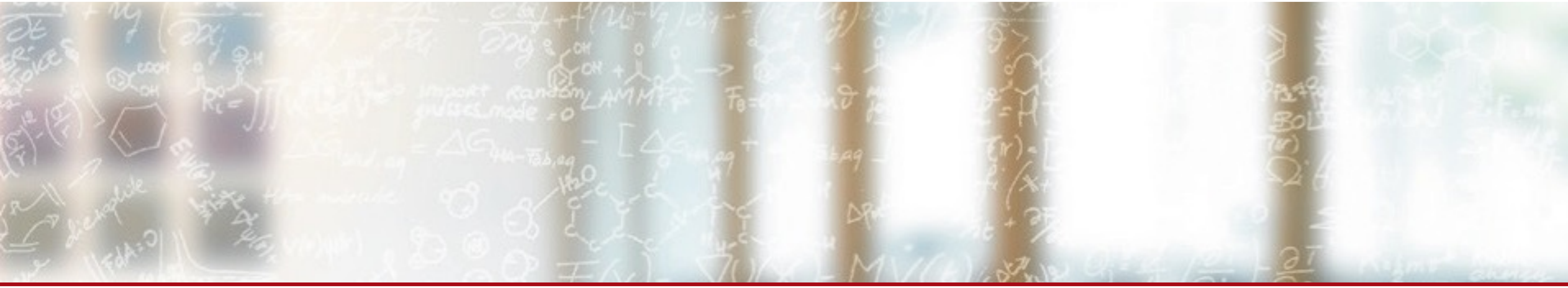
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



GPU acceleration of plane-wave codes using SIRIUS library

Materials Design Ecosystem at the Exascale: High-Performance and High-Throughput Computing

Anton Kozhevnikov, CSCS

January 29, 2018

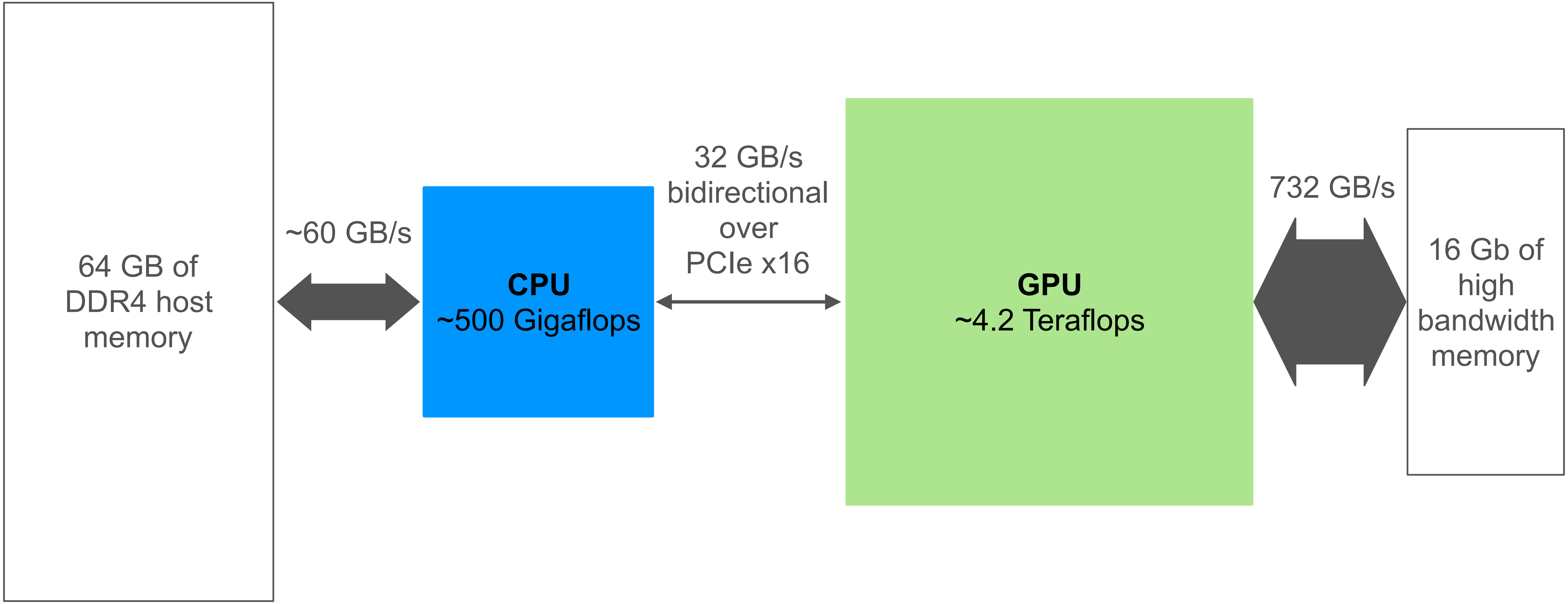
Introduction

Piz Daint: #3 supercomputer in the world

Cray XC50, 5320 nodes

Intel Xeon E5-2690v3 12C, 2.6GHz, 64GB + NVIDIA Tesla P100 16GB
4.761 Teraflops / node

Piz Daint node layout



64 GB of DDR4 host memory

~60 GB/s

CPU
~500 Gigaflops

32 GB/s bidirectional over PCIe x16

GPU
~4.2 Teraflops

732 GB/s

16 Gb of high bandwidth memory

Porting codes to GPUs

No magic “silver bullet” exists!

Porting codes to GPUs

No magic “silver bullet” exists!

Usual steps in porting codes to GPUs

Porting codes to GPUs

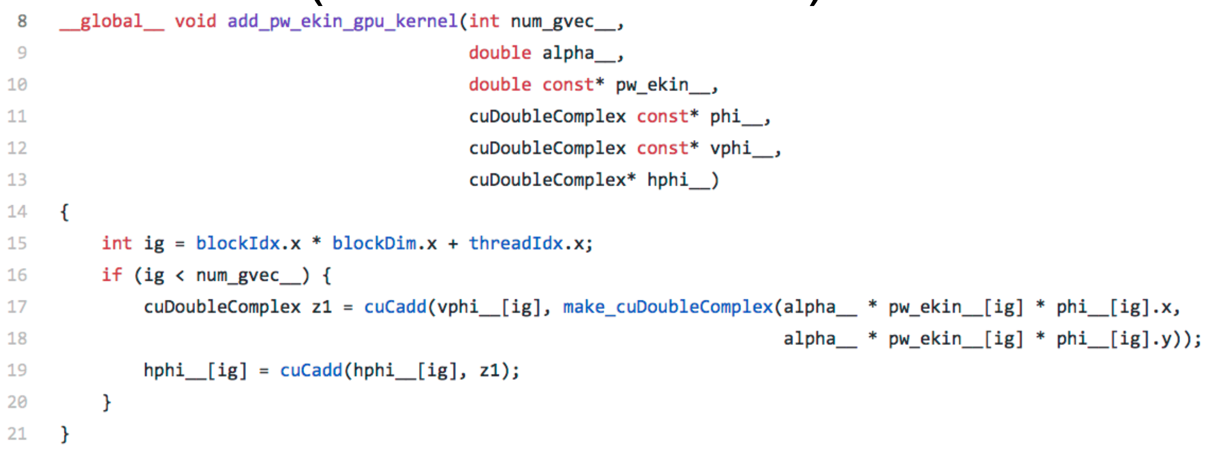
No magic “silver bullet” exists!

Usual steps in porting codes to GPUs

- cleanup and refactor the code
- (possibly) change the data layout
- fully utilize CPU threads and prepare code for node-level parallelization
- move compute-intensive kernels to GPUs

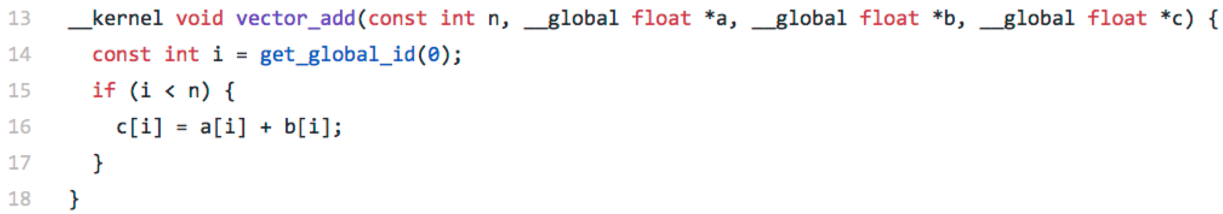
Porting codes to GPUs

- CUDA (C / C++ / Fortran)



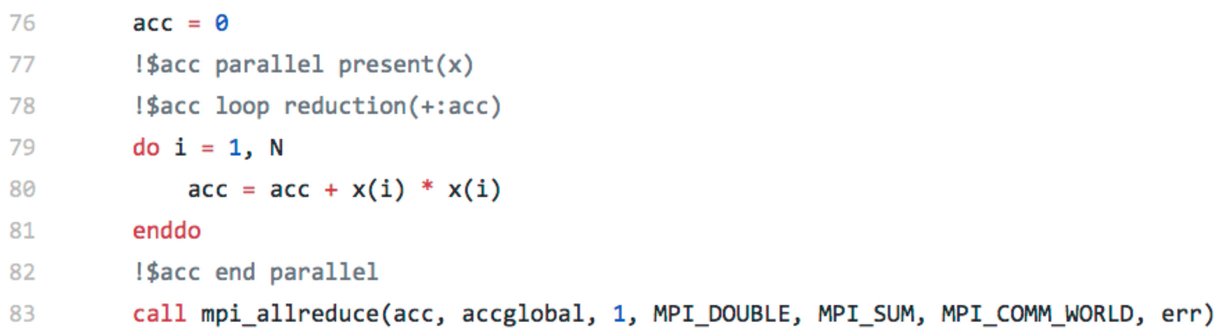
```
__global__ void add_pw_ekin_gpu_kernel(int num_gvec__,
                                       double alpha__,
                                       double const* pw_ekin__,
                                       cuDoubleComplex const* phi__,
                                       cuDoubleComplex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

- OpenCL



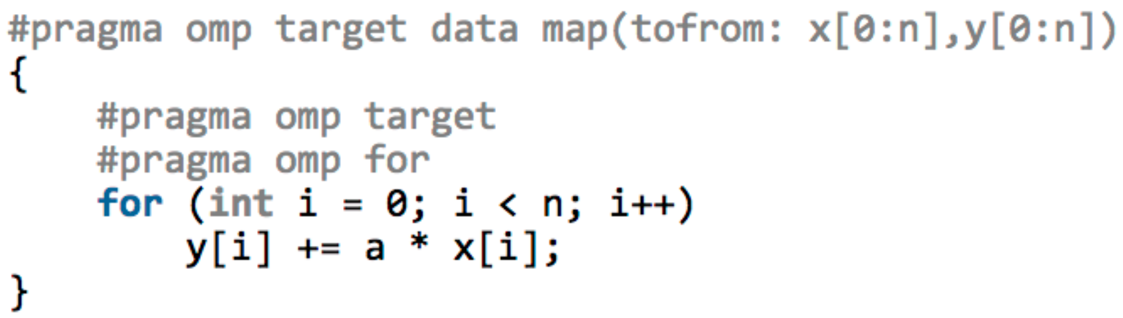
```
__kernel void vector_add(const int n, __global float *a, __global float *b, __global float *c) {
  const int i = get_global_id(0);
  if (i < n) {
    c[i] = a[i] + b[i];
  }
}
```

- OpenACC



```
acc = 0
!$acc parallel present(x)
!$acc loop reduction(+:acc)
do i = 1, N
    acc = acc + x(i) * x(i)
enddo
!$acc end parallel
call mpi_allreduce(acc, accglobal, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD, err)
```

- OpenMP 4.0



```
#pragma omp target data map(tofrom: x[0:n],y[0:n])
{
    #pragma omp target
    #pragma omp for
    for (int i = 0; i < n; i++)
        y[i] += a * x[i];
}
```

Porting codes to GPUs

- CUDA (C / C++ / Fortran)

```
__global__ void add_pw_ekin_gpu_kernel(int num_gvec__,
                                       double alpha__,
                                       double const* pw_ekin__,
                                       cuDoubleComplex const* p
                                       cuDoubleComplex const* v
                                       cuDoubleComplex* hphi__)
{
    int ig = blockIdx.x * blockDim.x + threadIdx.x;
    if (ig < num_gvec__) {
        cuDoubleComplex z1 = cuCadd(vphi__[ig], make_cuDoubleCo

        hphi__[ig] = cuCadd(hphi__[ig], z1);
    }
}
```

- OpenACC

```
acc = 0
!$acc parallel present(x)
!$acc loop reduction(+:acc)
do i = 1, N
    acc = acc + x(i) * x(i)
enddo
!$acc end parallel
call mpi_allreduce(acc, accglobal, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD, err)
```

- OpenCL

```
__kernel void vector_add(const int n, __global float *a, __global float *b, __global float *c) {
                                _id(0);
```

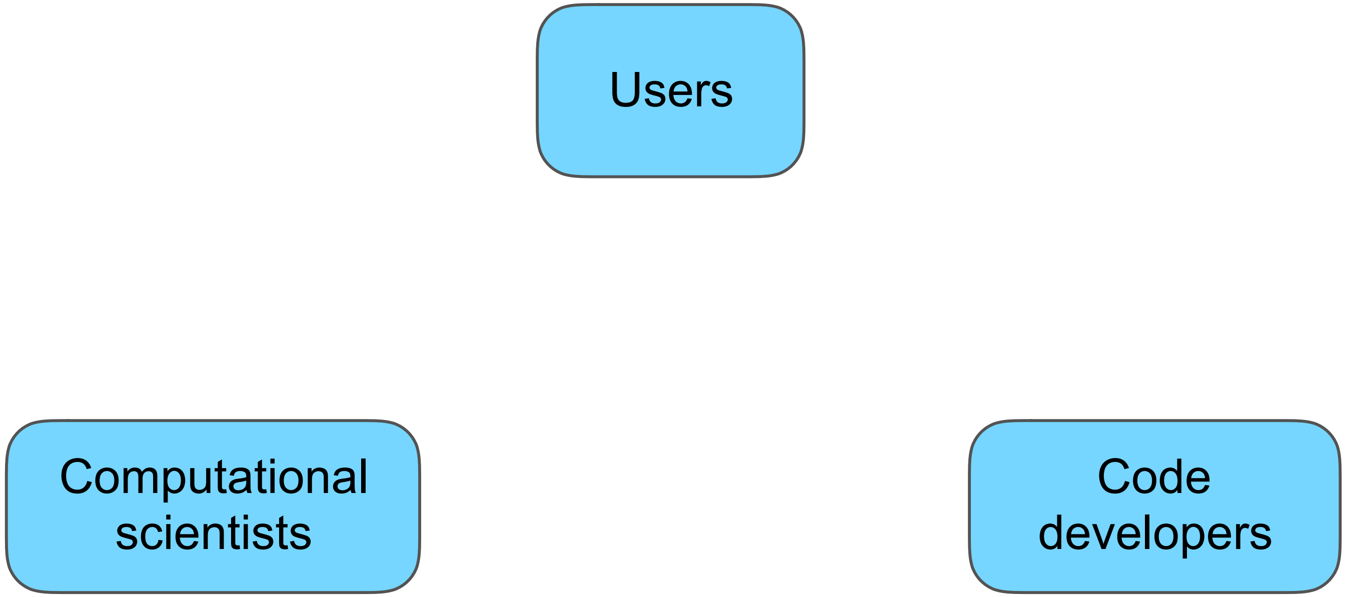
4.0

