



Towards Exascale LAPW
SIRIUS integration in exciting

Alexander Buccheri. MPSD, Hamburg.
5th August 2023. HU, Berlin

Overview

- **Part 1**

- Basics of Scaling with LAPW
- Introduction to **SIRIUS**
- Integration with **exciting**

- **Part 2**

- Benchmark 1
- Benchmark 2
- Current Bottlenecks in the implementation

Fundamentals of LAPW

- Basis: Linearised augmented plane waves
- Most precise numerical scheme/basis for solving KS equations in DFT
 - All electrons/no pseudo-potentials
 - No shape approximation to the potential
- Numerous flavours and approximations in basis and potential.
- Expensive

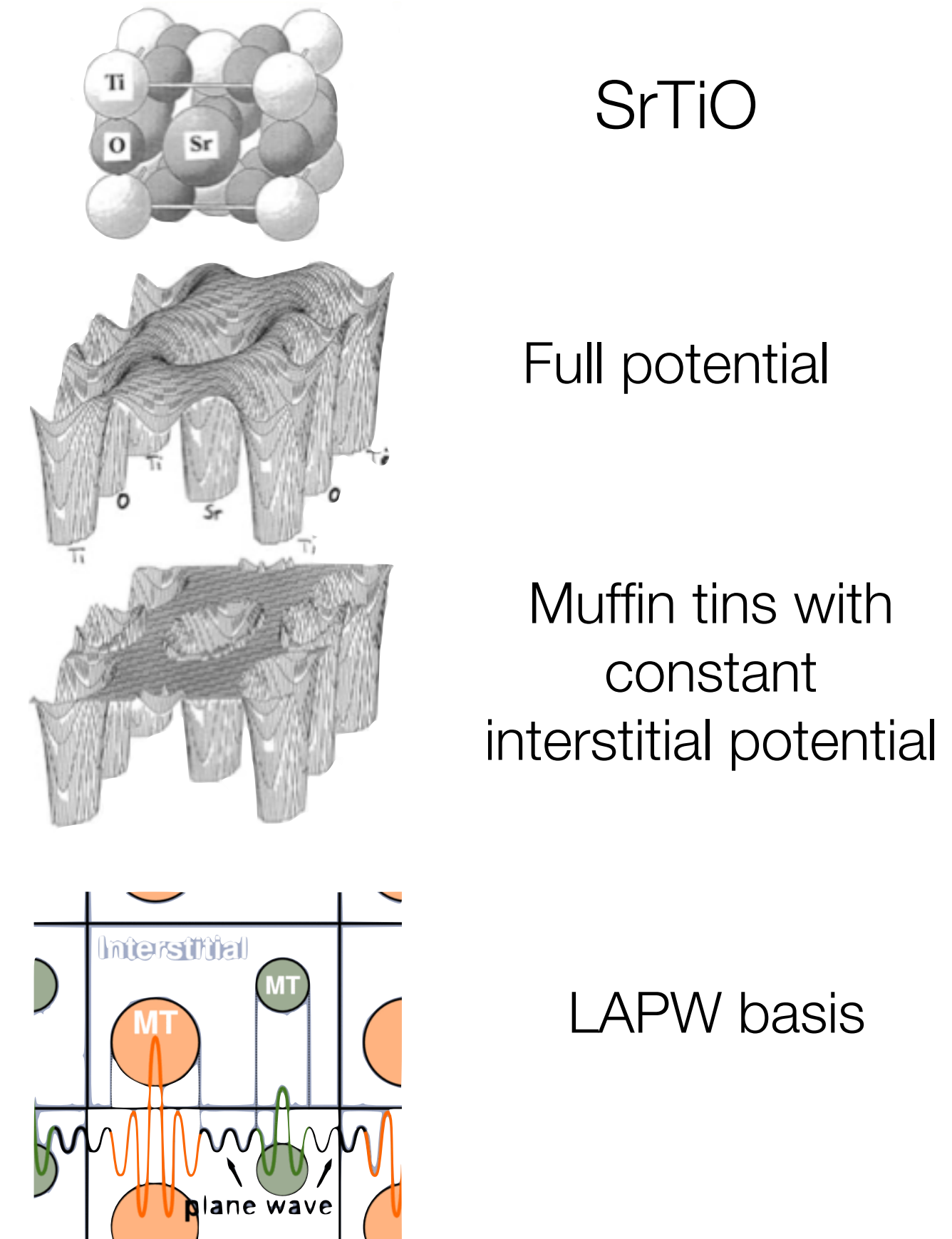


Fig 1. Potential and basis in an LAPW calculation. [1, 2]

[1]. Karlheinz Schwarz. <https://www.bc.edu/content/dam/bc1/schools/mcas/physics/pdf/wien2k/DFT%20and%20APW.pdf>

[2]. <https://elk.sourceforge.io/>

Most Expensive Part of a Ground State Calculation

- Hamiltonian setup (H, S)
 - Separate terms for interstitial and muffin-tin regions

$$H_{IR} + H_{MT} \quad S_{IR} + S_{MT}$$

- Solving the secular equation

$$\left[H(\mathbf{k}) - \epsilon_{\mathbf{k}\nu} S(\mathbf{k}) \right] c_{\mathbf{k}\nu} = 0 \quad \forall \mathbf{k} \in \text{BZ}$$

