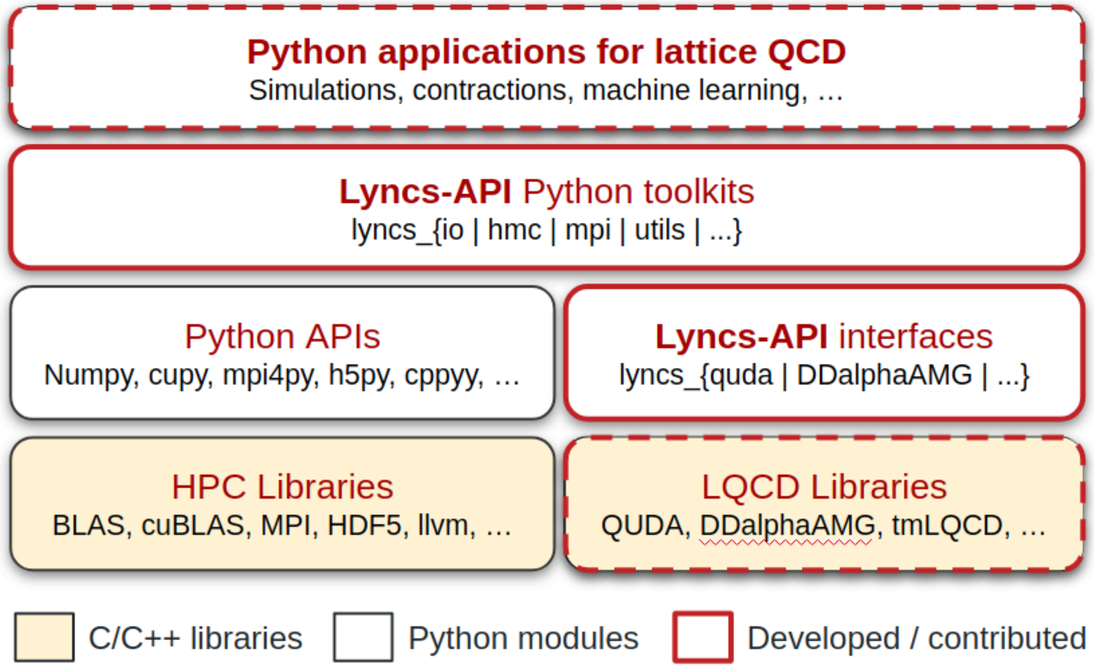
Running HMC Simulations with Python via QUDA

SHUHEI YAMAMOTO
SIMONE BACCHIO, JACOB FINKENRATH
AUG. 10TH, 2022

Funded by: PRACE-6IP, WP8 "Forward Looking Software Solutions". Grant agreement ID: 823767, Project name: LyNcs

Lyncs API

- is a Python toolkit that allows the user to use and run various lattice QCD libraries while programming in Python
- Goals: Performance, Portability, and Productivity
- uses Python APIs such as numpy, cupy, and mpi4py
- contains interfaces to lattice QCD libraries



S. Bacchio, 08/08/2022 14:40, Software session

Lyncs-QUDA

- provides an intermediate layer of Python interface to QUDA
- allows the user to program in Python while using various QUDA objects and functionalities
- wraps QUDA objects by the corresponding Python classes
- available at https://github.com/Lyncs-API/lyncs.quda
- to be installed via pip from the above source