```
       arget data map(tofrom: x[0:n],y[0:n])

       mp target
  #pragma omp for
  for (int i = 0; i < n; i++)
      y[i] += a * x[i];
}
```

Why do we need a separation of concerns?

Why do we need a separation of concerns?

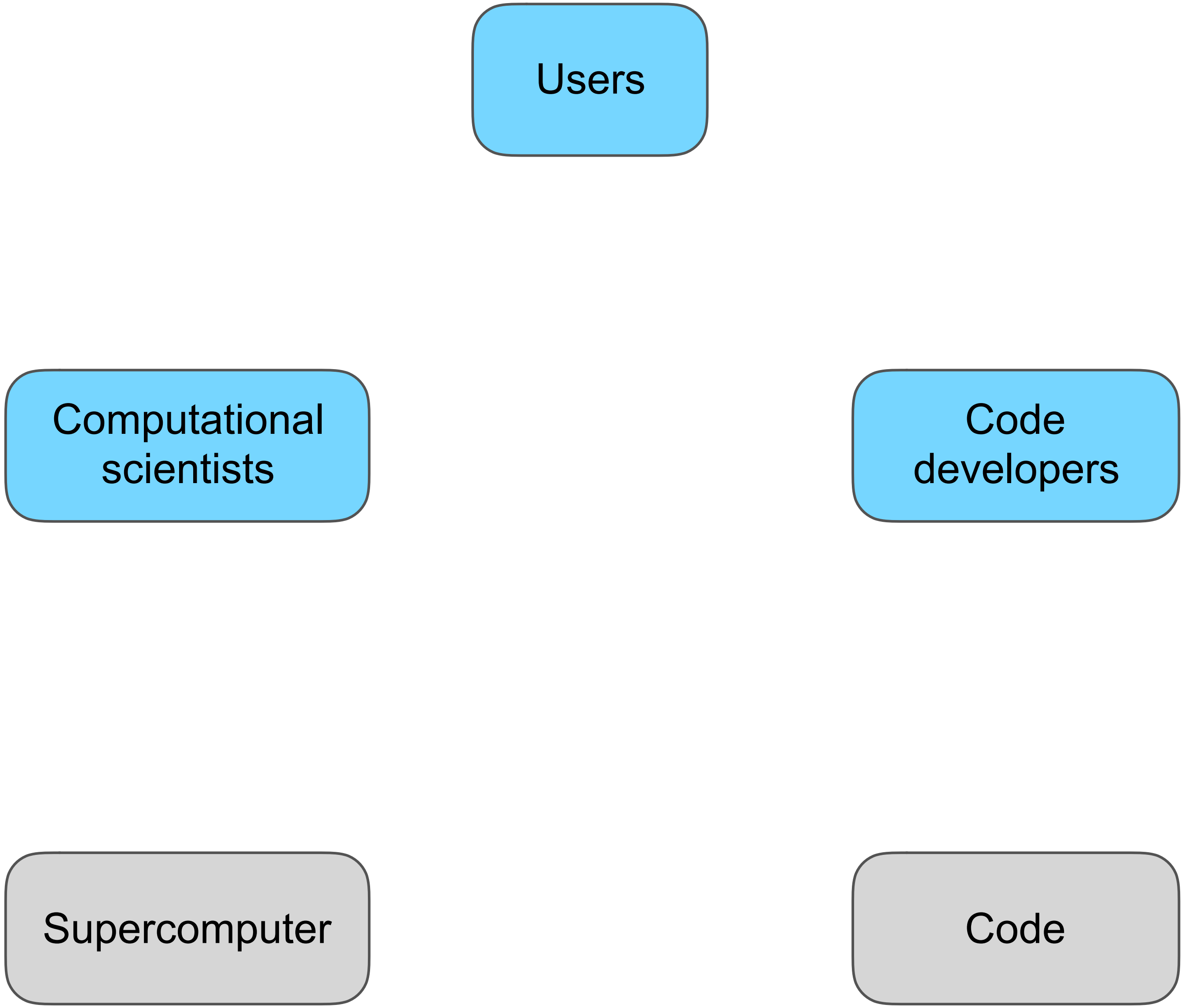


Users

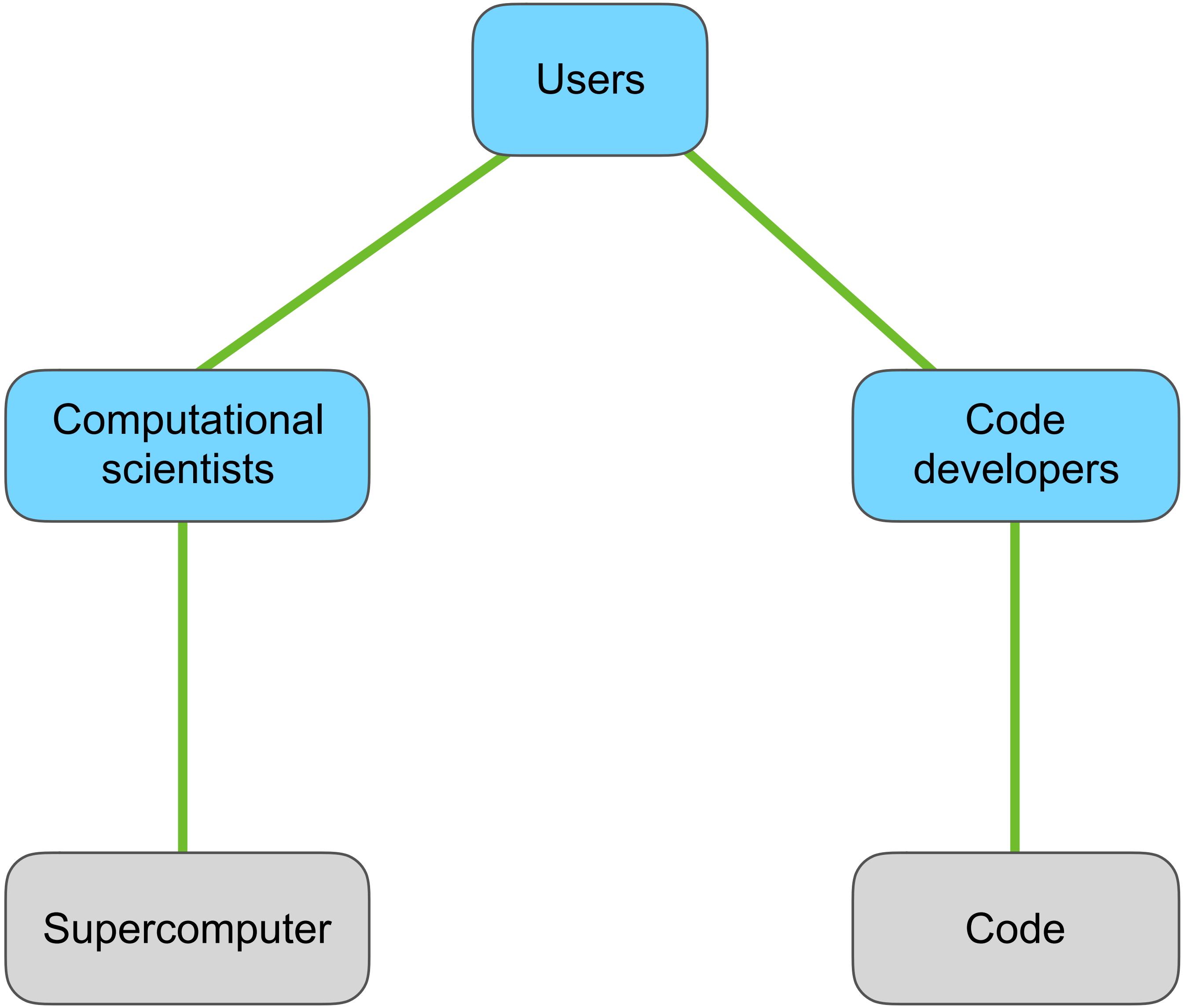
Computational scientists

Code developers

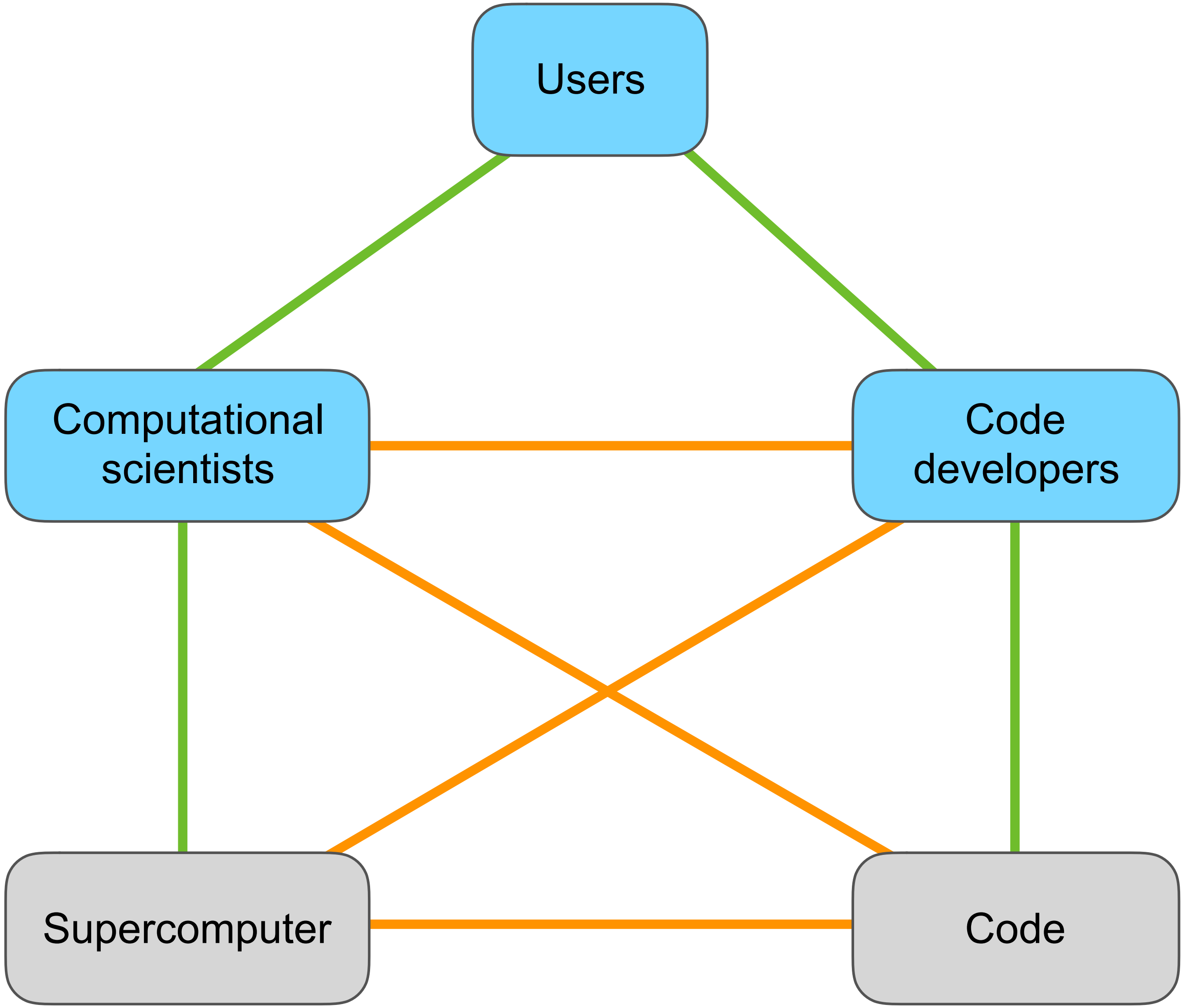
Why do we need a separation of concerns?



Why do we need a separation of concerns?



Why do we need a separation of concerns?

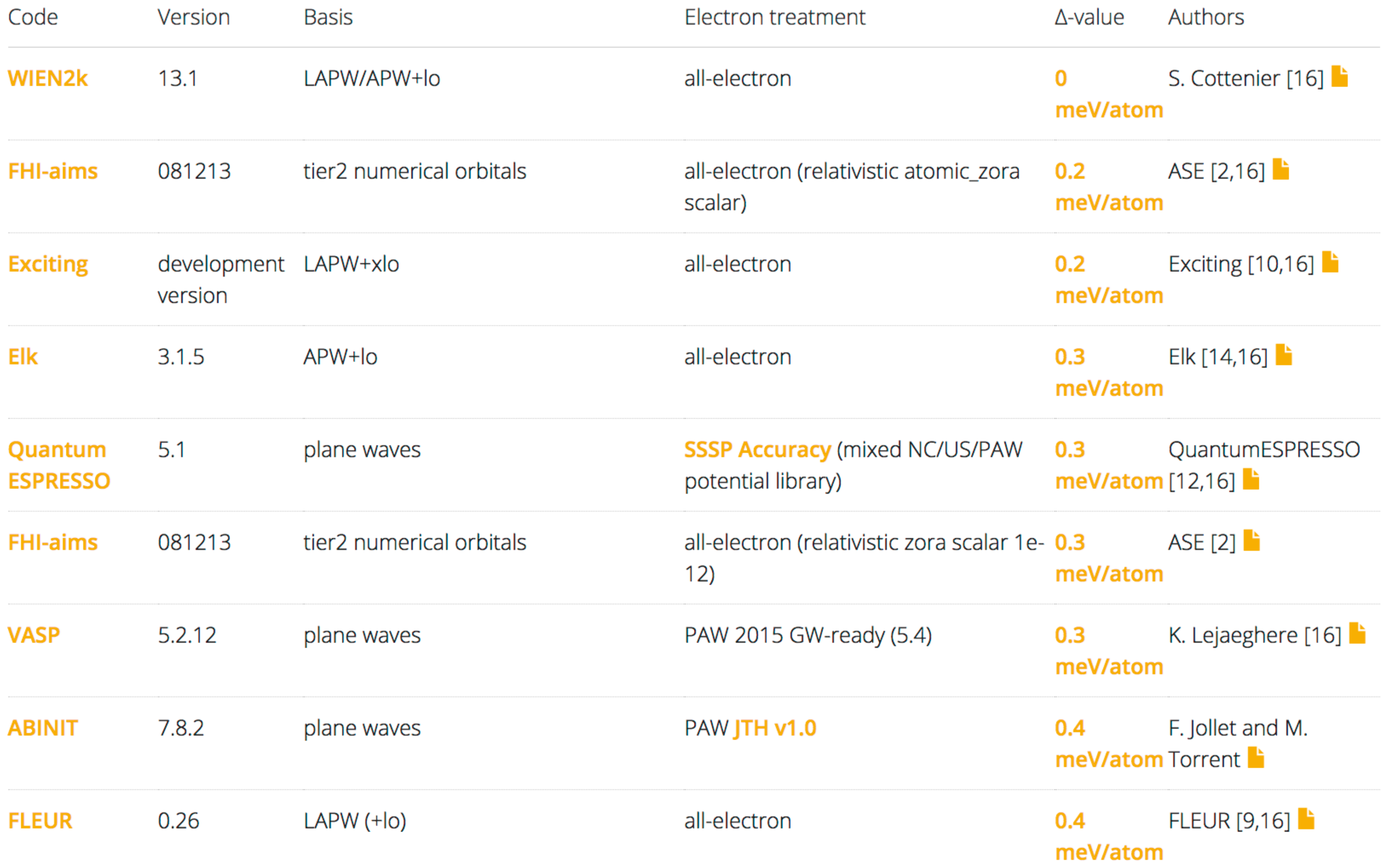


Electronic-structure codes

Electronic structure codes

Basis functions for KS states Atomic potential treatment	Periodic Bloch functions (plane-waves or similar)	Localized orbitals
Full-potential	FLEUR Wien2K Exciting Elk	FHI-aims FPLO
Pseudo-potential	VASP CPMD Quantum ESPRESSO Abinit Qbox	CP2K SIESTA OpenMX

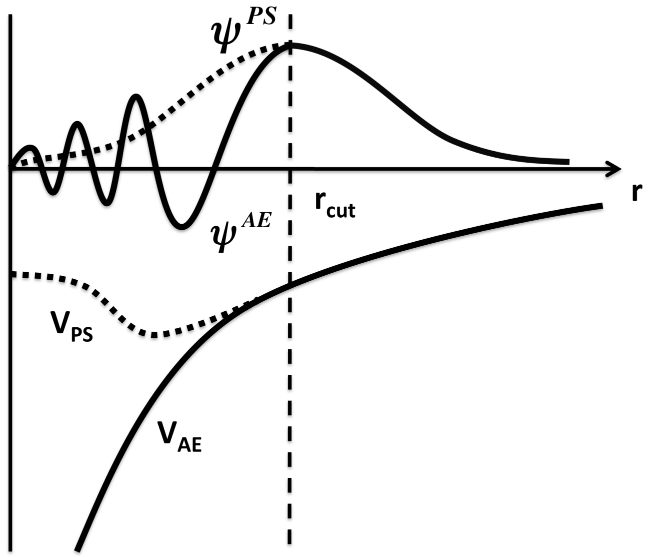
Delta DFT codes effort



Code	Version	Basis	Electron treatment	Δ-value	Authors
WIEN2k	13.1	LAPW/APW+lo	all-electron	0 meV/atom	S. Cottenier [16]
FHI-aims	081213	tier2 numerical orbitals	all-electron (relativistic atomic_zora scalar)	0.2 meV/atom	ASE [2,16]
Exciting	development version	LAPW+xlo	all-electron	0.2 meV/atom	Exciting [10,16]
Elk	3.1.5	APW+lo	all-electron	0.3 meV/atom	Elk [14,16]
Quantum ESPRESSO	5.1	plane waves	SSSP Accuracy (mixed NC/US/PAW potential library)	0.3 meV/atom	QuantumESPRESSO [12,16]
FHI-aims	081213	tier2 numerical orbitals	all-electron (relativistic zora scalar 1e-12)	0.3 meV/atom	ASE [2]
VASP	5.2.12	plane waves	PAW 2015 GW-ready (5.4)	0.3 meV/atom	K. Lejaeghere [16]
ABINIT	7.8.2	plane waves	PAW JTH v1.0	0.4 meV/atom	F. Jollet and M. Torrent
FLEUR	0.26	LAPW (+lo)	all-electron	0.4 meV/atom	FLEUR [9,16]

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 $\hat{V}_{\mathrm{PS}} = V_{loc}(\mathbf{r}) + \sum_{\alpha} \sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$

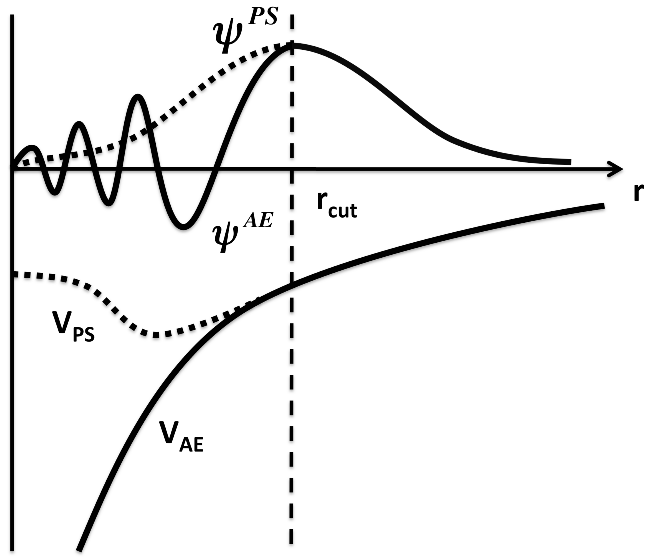


Pseudopotential plane-wave method

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

Basis functions:

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

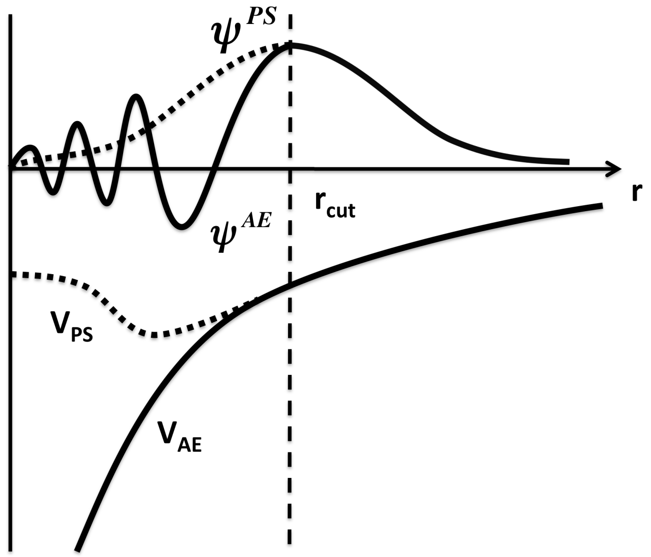


Pseudopotential plane-wave method

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

Basis functions:

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



Potential and density:

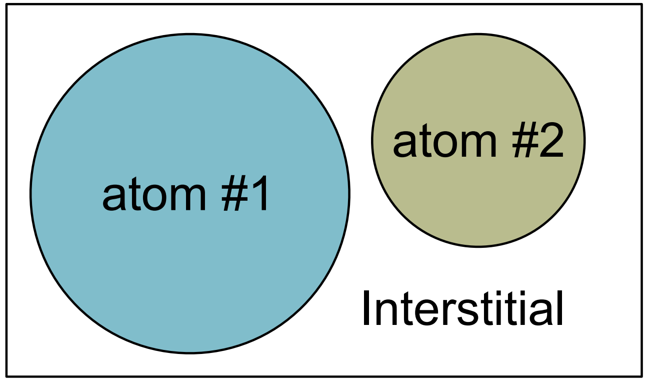
$$V(\mathbf{r}) = \sum_{\mathbf{G}} V(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} \qquad \rho(\mathbf{r}) = \sum_{\mathbf{G}} \rho(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}}$$

Pseudopotential plane-wave method

- Approximation to atomic potential
- Core states are excluded
- Number of basis functions: ~1000 / atom
- Number of valence states: ~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

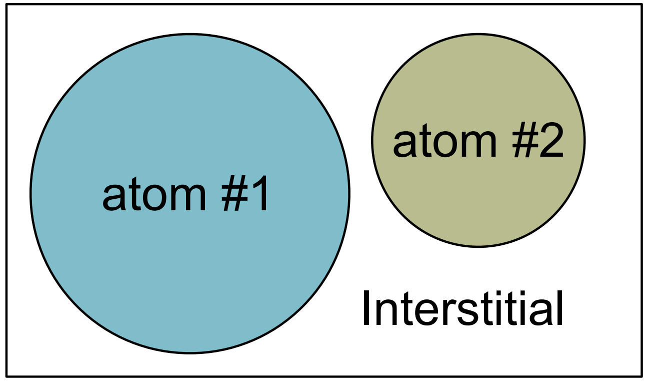


Full-potential linearized augmented plane-wave method

- 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:

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

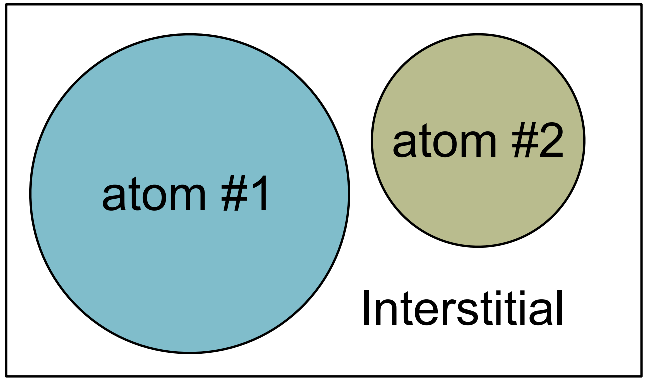


Full-potential linearized augmented plane-wave method

- 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:

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



Potential and density:

$$V(\mathbf{r}) = \begin{cases} \displaystyle\sum_{\ell m} V^\alpha_{\ell m}(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \mathrm{MT}\alpha \\ \displaystyle\sum_{\mathbf{G}} V(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} & \mathbf{r} \in \mathrm{I} \end{cases} \qquad \rho(\mathbf{r}) = \begin{cases} \displaystyle\sum_{\ell m} \rho^\alpha_{\ell m}(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \mathrm{MT}\alpha \\ \displaystyle\sum_{\mathbf{G}} \rho(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} & \mathbf{r} \in \mathrm{I} \end{cases}$$