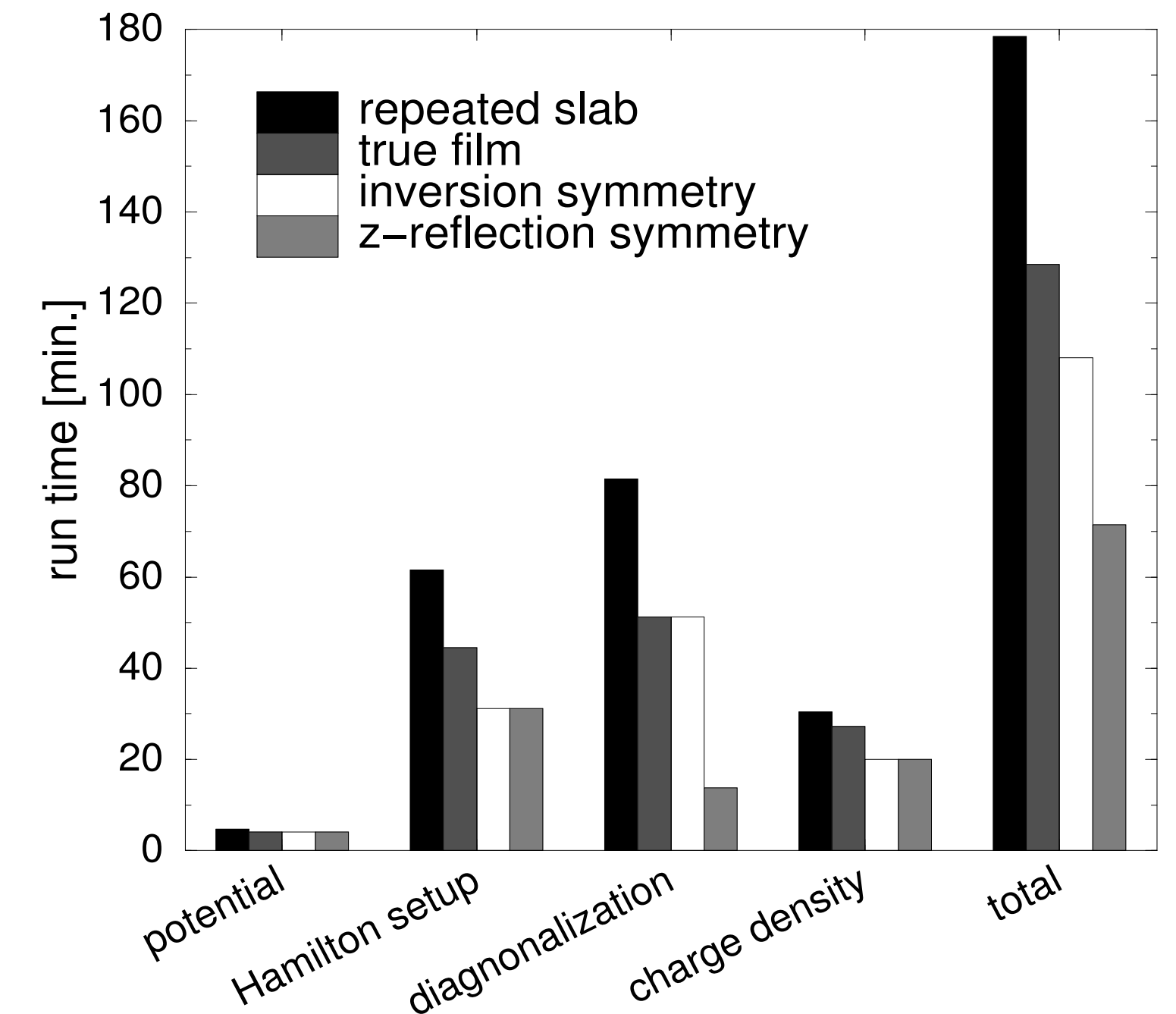


Fig1. Most time-consuming parts of an LAPW calculation on a p(4x2) Cu(110) slab, in 2006 [1].

Most Expensive Part of a Ground State Calculation

Muffin Tin

$$H_{MT}^{\mathbf{G}'\mathbf{G}}(\mathbf{k}) = \sum_{\mu} \sum_{L'L} \left(a_{L'}^{\mu\mathbf{G}'}(\mathbf{k}) \right)^* t_{L'L}^{\alpha\phi\phi} a_L^{\mu\mathbf{G}}(\mathbf{k}) + \left(b_{L'}^{\mu\mathbf{G}'}(\mathbf{k}) \right)^* t_{L'L}^{\alpha\phi\phi} b_L^{\mu\mathbf{G}}(\mathbf{k}) \\ + \left(a_{L'}^{\mu\mathbf{G}'}(\mathbf{k}) \right)^* t_{L'L}^{\alpha\phi\phi} b_L^{\mu\mathbf{G}}(\mathbf{k}) + \left(b_{L'}^{\mu\mathbf{G}'}(\mathbf{k}) \right)^* t_{L'L}^{\alpha\phi\phi} a_L^{\mu\mathbf{G}}(\mathbf{k})$$

Scaling

$$\text{prefactor} * N_{atoms} * N_{\mathbf{G}+\mathbf{k}}^2$$

$$\equiv \text{prefactor} * N_{atoms}^3$$

Terms

$t_{L'L}^{\alpha\phi\phi}$ Precomputed integrals

$a_L^{\mu\mathbf{G}}(\mathbf{k}), b_L^{\mu\mathbf{G}}(\mathbf{k})$ Matching coefficients

L . Index over radial basis functions

\mathbf{G}, \mathbf{G}' reciprocal lattice vectors

μ, α atomic, species indices

$N_{\mathbf{G}+\mathbf{k}}$ number of PWs with $\leq |\mathbf{G} + \mathbf{k}|$

Most Expensive Part of a Ground State Calculation

Interstitial

$$H_I^{\mathbf{G}\mathbf{G}'}(\mathbf{k}) = (V\Theta)_{(\mathbf{G}-\mathbf{G}')} + \frac{\hbar^2}{2m} (\mathbf{G}' + \mathbf{k})^2 \Theta_{(\mathbf{G}-\mathbf{G}')}$$

Terms

$\Theta_{(\mathbf{G}-\mathbf{G}')}$ Step function

$V_{(\mathbf{G})}$ Crystal potential Fourier coefficients

\mathbf{G}, \mathbf{G}' reciprocal lattice vectors

Scaling

$N_{\mathbf{G}+\mathbf{k}}$ number of PWs with $\leq |\mathbf{G} + \mathbf{k}|$

$N_{\mathbf{G}+\mathbf{k}} \ln N_{\mathbf{G}+\mathbf{k}}$ assuming use of FFT to compute $(V\Theta)_{(\mathbf{G}-\mathbf{G}')}$

SIRIUS Library

SIRIUS is a high-level domain-specific GPU accelerated PW and LAPW DFT library.

