URL	<a href="https://github.com/eth-cscs/SpFFT">https://github.com/eth-cscs/SpFFT</a>

# SpFFT benchmark



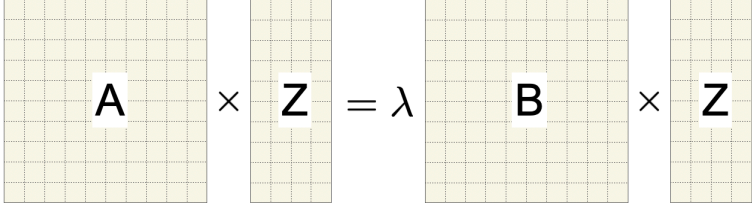
Test: local potential application to 1000 wave-functions

$$\Psi_i(\mathbf{G}) \xrightarrow{FFT^{-1}} \Psi_i(\mathbf{r}) \rightarrow \Psi_i(\mathbf{r}) \cdot V_{loc}(\mathbf{r}) \xrightarrow{FFT} [\Psi_i V](\mathbf{G})$$

FFT grid size : 450 x 450 x 450

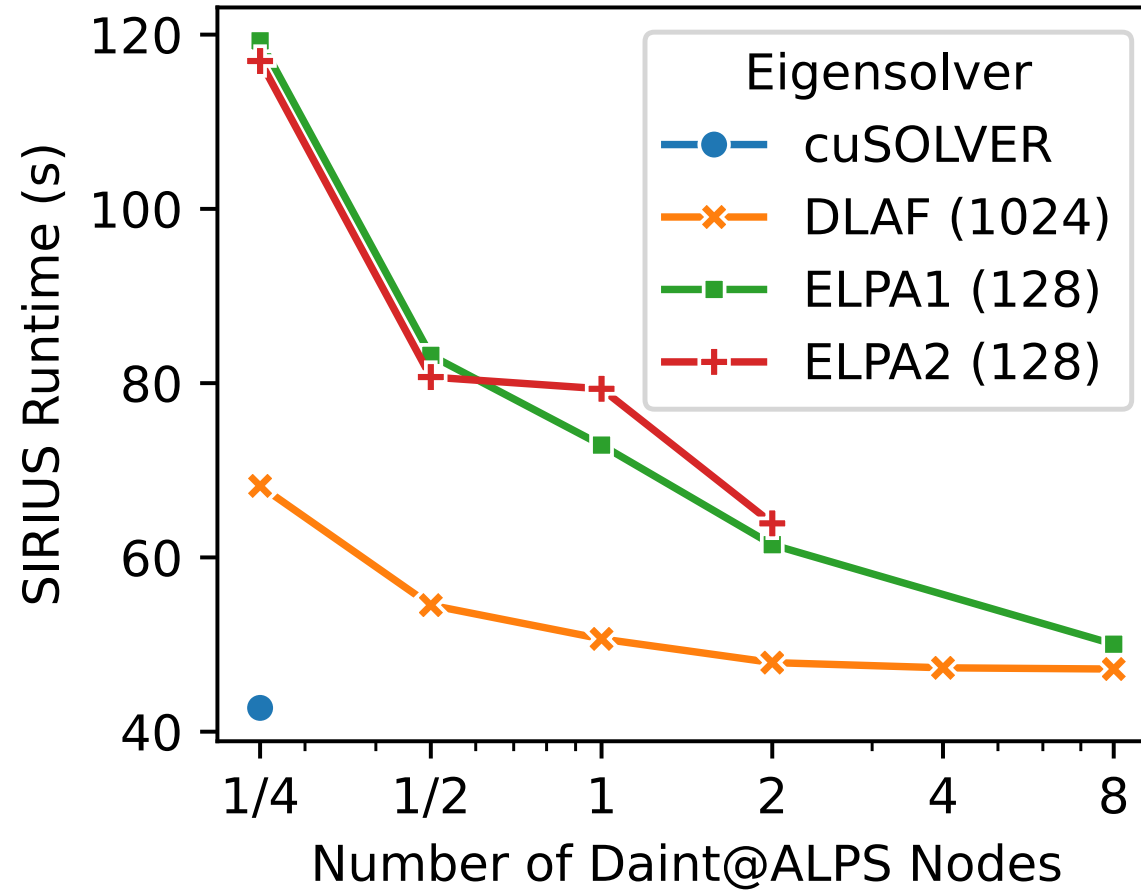
Number of G-vectors : 23'167'857

# DLA-Future

Short description	DLA-Future is a distributed linear algebra library implemented using C++ <code>std::execution</code> which provides generalized symmetric / hermitian eigenvalue solver  $A \times Z = \lambda B \times Z$
Key features	<ul style="list-style-type: none"><li>• Fully asynchronous task based approach, written in modern C++</li><li>• MPI parallel</li><li>• Modern C++ and C/Fortran ScaLAPACK-like interfaces</li><li>• ELSI interface</li><li>• Backends for Nvidia and AMD GPUs via CUDA and ROCm</li></ul>
Use cases	<ul style="list-style-type: none"><li>• Dense matrix diagonalization (arising, for example, in FP-LAPW method)</li></ul>
URL	<a href="https://github.com/eth-cscs/DLA-Future">https://github.com/eth-cscs/DLA-Future</a>

# DLA-F benchmark

## SIRIUS FP-LAPWlo for C60 (18'559)



# Summary

- CSCS develops and supports several libraries that can be of use to electronic structure community
- Sirius is an open-source plane-wave electronic structure library that is best suited for embedded DFT calculations
- It implements pseudo-potential and full-potential DFT ground state solvers and can run on NVIDIA and AMD GPU accelerators







**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

**Q & A**

---