Full-potential linearized augmented plane-wave method

- No approximation to atomic potential
- Core states are included
- Number of basis functions: ~100 / atom
- Number of valence states: ~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 often required)

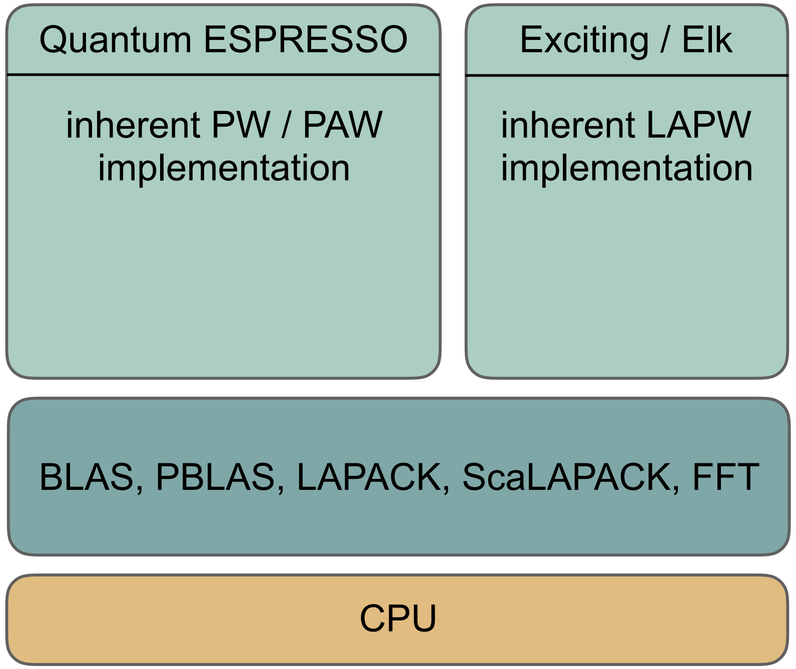
Common features of the FP-LAPW and PP-PW methods

- Definition of the unit cell (atoms, atom types, lattice vectors, symmetry operations, etc.)
- Definition of the reciprocal lattice, plane-wave cutoffs, **G** vectors, **G+k** vectors
- Definition of the wave-functions
- 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.)

SIRIUS library

Motivation for a common domain specific library

Extend the legacy Fortran codes with the API calls to a domain-specific library which runs on GPUs and other novel architectures.



Quantum ESPRESSO

inherent PW / PAW implementation

Exciting / Elk

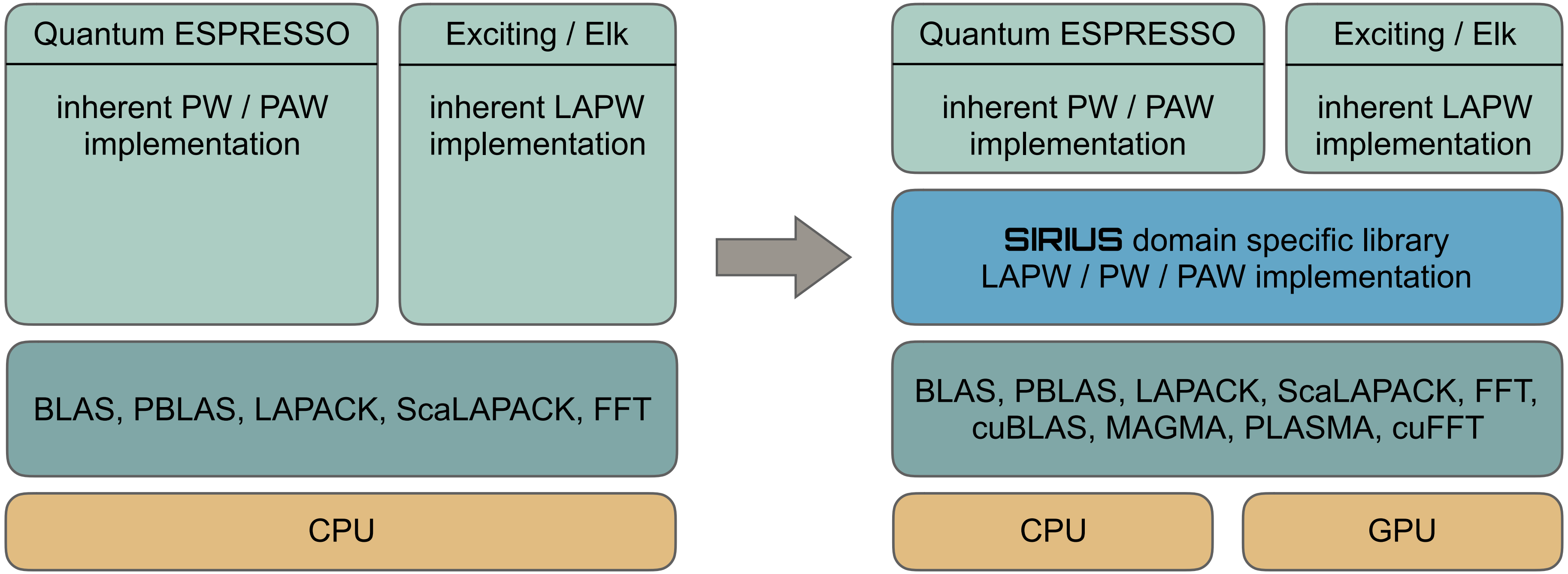
inherent LAPW implementation

BLAS, PBLAS, LAPACK, ScaLAPACK, FFT

CPU

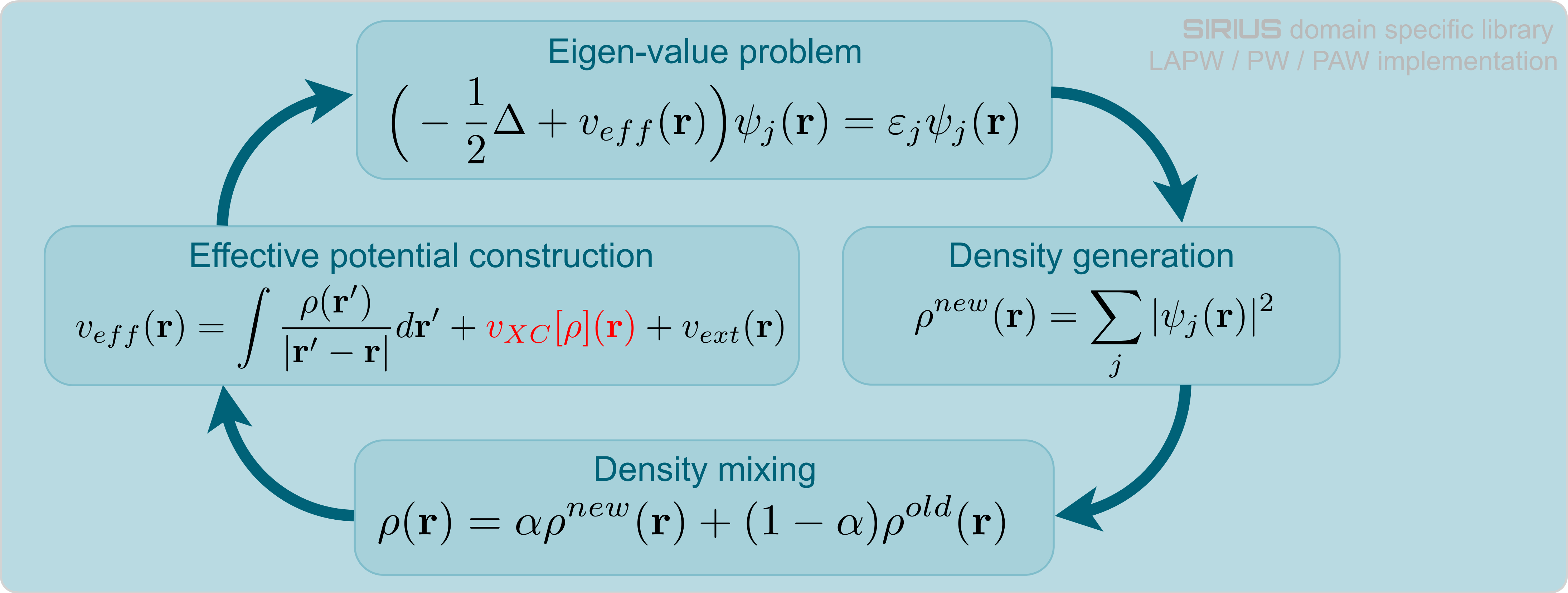
Motivation for a common domain specific library

Extend the legacy Fortran codes with the API calls to a domain-specific library which runs on GPUs and other novel architectures.



Quantum ESPRESSO
inherent PW / PAW implementation
Exciting / Elk
inherent LAPW implementation
BLAS, PBLAS, LAPACK, ScaLAPACK, FFT
CPU
Quantum ESPRESSO
inherent PW / PAW implementation
Exciting / Elk
inherent LAPW implementation
SIRIUS domain specific library
LAPW / PW / PAW implementation
BLAS, PBLAS, LAPACK, ScaLAPACK, FFT, cuBLAS, MAGMA, PLASMA, cuFFT
CPU
GPU

Where to draw the line?



SIRIUS domain specific library
LAPW / PW / PAW implementation

Eigen-value problem

$$\Big(-\frac{1}{2}\Delta + v_{eff}(\mathbf{r})\Big)\psi_j(\mathbf{r}) = \varepsilon_j\psi_j(\mathbf{r})$$

Density generation

$$\rho^{new}(\mathbf{r}) = \sum_j |\psi_j(\mathbf{r})|^2$$

Density mixing

$$\rho(\mathbf{r}) = \alpha\rho^{new}(\mathbf{r}) + (1-\alpha)\rho^{old}(\mathbf{r})$$

Effective potential construction

$$v_{eff}(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}'-\mathbf{r}|}d\mathbf{r}' + v_{XC}[\rho](\mathbf{r}) + v_{ext}(\mathbf{r})$$

Output:

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

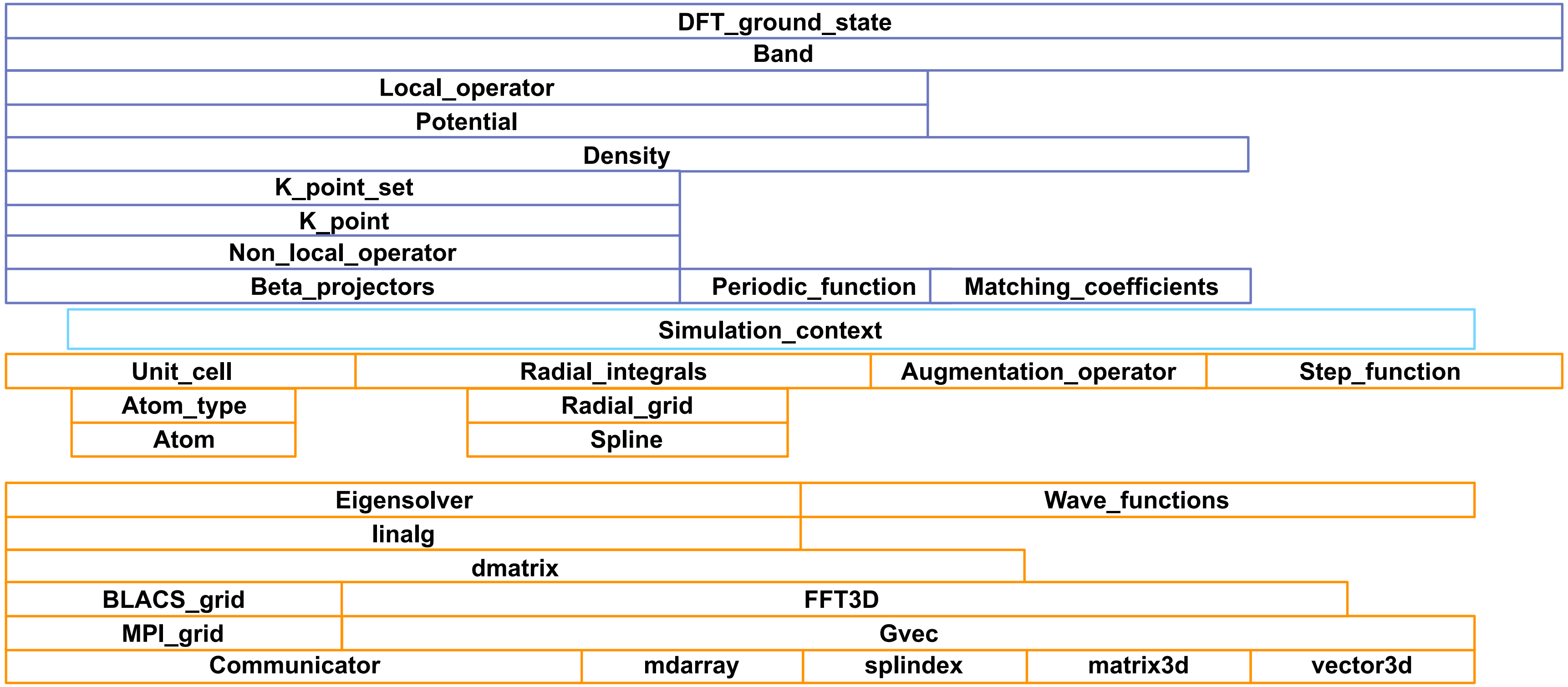
- full-potential (L)APW+lo
  - non-magnetic, collinear and non-collinear magnetic ground states
  - non-relativistic, ZORA and IORA valence solvers
  - Dirac solver for core states

- norm-conserving, ultrasoft and PAW pseudopotentials
  - non-magnetic, collinear and non-collinear magnetic ground states
  - spin-orbit correction
  - atomic forces
  - stress tensor
  - Gamma-point case

SIRIUS library

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

SIRIUS is a collection of classes that abstract away the different building blocks of PW and LAPW codes. The class composition hierarchy starts from the most primitive classes (**Communicator**, **mdarray**, etc.) and progresses towards several high-level classes (**DFT_ground_state**, **Band**, **Potential**, etc.). The code is written in C++11 with MPI, OpenMP and CUDA programming models.



DFT_ground_state

Band

Local_operator

Potential

Density

K_point_set

K_point

Non_local_operator

Beta_projectors | Periodic_function | Matching_coefficients

Simulation_context

Unit_cell | Radial_integrals | Augmentation_operator | Step_function

Atom_type | Radial_grid

Atom | Spline

Eigensolver | Wave_functions

linalg

dmatrix

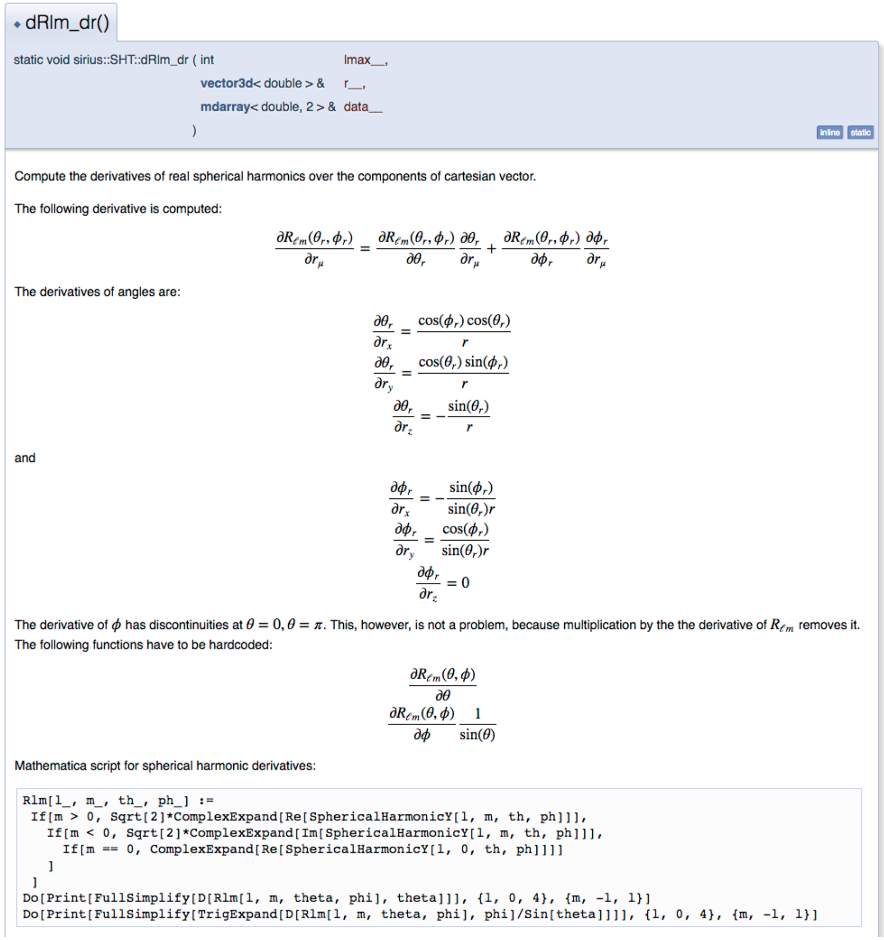
BLACS_grid | FFT3D

MPI_grid | Gvec

Communicator | mdarray | splindex | matrix3d | vector3d

Doxygen documentation

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



◆ dRlm_dr()