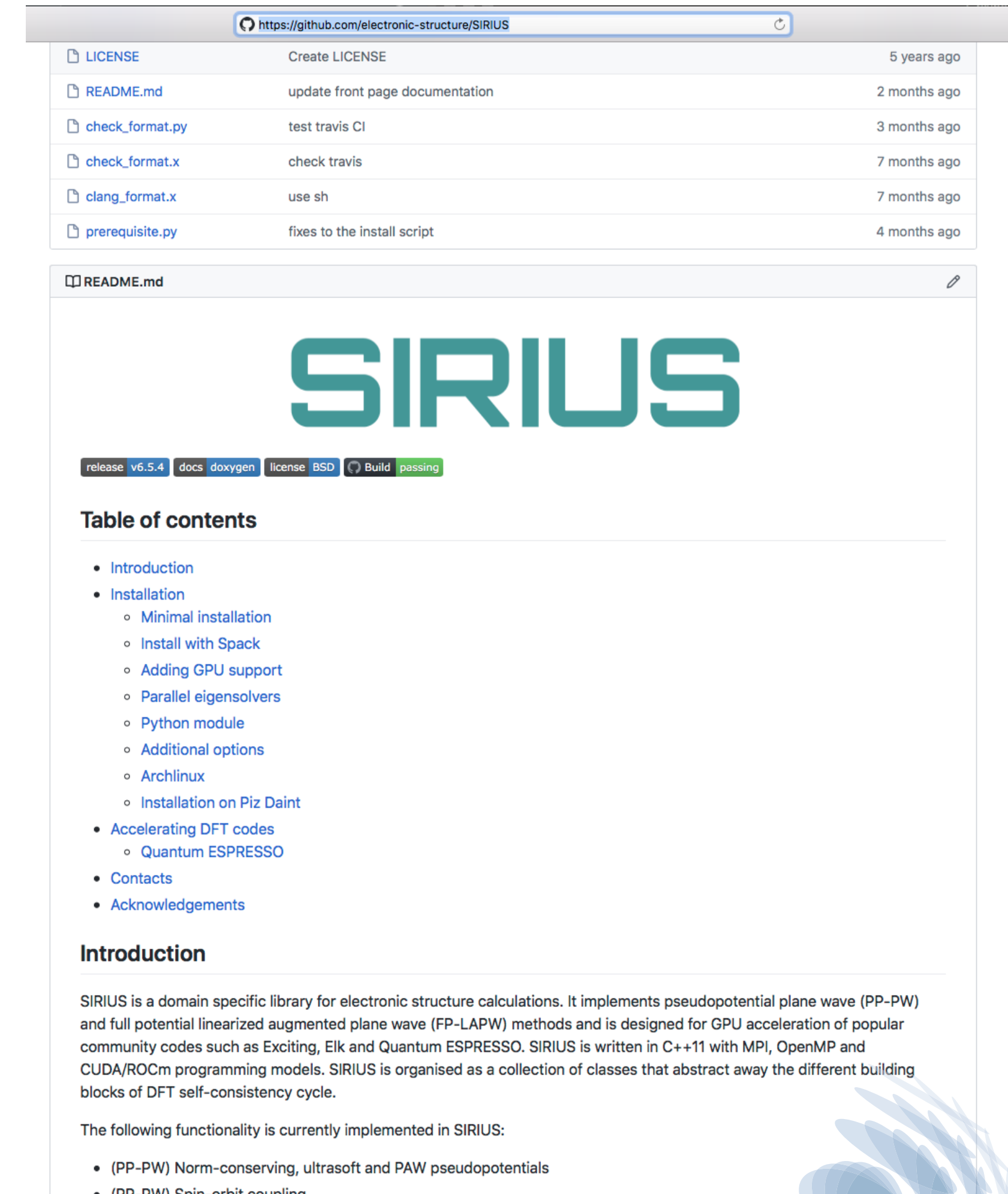
It is organised as a collection of C++ classes that abstract away the main components of PW and LAPW codes:

- Density and Potential
- Mixer
- **Kohn-Sham Eigen-solver**
- Symmetrization
- Stress and Forces

The library is written in C++14 with MPI, OpenMP and CUDA/ROCm programming models.

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

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



The screenshot shows the GitHub repository for SIRIUS. The top part of the page lists recent commits with their titles and dates. Below this is the README.md file content, which features the SIRIUS logo in a large, teal, stylized font. Underneath the logo are several status badges: 'release v6.5.4', 'docs doxygen', 'license BSD', and 'Build passing'. A 'Table of contents' section follows, listing various sections such as 'Introduction', 'Installation' (with sub-items like 'Minimal installation', 'Install with Spack', etc.), 'Accelerating DFT codes', 'Contacts', and 'Acknowledgements'. The 'Introduction' section is partially visible, describing SIRIUS as a domain-specific library for electronic structure calculations, implemented in C++11 with MPI, OpenMP, and CUDA/ROCm.

Sirius Dependencies

SIRIUS domain specific library
LAPW / PW implementation in C++

ELPA

spglib

libvdxwc

NLCGLIB

ScaLAPACK
PBLAS

GSL

LibXC

SpFFT

SPLA

LAPACK
BLAS

HDF5

CUDA
cuBlas
cuSolver
cuFFT
ELPA

ROCm
rocBLAS
rocFFT

MAGMA

FFTW

Use  Spack to manage the dependencies

```
spack install sirius@develop %gcc@9.3.0  
build_type=Release +scalapack +fortran +tests ^spla ^intel-  
mkl threads=openmp ^mpich@3.3.2 ^spfft+single_precision
```

```
==> Installing sirius-develop-  
vhzi64vzqgvkcoinf6sfilsirk34avd  
==> No binary for sirius-develop-  
vhzi64vzqgvkcoinf6sfilsirk34avd found: installing from  
source  
==> No patches needed for sirius  
==> sirius: Executing phase: 'cmake'  
==> sirius: Executing phase: 'build'  
==> sirius: Executing phase: 'install'  
==> sirius: Successfully installed sirius-develop-  
vhzi64vzqgvkcoinf6sfilsirk34avd
```

```
Fetch: 5.45s. Build: 2m 11.94s. Total: 2m 17.39s.  
[+] /home/ubuntu/src/spack/opt/spack/linux-ubuntu20.04-  
haswell/gcc-9.3.0/sirius-develop-  
vhzi64vzqgvkcoinf6sfilsirk34avd
```

- CPU kernels and support libs
- GPU Kernels
- Problem-specific interface layers to CPU/GPU kernels

Supported Functionality

- NC/US/PAW pseudopotentials
- L(A)PW+lo basis with arbitrary number of local orbitals
- Collinear and non-collinear magnetism
- L(S)DA and GGA functionals from LibXC
- Spin-orbit coupling
- Stress tensor (PP-PW only)
- Atomic forces
- Iterative Davidson and exact diagonalization solvers (for both PP-PW and FP-LAPW methods)
- Orbital transformation (wave-function optimisation) method (nlcglib library, PP-PW only)