```
static void sirius::SHT::dRlm_dr ( int                        lmax__,
                                    vector3d< double > &       r__,
                                    mdarray< double, 2 > &     data__
                                  )
```

inline static

Compute the derivatives of real spherical harmonics over the components of cartesian vector.

The following derivative is computed:

$$\frac{\partial R_{\ell m}(\theta_r, \phi_r)}{\partial r_{\mu}} = \frac{\partial R_{\ell m}(\theta_r, \phi_r)}{\partial \theta_r} \frac{\partial \theta_r}{\partial r_{\mu}} + \frac{\partial R_{\ell m}(\theta_r, \phi_r)}{\partial \phi_r} \frac{\partial \phi_r}{\partial r_{\mu}}$$

The derivatives of angles are:

$$\frac{\partial \theta_r}{\partial r_x} = \frac{\cos(\phi_r)\cos(\theta_r)}{r}$$
$$\frac{\partial \theta_r}{\partial r_y} = \frac{\cos(\theta_r)\sin(\phi_r)}{r}$$
$$\frac{\partial \theta_r}{\partial r_z} = -\frac{\sin(\theta_r)}{r}$$

and

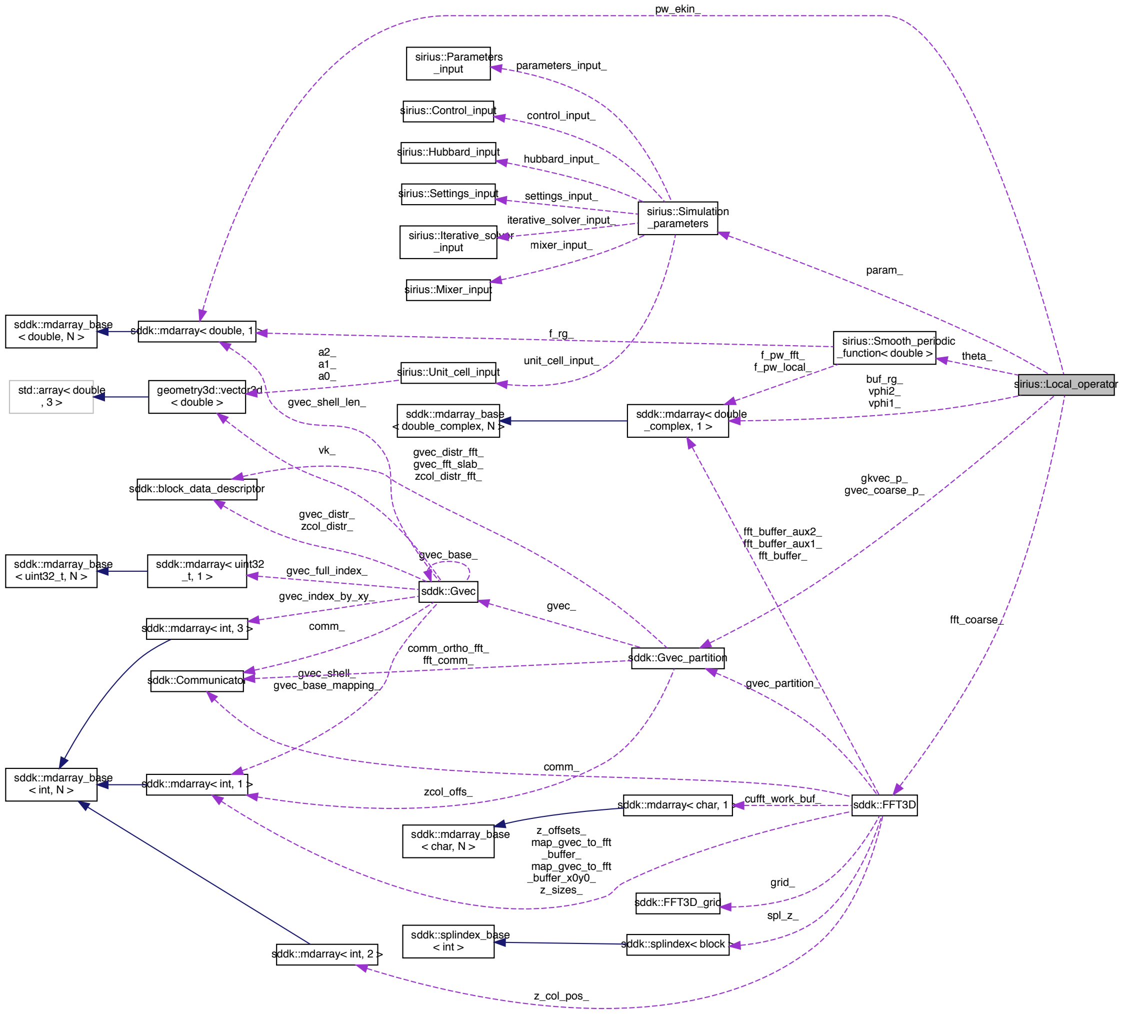
$$\frac{\partial \phi_r}{\partial r_x} = -\frac{\sin(\phi_r)}{\sin(\theta_r) r}$$
$$\frac{\partial \phi_r}{\partial r_y} = \frac{\cos(\phi_r)}{\sin(\theta_r) r}$$
$$\frac{\partial \phi_r}{\partial r_z} = 0$$

The derivative of ϕ has discontinuities at $\theta = 0, \theta = \pi$. This, however, is not a problem, because multiplication by the the derivative of $R_{\ell m}$ removes it. The following functions have to be hardcoded:

$$\frac{\partial R_{\ell m}(\theta, \phi)}{\partial \theta}$$
$$\frac{\partial R_{\ell m}(\theta, \phi)}{\partial \phi} \frac{1}{\sin(\theta)}$$

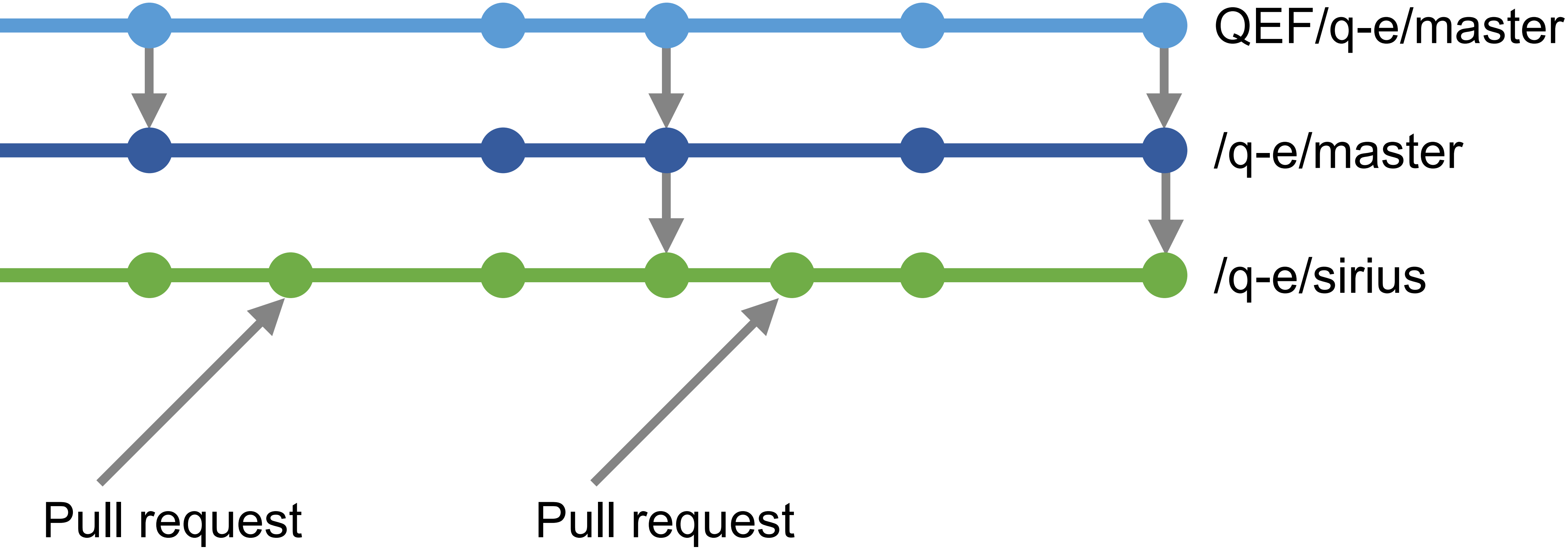
Mathematica script for spherical harmonic derivatives:

```
Rlm[l_, m_, th_, ph_] :=
 If[m > 0, Sqrt[2]*ComplexExpand[Re[SphericalHarmonicY[l, m, th, ph]]],
   If[m < 0, Sqrt[2]*ComplexExpand[Im[SphericalHarmonicY[l, m, th, ph]]],
     If[m == 0, ComplexExpand[Re[SphericalHarmonicY[l, 0, th, ph]]]]]
   ]
 ]
Do[Print[FullSimplify[D[Rlm[l, m, theta, phi], theta]]], {l, 0, 4}, {m, -1, 1}]
Do[Print[FullSimplify[TrigExpand[D[Rlm[l, m, theta, phi], phi]/Sin[theta]]]], {l, 0, 4}, {m, -1, 1}]
```

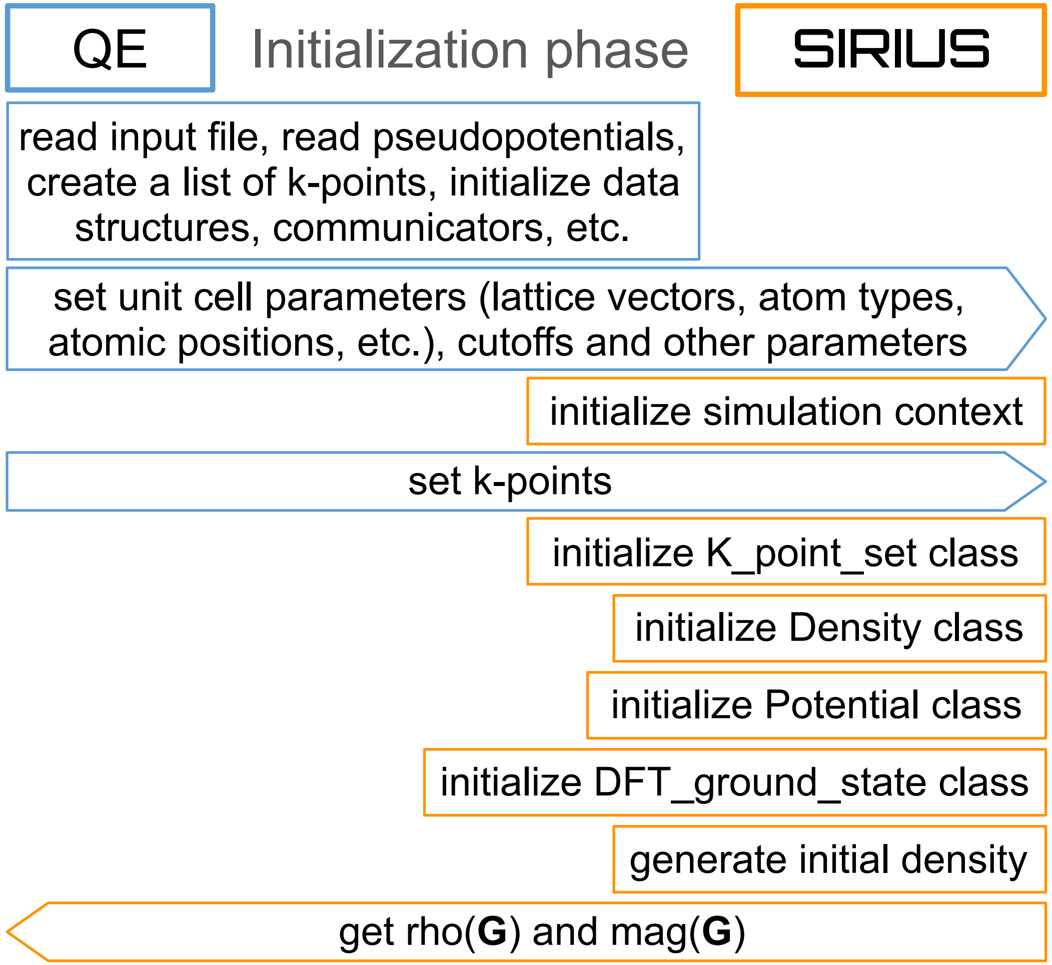


Development cycle

https://github.com/electronic-structure/q-e



Example of QE/SIRIUS interoperability



QE

Initialization phase

SIRIUS

read input file, read pseudopotentials, create a list of k-points, initialize data structures, communicators, etc.

set unit cell parameters (lattice vectors, atom types, atomic positions, etc.), cutoffs and other parameters

initialize simulation context

set k-points

initialize K_point_set class

initialize Density class

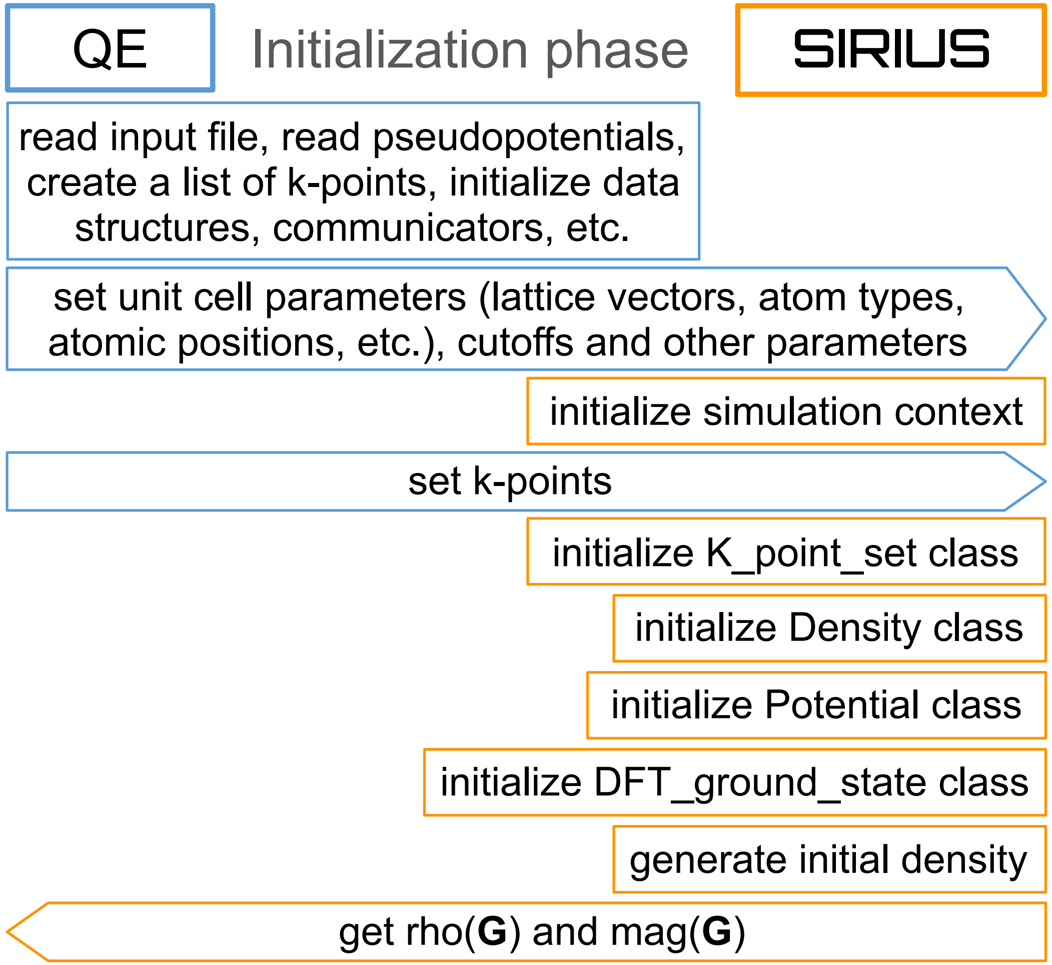
initialize Potential class

initialize DFT_ground_state class

generate initial density

get rho(**G**) and mag(**G**)

Example of QE/SIRIUS interoperability



QE

Initialization phase

SIRIUS

read input file, read pseudopotentials, create a list of k-points, initialize data structures, communicators, etc.

set unit cell parameters (lattice vectors, atom types, atomic positions, etc.), cutoffs and other parameters

initialize simulation context

set k-points

initialize K_point_set class

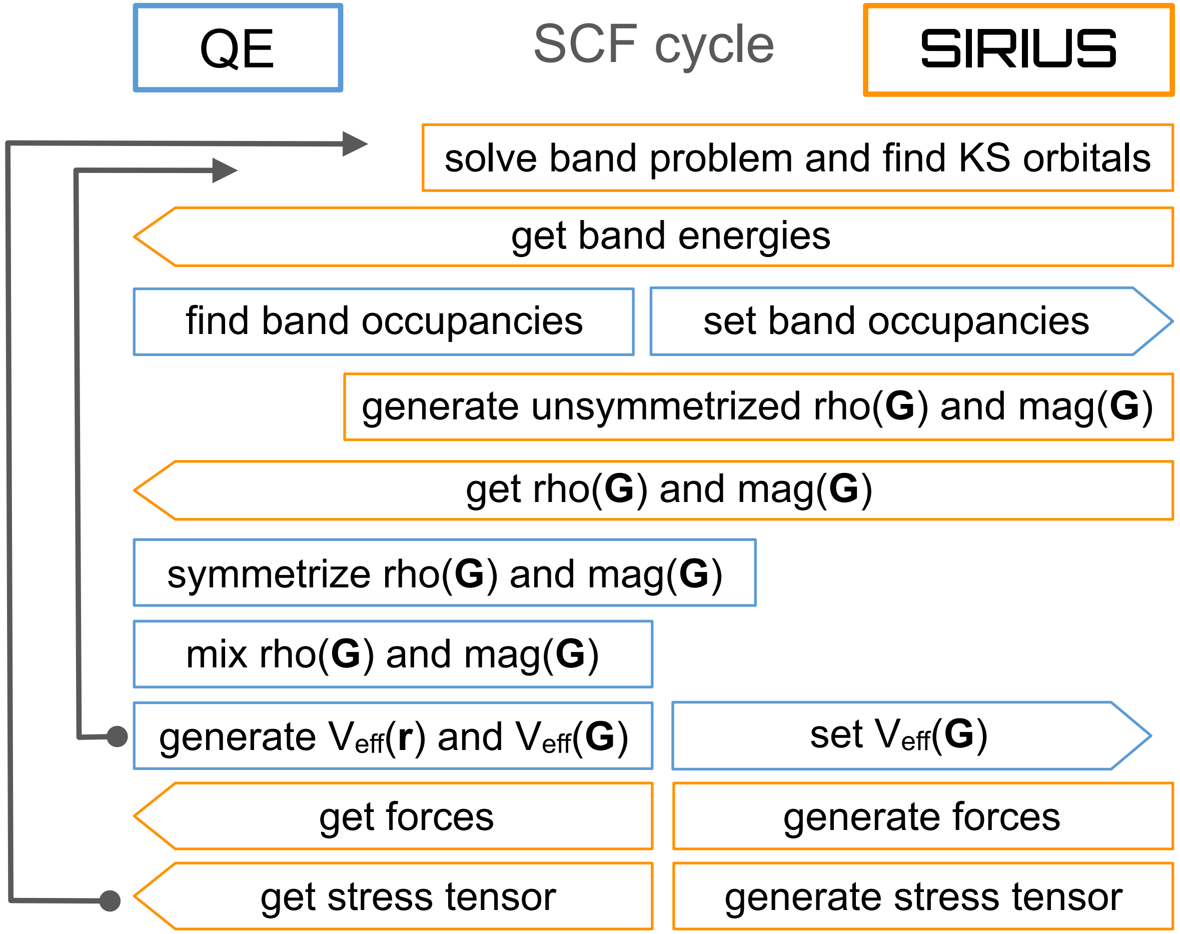
initialize Density class

initialize Potential class

initialize DFT_ground_state class

generate initial density

get rho($\mathbf{G}$) and mag($\mathbf{G}$)



QE

SCF cycle

SIRIUS

solve band problem and find KS orbitals

get band energies

find band occupancies

set band occupancies

generate unsymmetrized rho($\mathbf{G}$) and mag($\mathbf{G}$)

get rho($\mathbf{G}$) and mag($\mathbf{G}$)

symmetrize rho($\mathbf{G}$) and mag($\mathbf{G}$)

mix rho($\mathbf{G}$) and mag($\mathbf{G}$)

generate $V_{\text{eff}}(\mathbf{r})$ and $V_{\text{eff}}(\mathbf{G})$

set $V_{\text{eff}}(\mathbf{G})$

get forces

generate forces