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

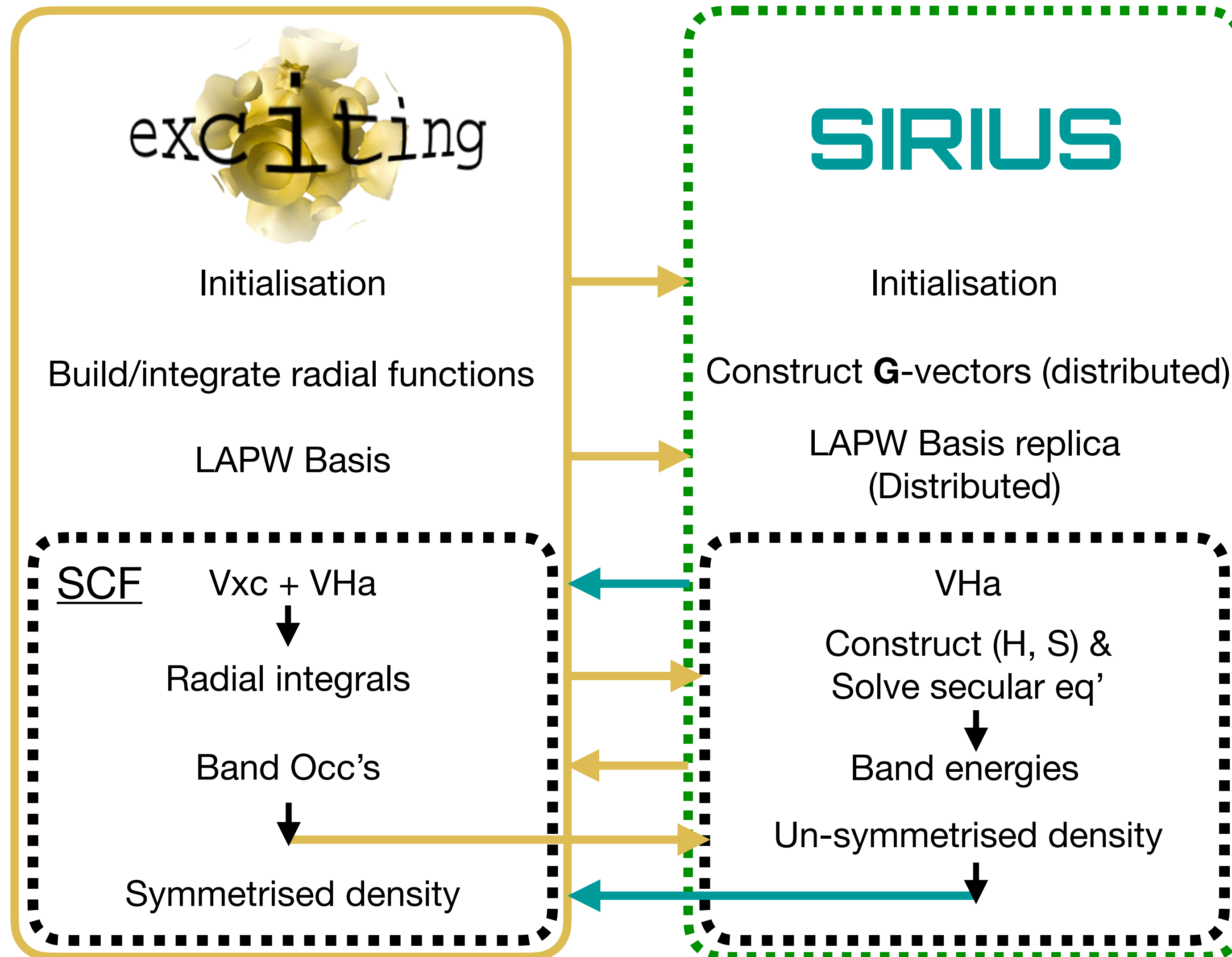
SIRIUS Execution Phase

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

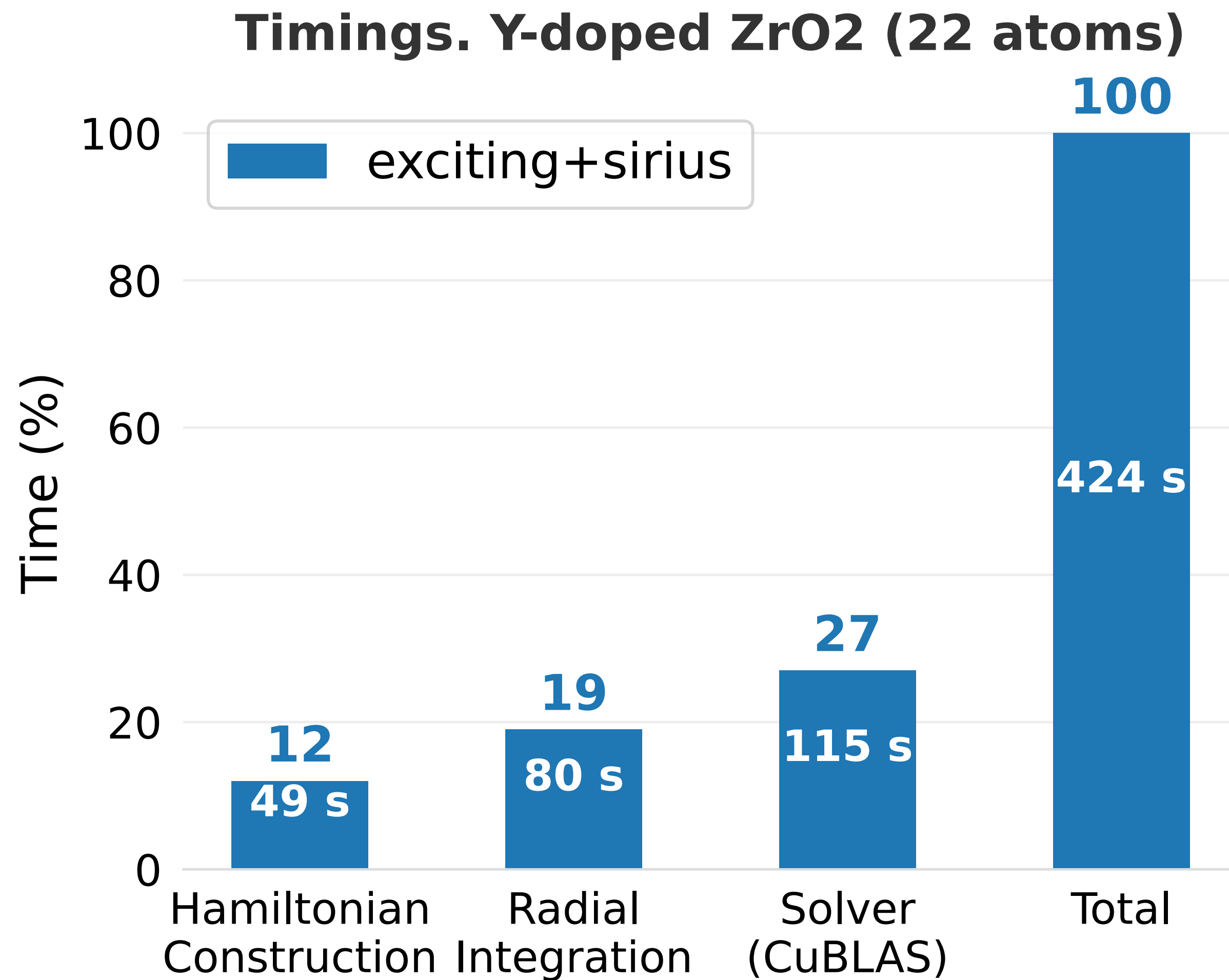
Integration with **exciting**



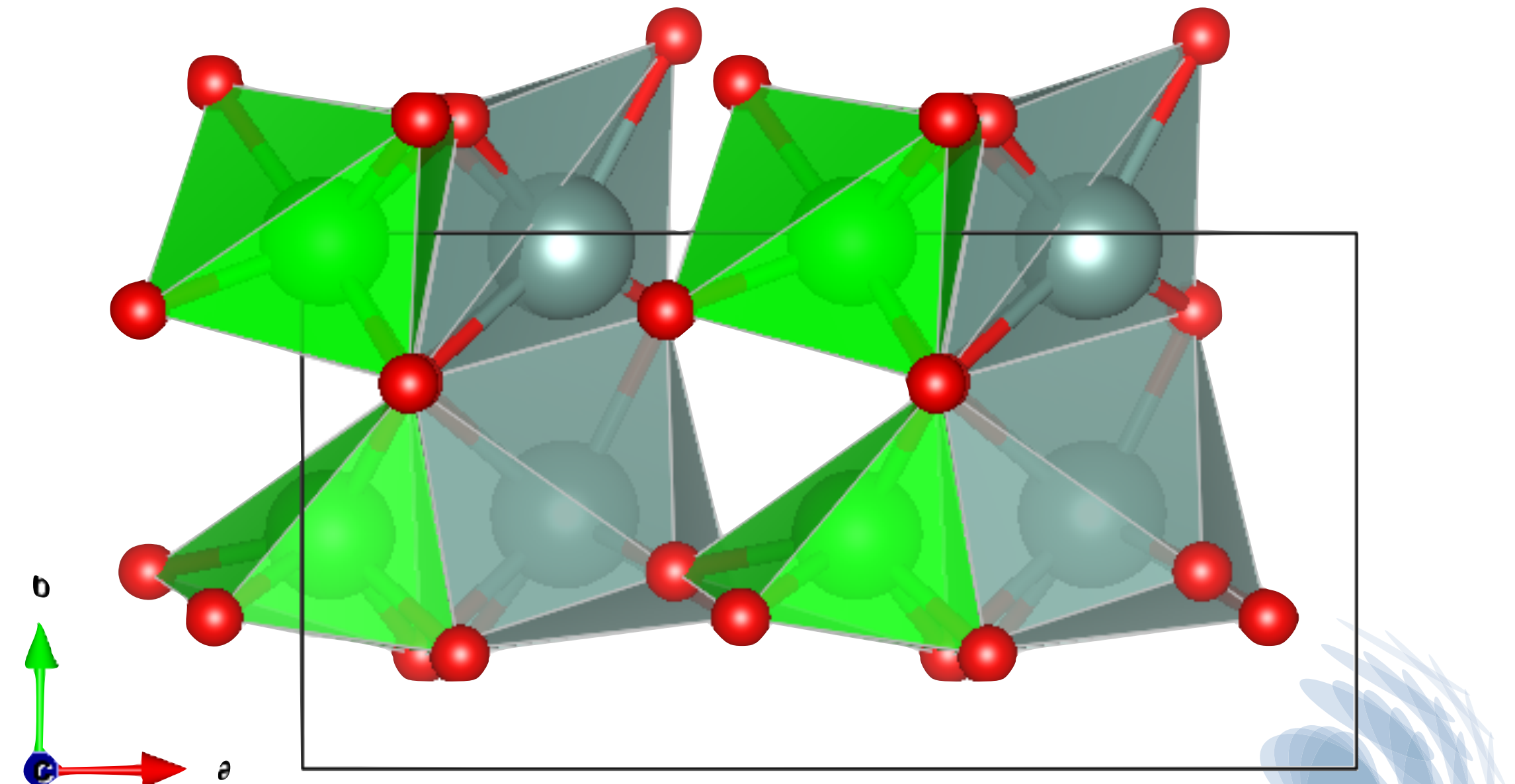
```
</input>
<STRUCTURE>
...
<groundstate
  do="fromscratch"
  rgkmax="8.0"
  ngridk="4 4 4"
  xctype="GGA_PBE_SOL"
  epsengy="1e-07"
  gmaxvr="16.0"
  deband="0.001"
  maxscl="30">
  <sirius xc="false"
    vha="true"
    eigenstates="true"
    density="true"
    densityinit="false"
    cfun="true">
  </sirius>
</groundstate>
</input>
```