get stress tensor

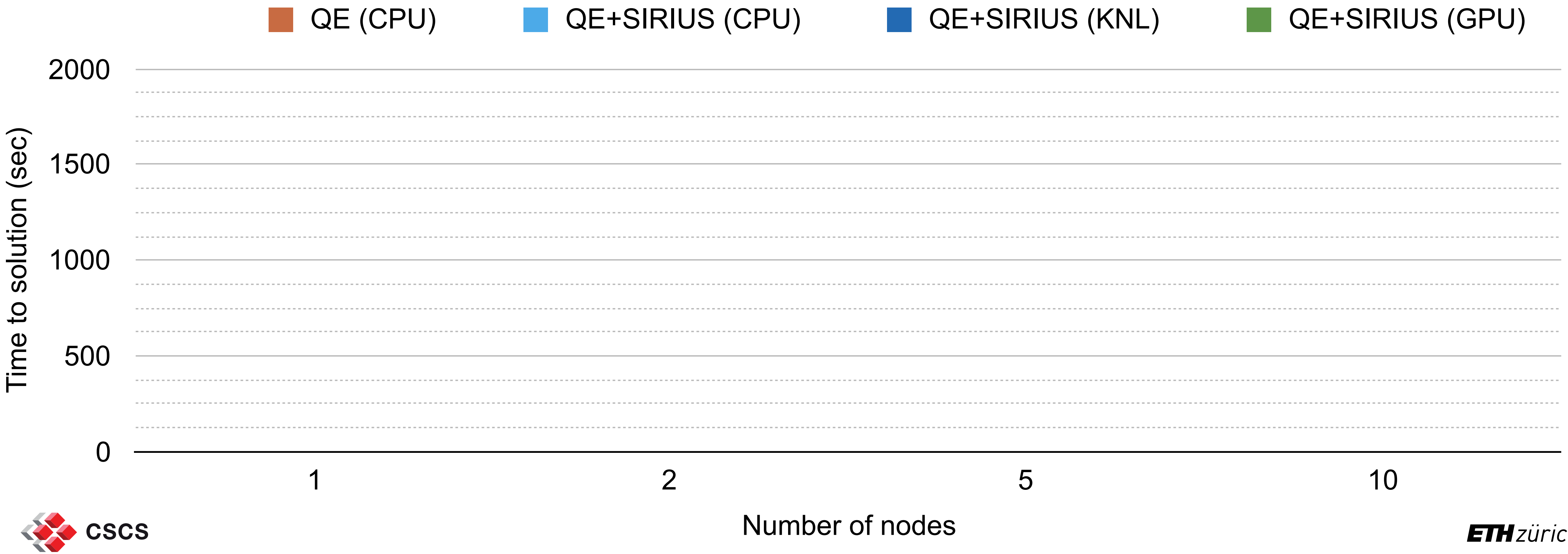
generate stress tensor

QE: variable cell relaxation of $Si_{63}Ge$

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 64-atom unit cell of $Si_{1-x}Ge_x$ The runs we performed on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) , on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). Time for the full ‘vc-relax’ calculation is reported.

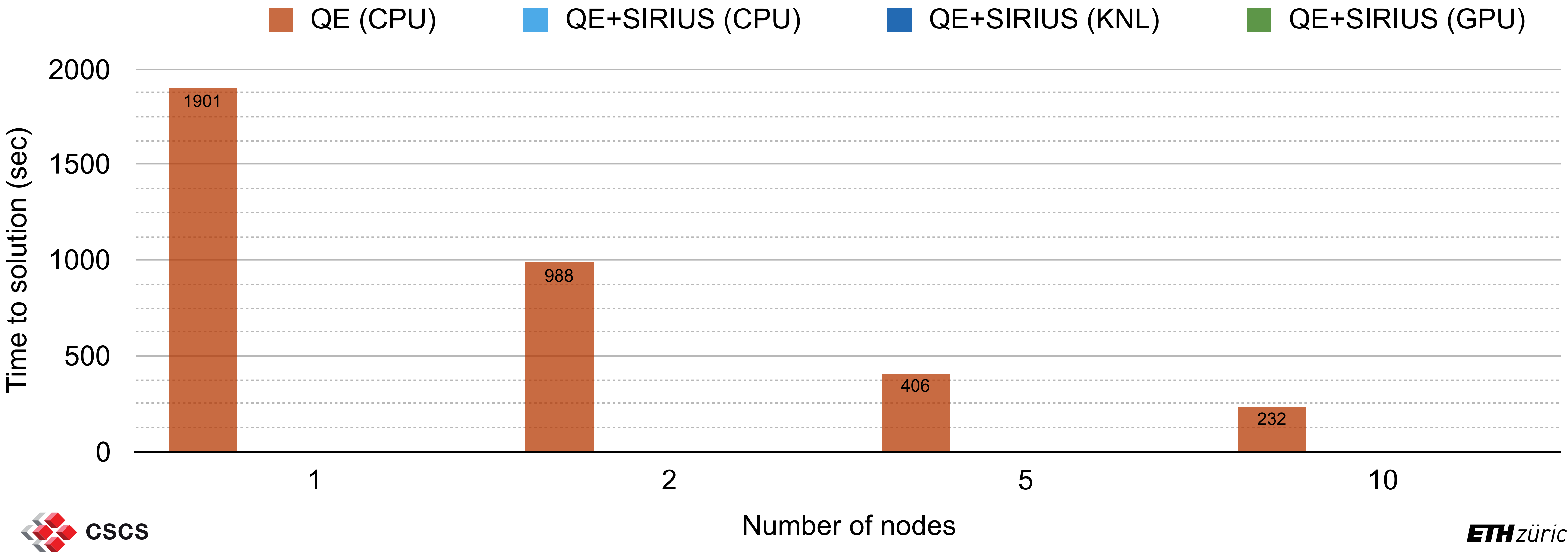
QE: variable cell relaxation of $Si_{63}Ge$

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 64-atom unit cell of $Si_{1-x}Ge_x$ The runs we performed on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) , on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). Time for the full ‘vc-relax’ calculation is reported.



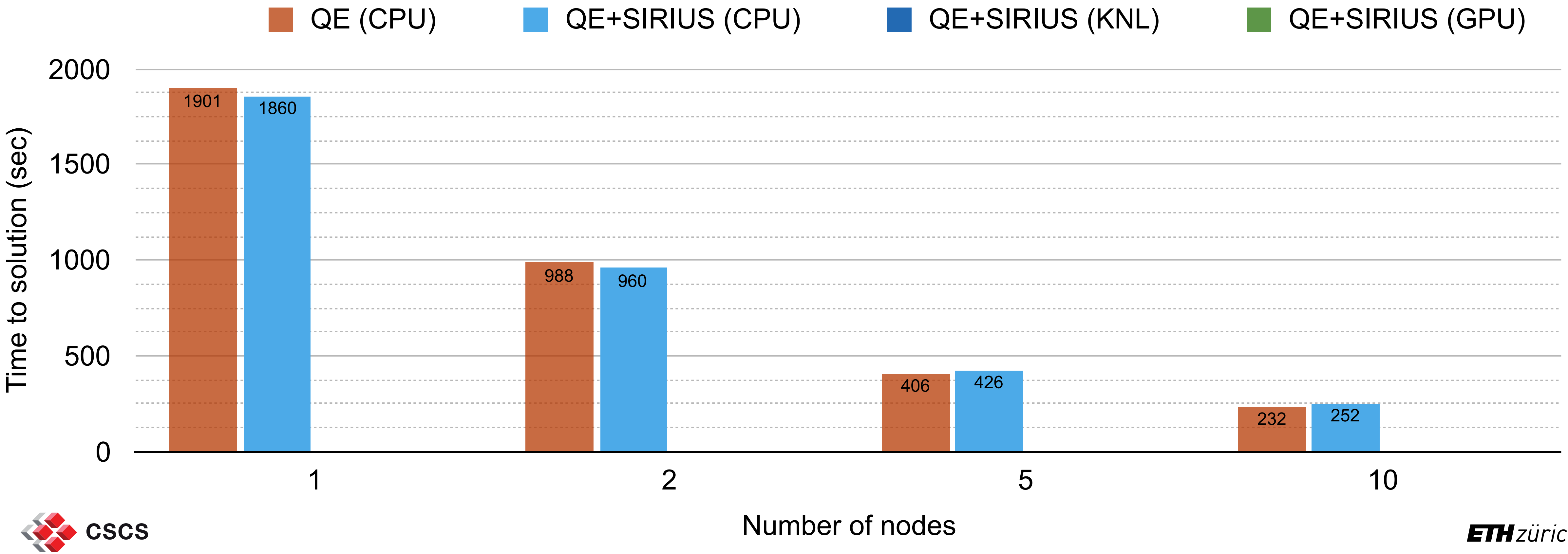
QE: variable cell relaxation of $Si_{63}Ge$

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 64-atom unit cell of $Si_{1-x}Ge_x$ The runs we performed on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) , on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). Time for the full ‘vc-relax’ calculation is reported.



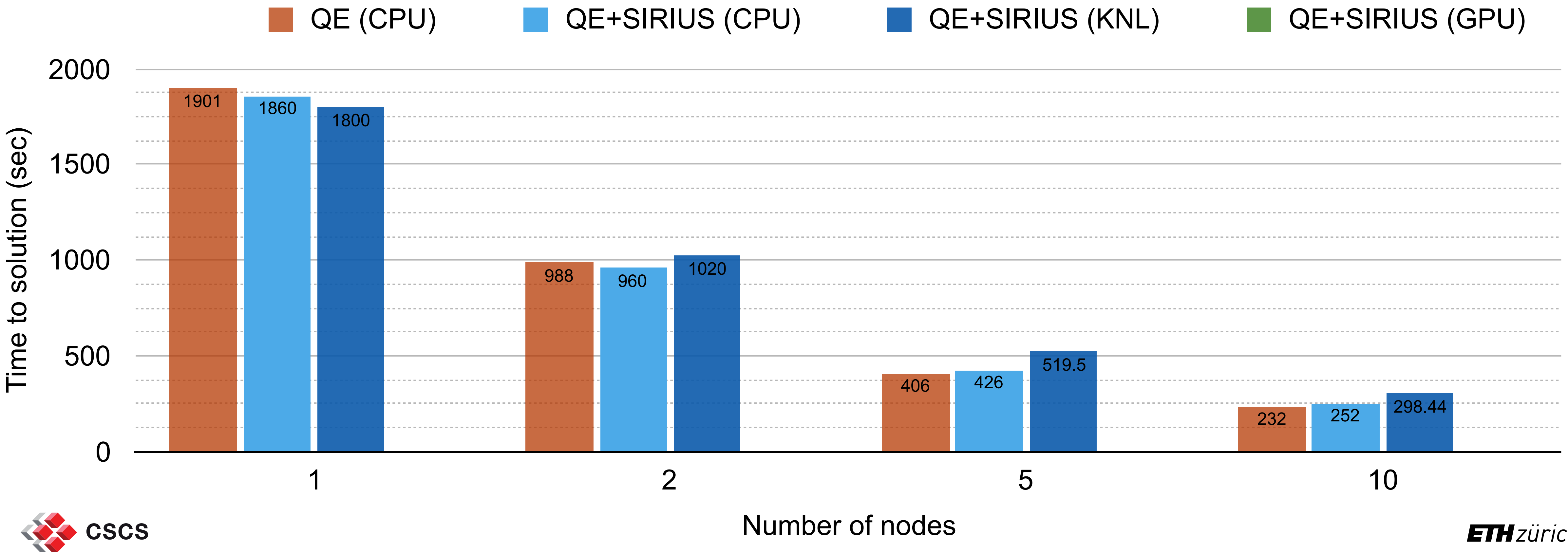
QE: variable cell relaxation of $Si_{63}Ge$

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 64-atom unit cell of $Si_{1-x}Ge_x$ The runs we performed on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) , on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). Time for the full 'vc-relax' calculation is reported.



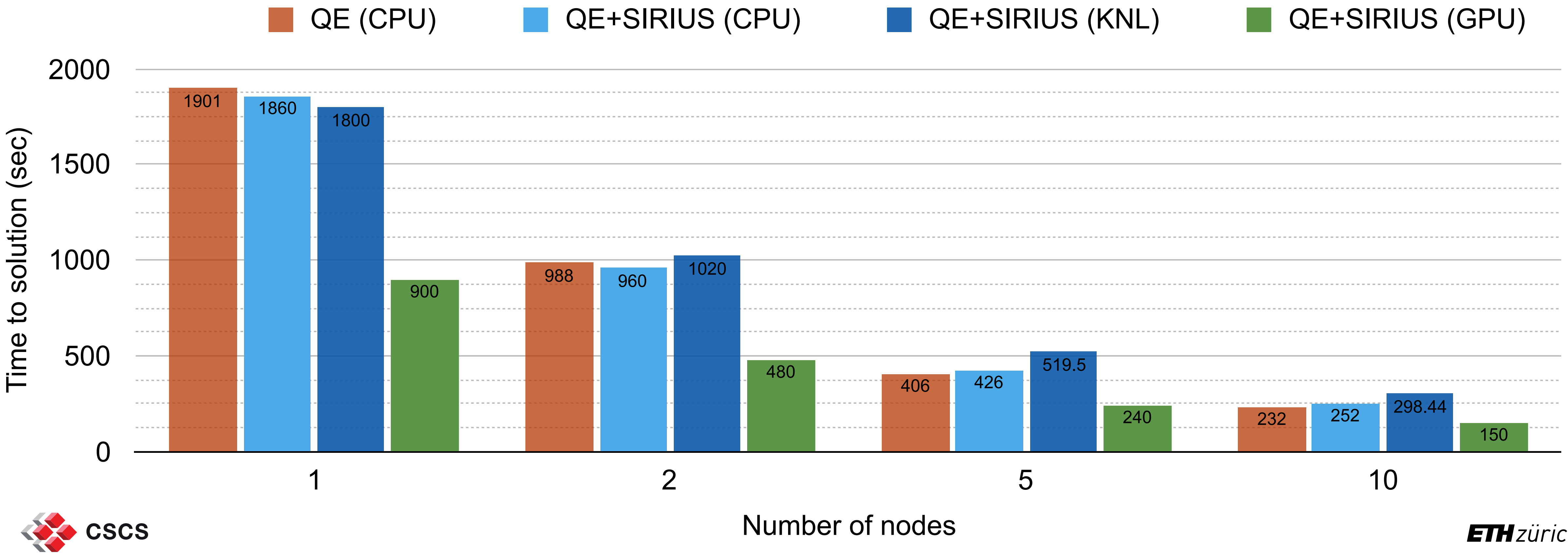
QE: variable cell relaxation of $Si_{63}Ge$

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 64-atom unit cell of $Si_{1-x}Ge_x$ The runs we performed on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) , on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). Time for the full 'vc-relax' calculation is reported.



QE: variable cell relaxation of $Si_{63}Ge$

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 64-atom unit cell of $Si_{1-x}Ge_x$ The runs we performed on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) , on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). Time for the full 'vc-relax' calculation is reported.



cscs

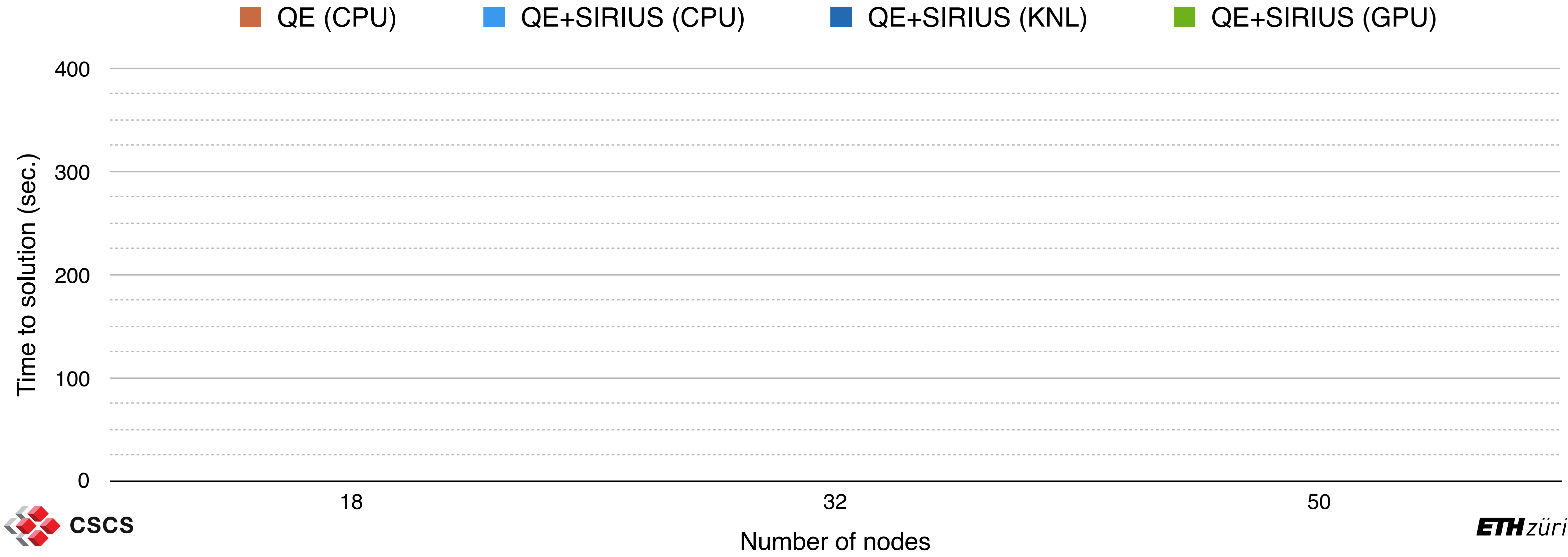
ETH zürich

QE: ground state of Pt-cluster in water

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 288-atom unit cell of Pt cluster embedded in the water. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (BW), on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). ELPA eigen-value solver was used for CPU runs. Time for the SCF ground state calculation is reported.

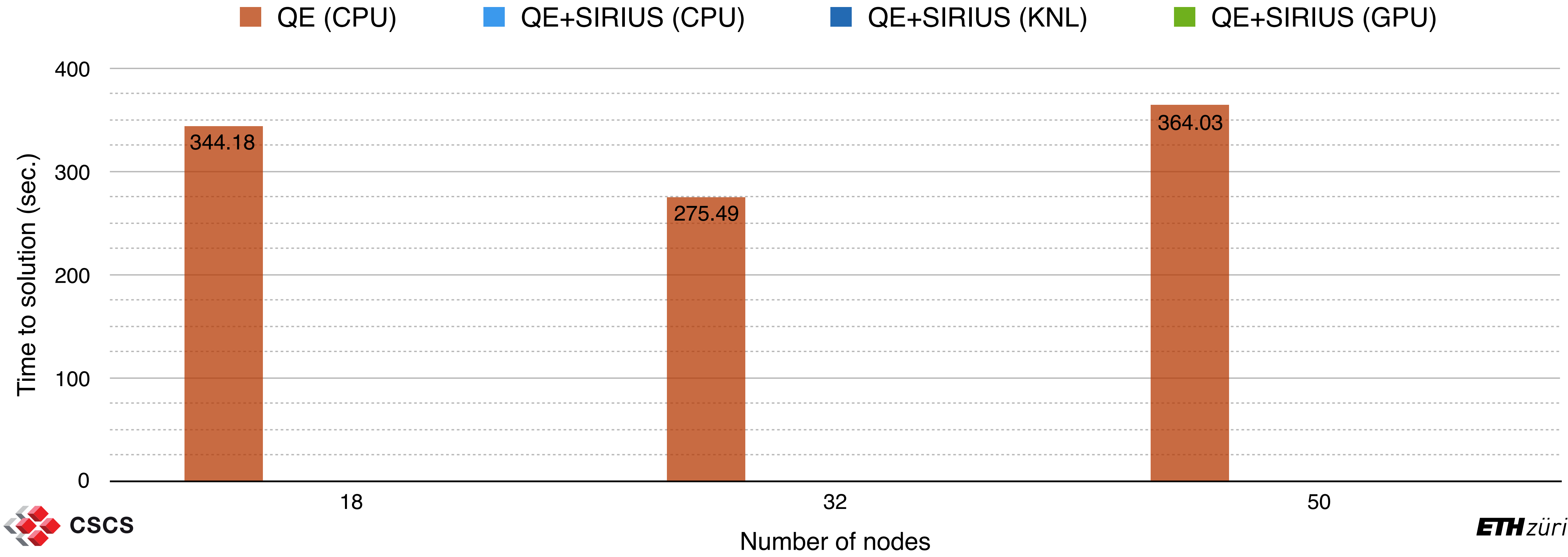
QE: ground state of Pt-cluster in water

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 288-atom unit cell of Pt cluster embedded in the water. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (BW), on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). ELPA eigen-value solver was used for CPU runs. Time for the SCF ground state calculation is reported.



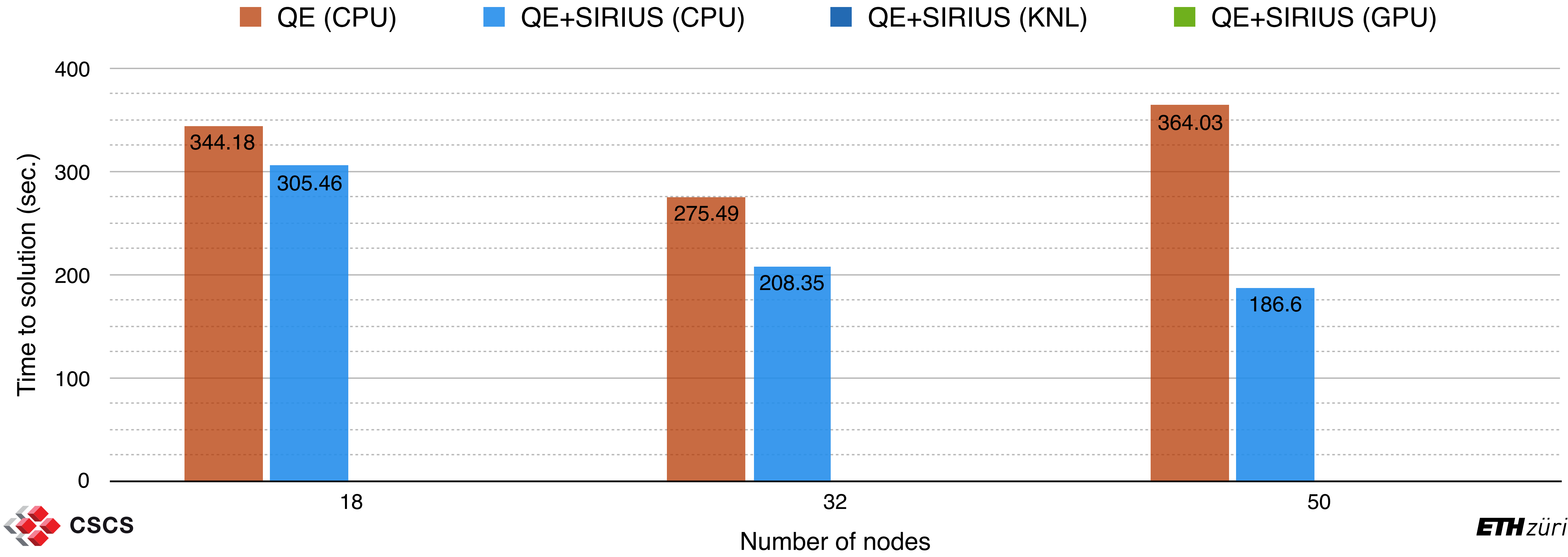
QE: ground state of Pt-cluster in water

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 288-atom unit cell of Pt cluster embedded in the water. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (BW), on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). ELPA eigen-value solver was used for CPU runs. Time for the SCF ground state calculation is reported.



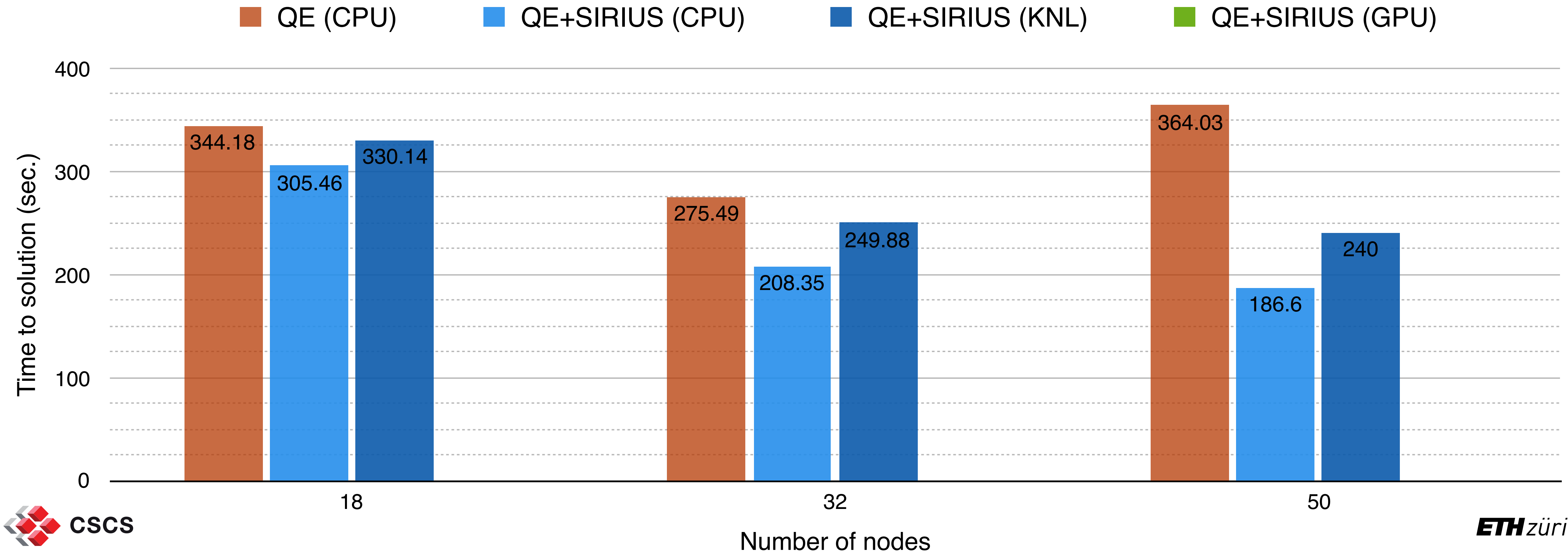
QE: ground state of Pt-cluster in water

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 288-atom unit cell of Pt cluster embedded in the water. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (BW), on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). ELPA eigen-value solver was used for CPU runs. Time for the SCF ground state calculation is reported.



QE: ground state of Pt-cluster in water

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 288-atom unit cell of Pt cluster embedded in the water. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (BW), on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). ELPA eigen-value solver was used for CPU runs. Time for the SCF ground state calculation is reported.

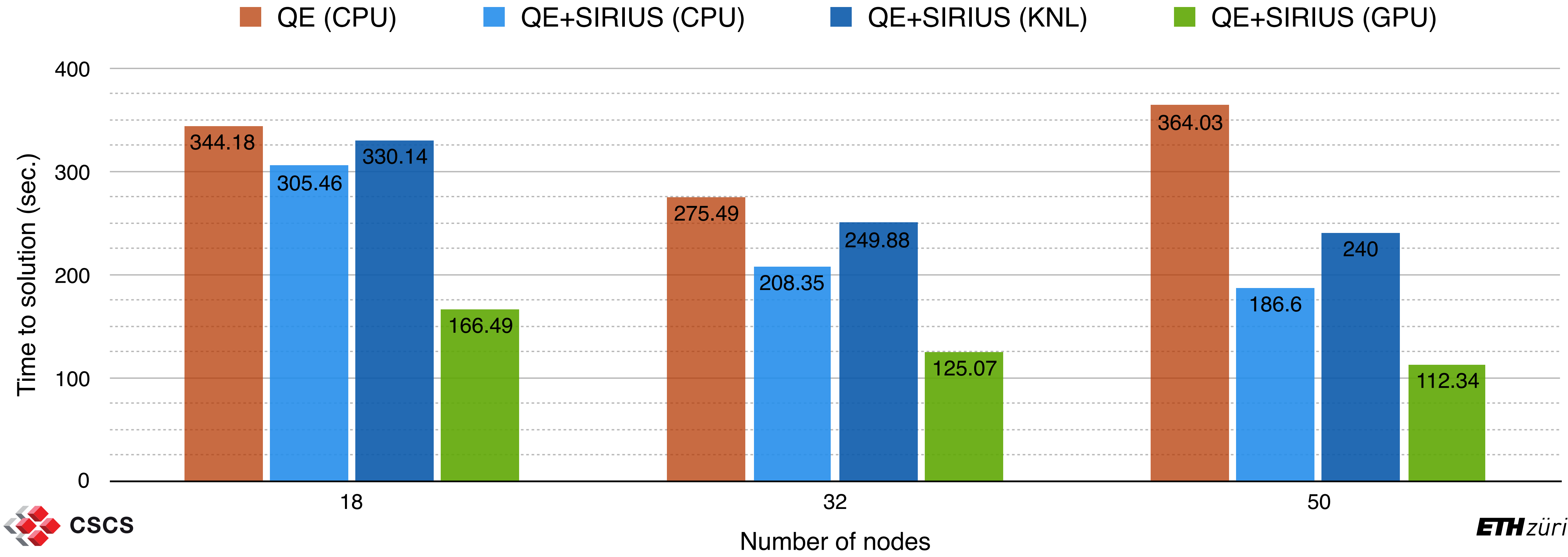


CSCS

ETH zürich