2. exciting + SIRIUS Initial Benchmarks

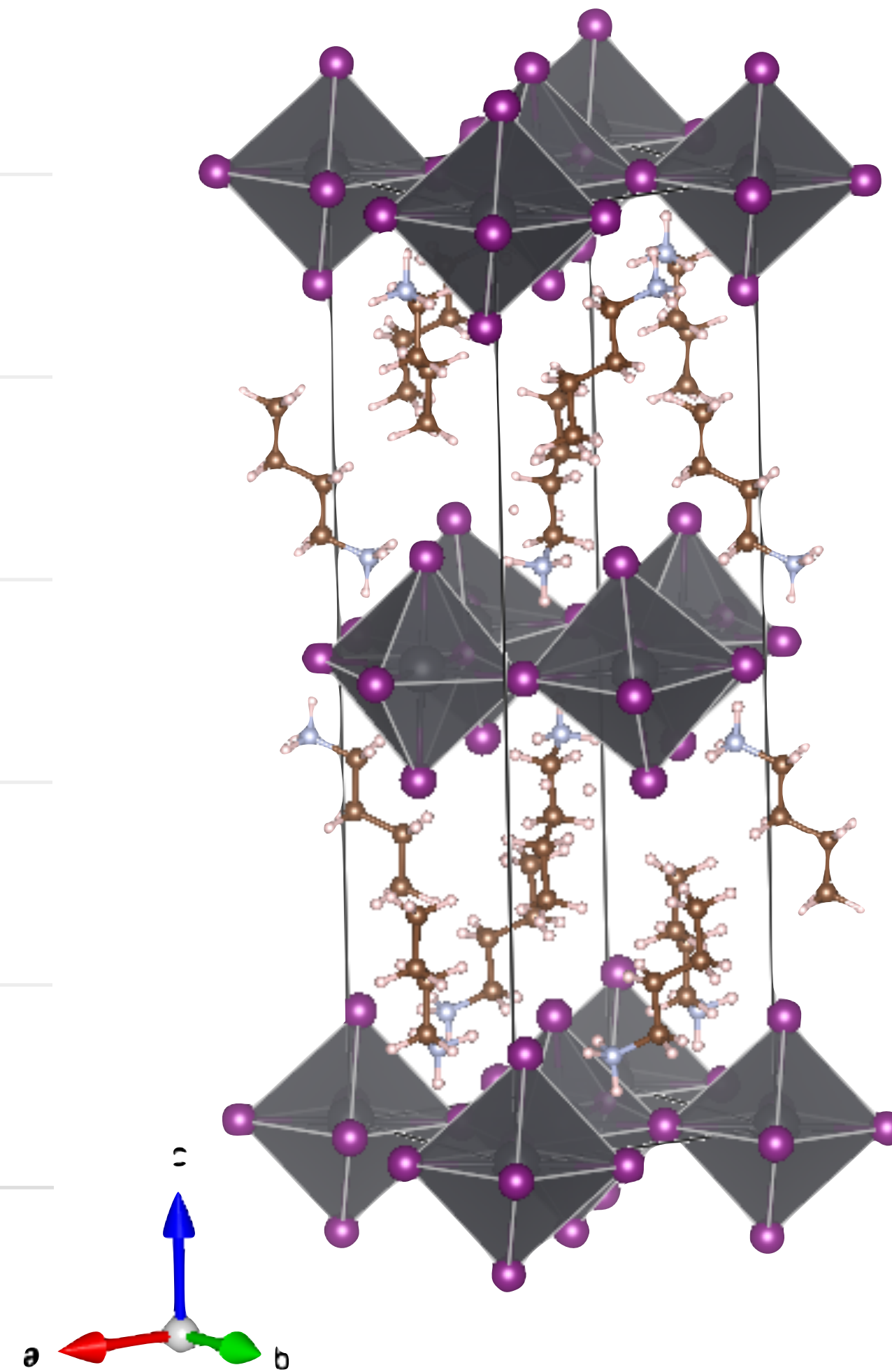
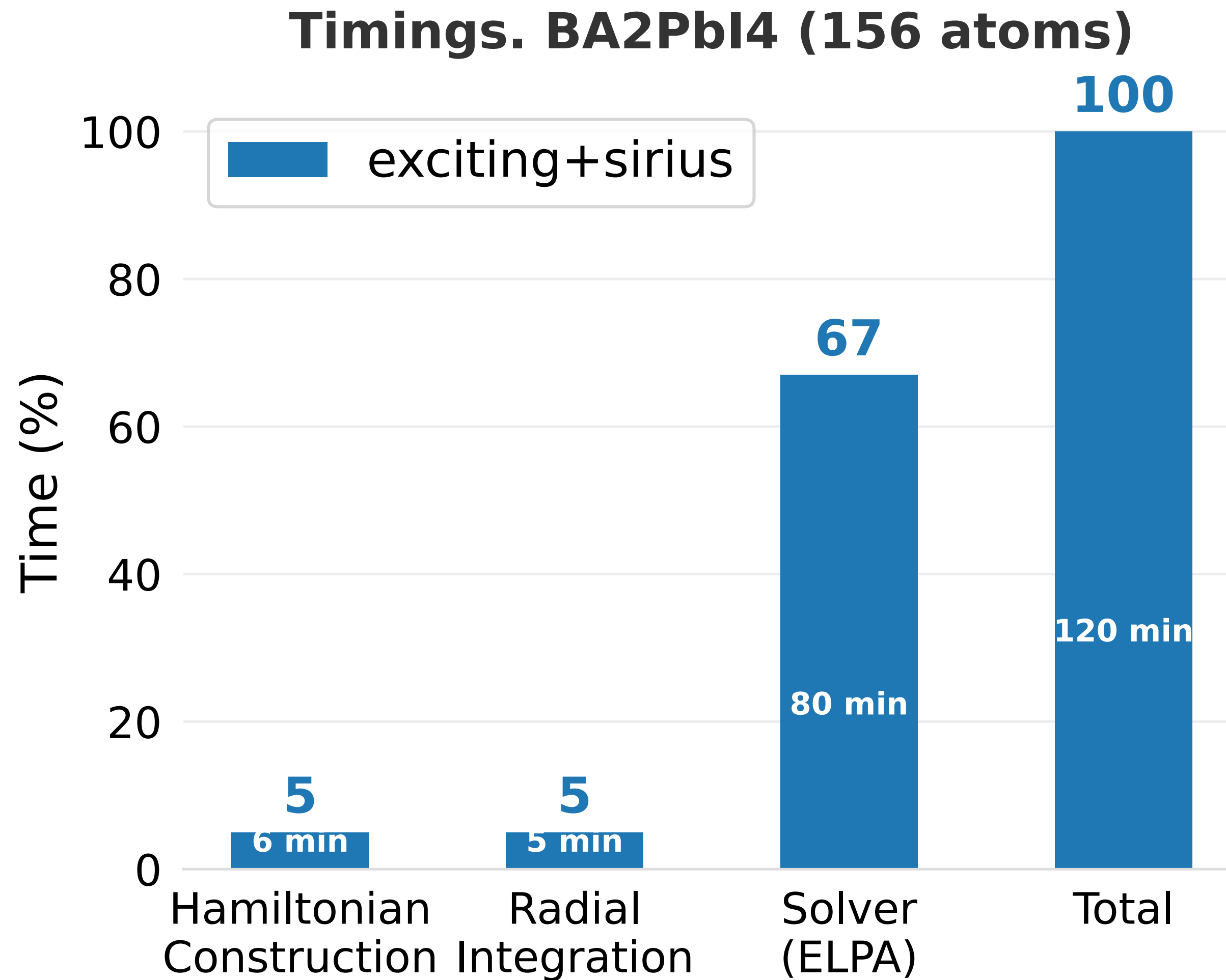
Initial Benchmarks $Zr_4Y_4O_{14}$