QE: ground state of Pt-cluster in water

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 288-atom unit cell of Pt cluster embedded in the water. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (BW), on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). ELPA eigen-value solver was used for CPU runs. Time for the SCF ground state calculation is reported.

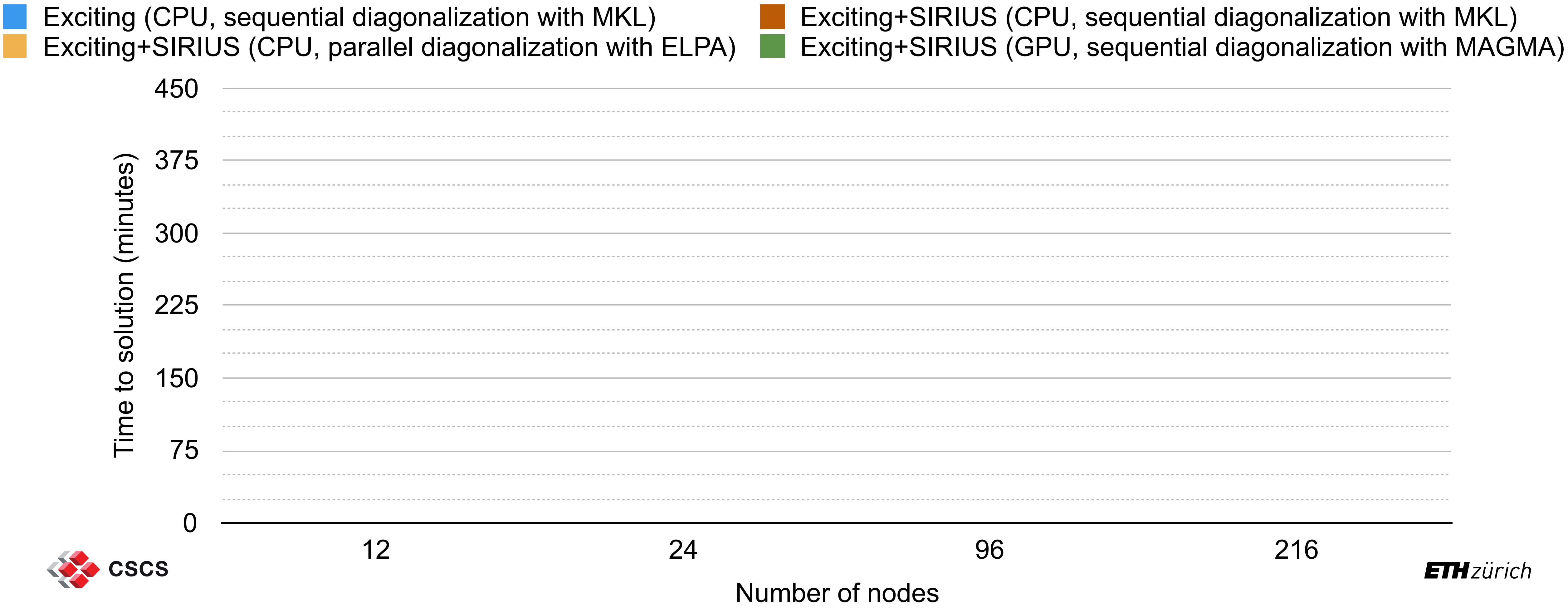


Exciting: ground state of Mn-based MoF ($C_5H_{11}MnNO_6$)

Performance benchmark of the QE and SIRIUS-enabled Exciting codes for the 96-atom unit cell of Mn metal-organic framework. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU).

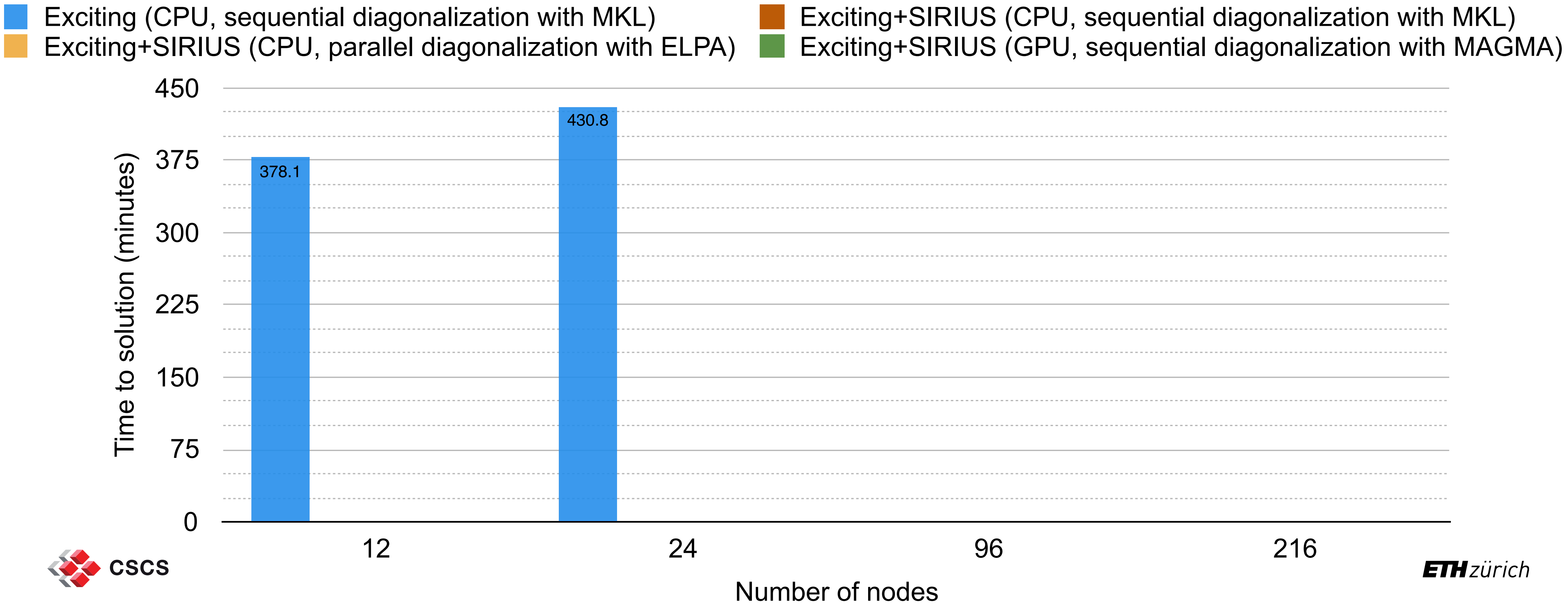
Exciting: ground state of Mn-based MoF ($C_5H_{11}MnNO_6$)

Performance benchmark of the QE and SIRIUS-enabled Exciting codes for the 96-atom unit cell of Mn metal-organic framework. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU).



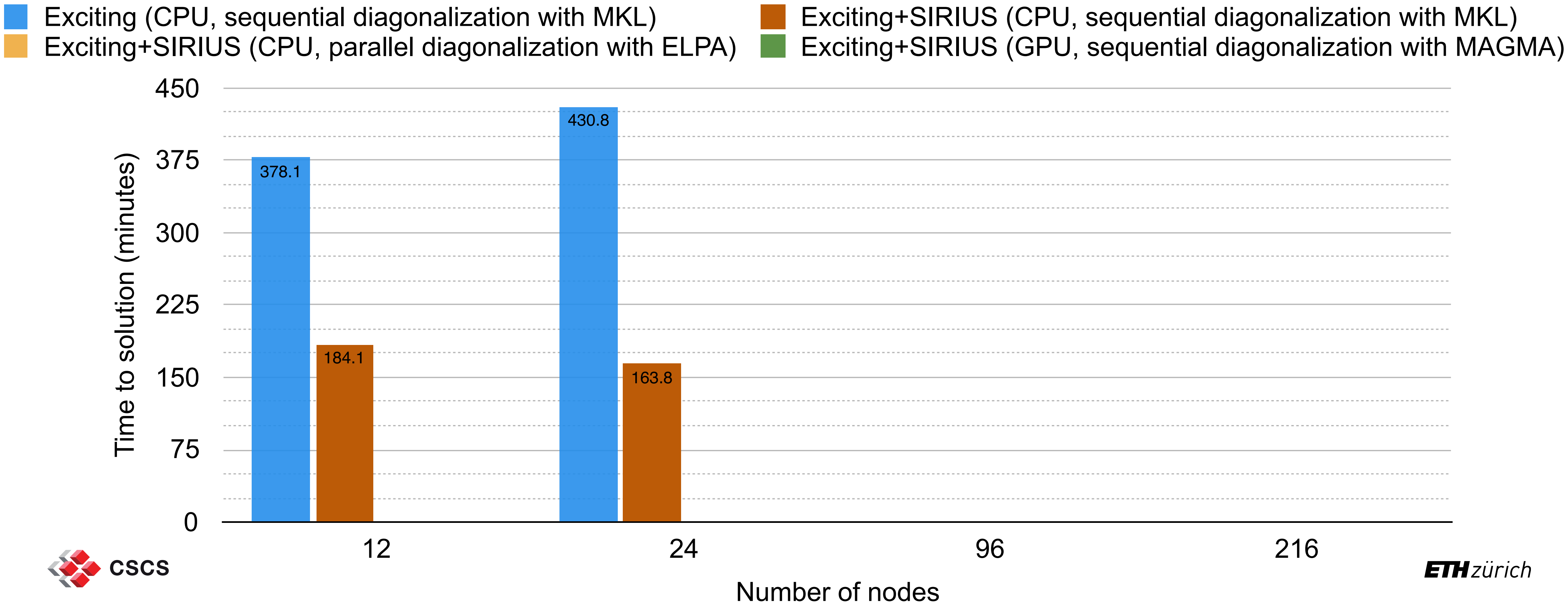
Exciting: ground state of Mn-based MoF ($C_5H_{11}MnNO_6$)

Performance benchmark of the QE and SIRIUS-enabled Exciting codes for the 96-atom unit cell of Mn metal-organic framework. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU).



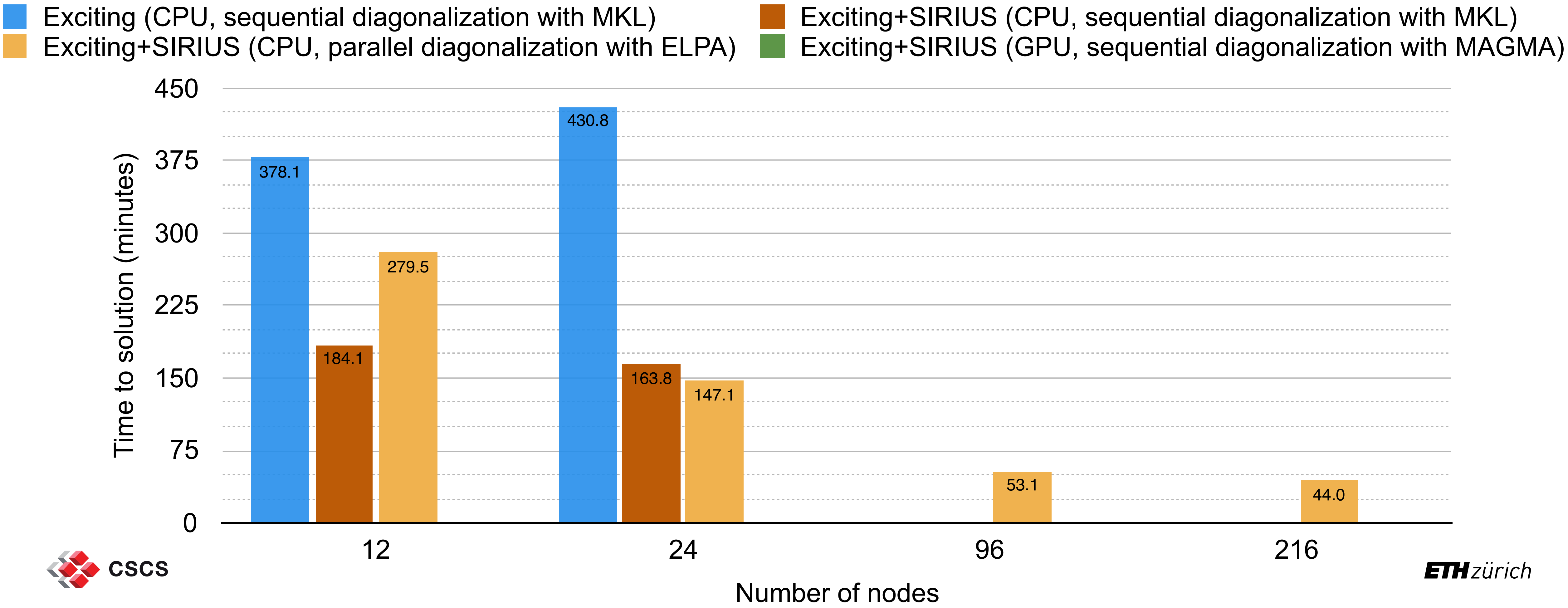
Exciting: ground state of Mn-based MoF ($C_5H_{11}MnNO_6$)

Performance benchmark of the QE and SIRIUS-enabled Exciting codes for the 96-atom unit cell of Mn metal-organic framework. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU).



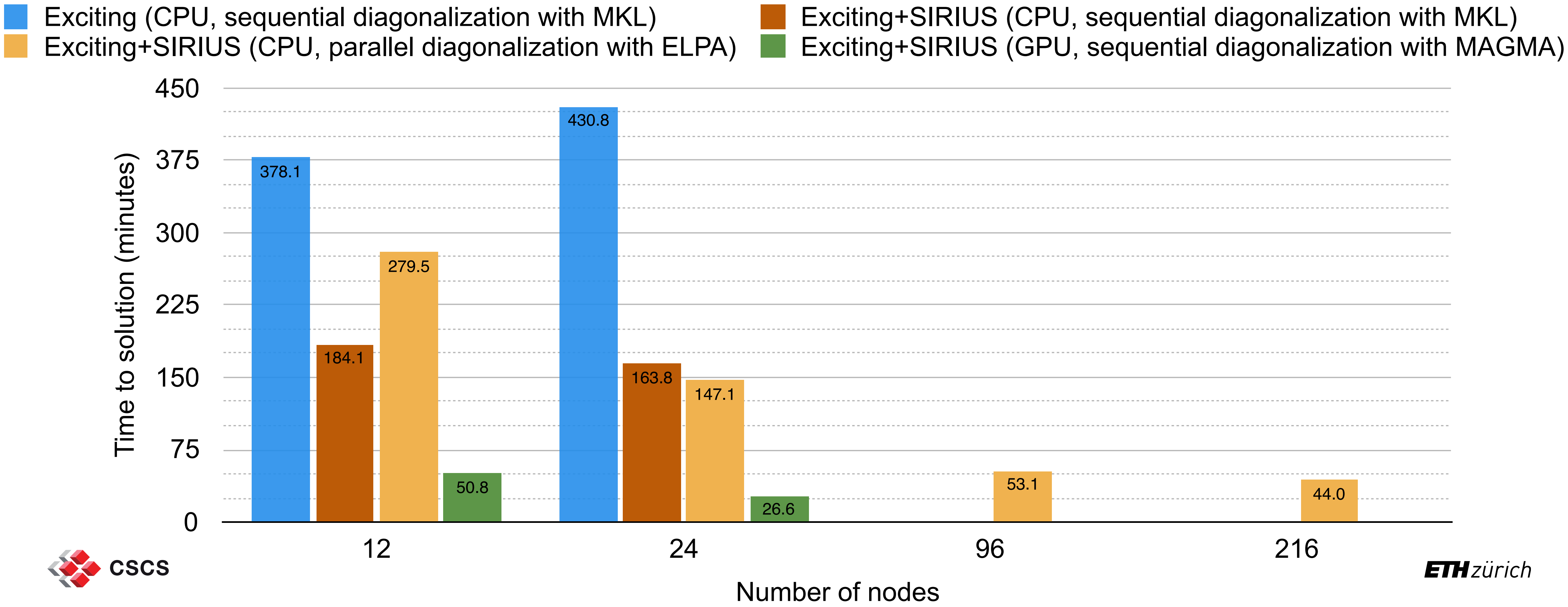
Exciting: ground state of Mn-based MoF ($C_5H_{11}MnNO_6$)

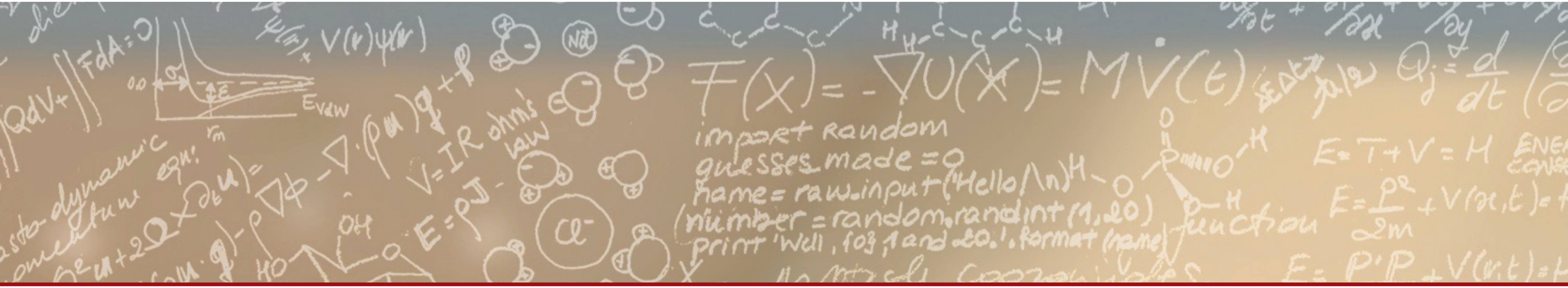
Performance benchmark of the QE and SIRIUS-enabled Exciting codes for the 96-atom unit cell of Mn metal-organic framework. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU).



Exciting: ground state of Mn-based MoF ($C_5H_{11}MnNO_6$)

Performance benchmark of the QE and SIRIUS-enabled Exciting codes for the 96-atom unit cell of Mn metal-organic framework. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (CPU) and on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU).





Thank you for your attention.