Maximum Hamiltonian size	6844
Maximum number of plane-waves	4184
Total number of local-orbitals	2660



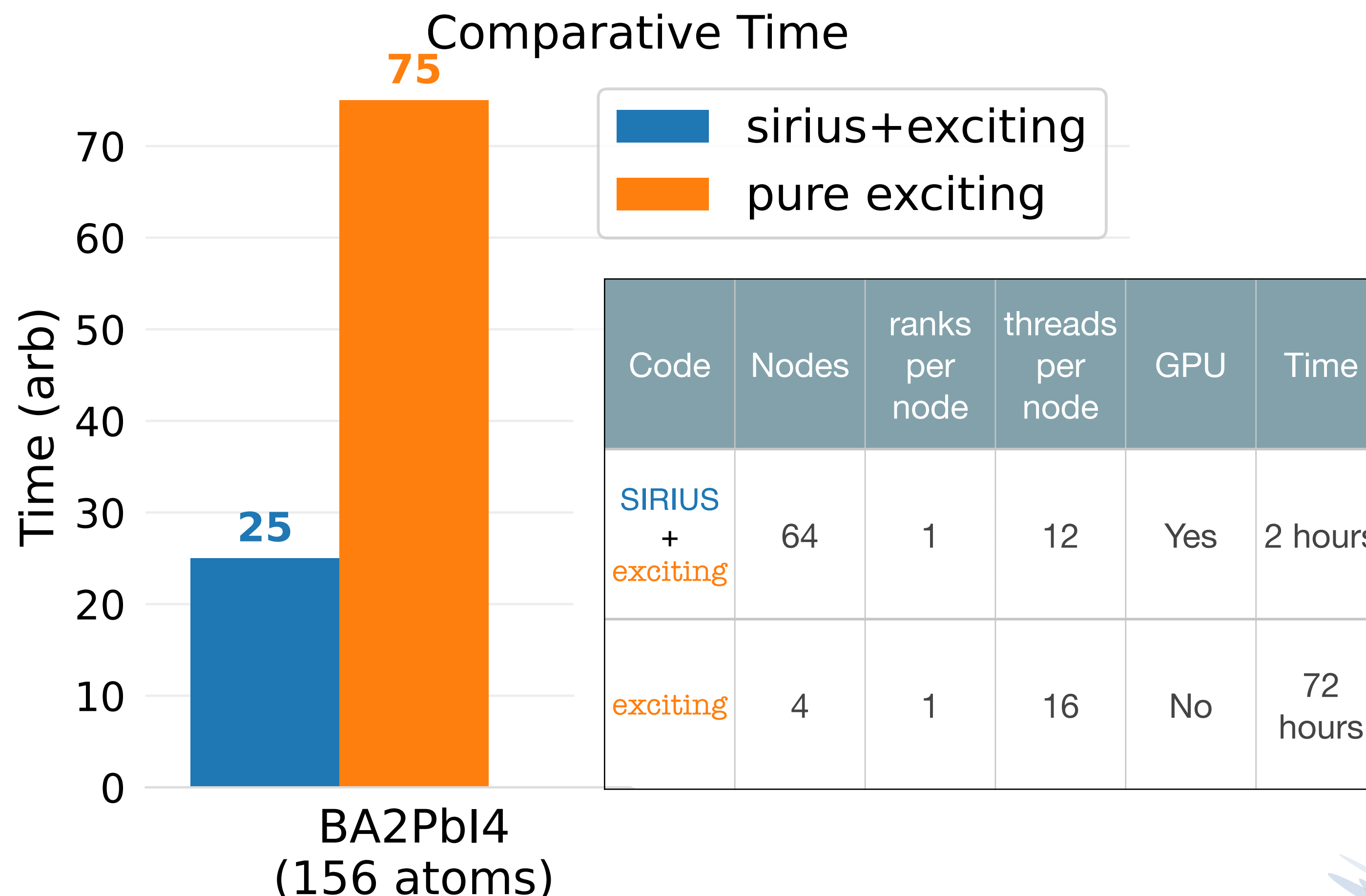
Initial Benchmarks $C_{32}H_{96}N_8Pb_4I_{16}$



Maximum Hamiltonian size	29563
Maximum number of plane-waves	28371
Total number of local-orbitals	1192

Initial Benchmarks

- The comparison is **illustrative** not **quantitative***
- Does not account for differences in:
 - Choice of resources
 - CPU spec (power)
 - Resource assignment effect on performance.
- Plot "time * total CPU cores used"






* A proper benchmark has to be done on the same machine with and without GPU support.

Current Bottlenecks

- Large arrays not distributed in **exciting**:
 - Spherical harmonics expanded in \mathbf{G} : $Y_{lm}(\widehat{\mathbf{G} + \mathbf{k}})$
 - Structure factor: $e^{i(\mathbf{G} + \mathbf{k}) \cdot \boldsymbol{\tau}_\alpha}$
- Structure factor scales as N_{atoms}^2
 - For 200+ atoms, these can easily exceed 1 GB each, limiting number of MPI processes per node.



How do you try out exciting + SIRIUS?

- In the code since **exciting** Fluorine
- **Build Process**
 -  **Spack** : Build sirius dependencies
 -  **Spack** : Build sirius
 - make.inc Build exciting, linking to sirius
- Provide a dockerfile for building all dependencies, which we use in our own CI
-  \$excitingroot/build/utilities/docker/

```
# Dockerfile for exciting plus Sirius dependencies. Anton Kozhevnikov [8 weeks ago] • Fix to Sirius API function:
#
# * Build (from directory containing Dockerfile, else replace . with path/2/Dockerfile)
# docker build -t sirius-cpu:develop-may2023 .
#
# * Run natively
# docker run --entrypoint bash -it sirius-cpu:develop-may2023
FROM ubuntu:focal

ENV DEBIAN_FRONTEND noninteractive

# Lowest-level Build Dependencies
RUN \
apt-get update && \
apt-get install -y apt-utils gcc g++ gfortran git make unzip \
wget pkg-config python3-pip curl tcl m4 cpio hwloc automake \
xsltproc libomp-dev

# Library dependencies which are slow to build with spack
RUN apt-get update && apt-get install -y mpich libhdf5-cpp-103 libhdf5-mpich-103 libhdf5-mpich-dev libxc-dev

RUN pip install cmake==3.24.1

# Install spack
RUN git clone -c feature.manyFiles=true https://github.com/spack/spack.git

RUN echo "source /spack/share/spack/setup-env.sh" >> /etc/profile.d/spack.sh
SHELL ["/bin/bash", "--login", "-c"]

RUN /spack/bin/spack compiler find
RUN /spack/bin/spack external find

# Sirius build specification, at commit: 38988070cda2772f79892f079803875d4835d117
ENV SPEC="sirius@develop %gcc@9.4.0 build_type=Release +scalapack +fortran ^mpich@3.3.2 ^intel-oneapi-mkl+cluster ^spfft target=x86_64"

# Install sirius dependencies
RUN /spack/bin/spack install --fresh --only=dependencies $SPEC

# Sirius installation location
ENV SIRIUS_ROOT=spack-sirius-install

# Location of sirius env `spack.yaml`
ENV SIRIUS_ENV=spack-sirius-env

# Create an environment
RUN /spack/bin/spack env create --with-view $SIRIUS_ROOT $SIRIUS_ENV
RUN /spack/bin/spack -e $SIRIUS_ENV add $SPEC

# exciting test suite dependencies
RUN apt-get update && apt-get install -y python3.8-full python3.8-venv
RUN pip install termcolor lxml pytest pyyaml pytest-cov xmllschema
```

Acknowledgements



- Anton Kozhevnikov (CSCS)
 - Lead developer of Sirius



- Andris Gulans
 - General support with exciting and LAPW



- Cecilia Vona
 - Provided the perovskite system for benchmarking



- Claudia Draxl
 - SOL Group Leader and NOMAD Coordinator