Lyncs-QUDA

- comes with test and example scripts

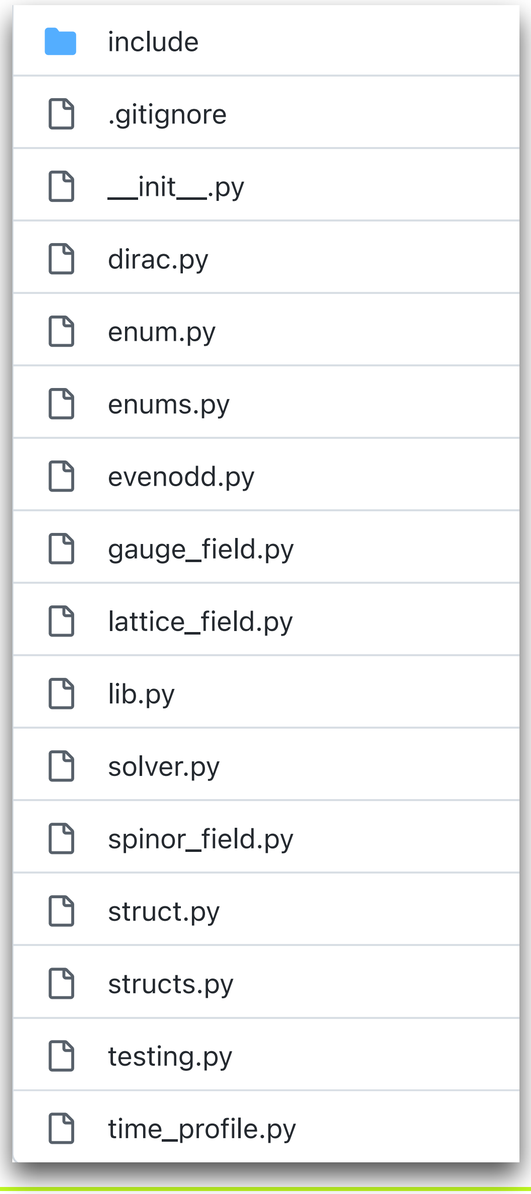
Lyncs-API / **lyncs.quda**

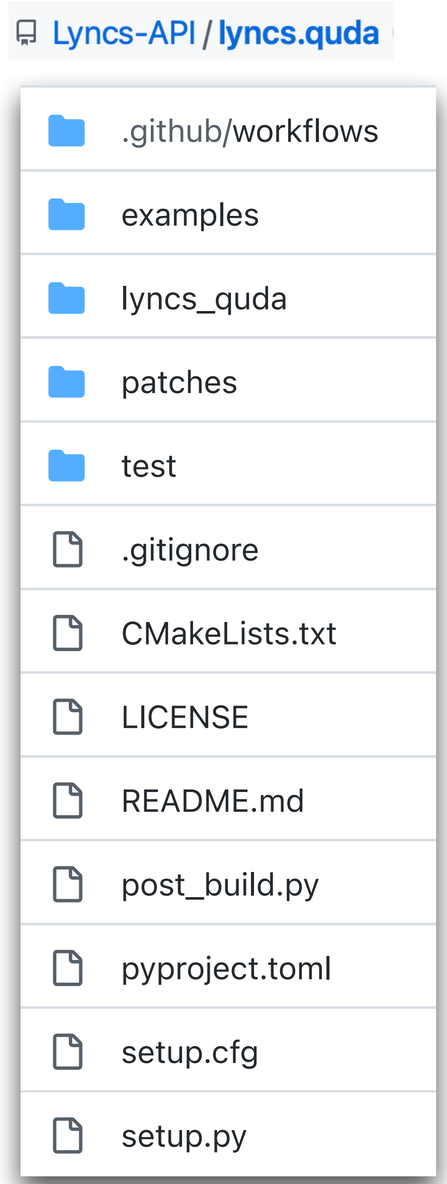
- .github/workflows
- examples
- lyncs_quda
- patches
- test
- .gitignore
- CMakeLists.txt
- LICENSE
- README.md
- post_build.py
- pyproject.toml
- setup.cfg
- setup.py

- parallel_test
- test_clover_field.py
- test_dirac.py
- test_enums.py
- test_evenodd.py
- test_field.py
- test_gauge_field.py
- test_lib.py
- test_solver.py
- test_spinor.py
- test_structs.py
- test_testing.py

Lyncs-QUDA

- Wrapper scripts in lyncs_quda/





Lyncs-QUDA

- Wrapper scripts in lyncs_quda/
- lib.py
  - load headers and library

```python
headers = [
    "quda.h",
    "gauge_field.h",
    "gauge_tools.h",
    "gauge_force_quda.h",
    "gauge_update_quda.h",
    "dirac_quda.h",
    "invert_quda.h",
    "blas_quda.h",
    "multigrid.h",
    "evenodd.h",
]


lib = QudaLib(
    path=PATHS,
    header=headers,
    library=["libquda.so"] + libs,
    namespace=["quda", "lyncs_quda"],
)
```

- include
- .gitignore
- __init__.py
- dirac.py
- enum.py
- enums.py
- evenodd.py
- gauge_field.py
- lattice_field.py
- lib.py
- solver.py
- spinor_field.py
- struct.py
- structs.py
- testing.py
- time_profile.py

Lyncs-API / **lyncs.quda**

- .github/workflows
- examples
- lyncs_quda
- patches
- test
- .gitignore
- CMakeLists.txt
- LICENSE
- README.md
- post_build.py
- pyproject.toml
- setup.cfg
- setup.py

Lyncs-QUDA

- Wrapper scripts in lyncs_quda/
- lib.py
  - load headers and library
- Other scripts
  - call QUDA functions through lib

```python
def gaussian(self, epsilon=1, seed=None):

    seed = seed or int(time() * 1e9)
    lib.gaugeGauss(self.quda_field, seed, epsilon)
```

- include
- .gitignore
- __init__.py
- dirac.py
- enum.py
- enums.py
- evenodd.py
- gauge_field.py
- lattice_field.py
- lib.py
- solver.py
- spinor_field.py
- struct.py
- structs.py
- testing.py
- time_profile.py

Lyncs-API / **lyncs.quda**

- .github/workflows
- examples
- lyncs_quda
- patches
- test
- .gitignore
- CMakeLists.txt
- LICENSE
- README.md
- post_build.py
- pyproject.toml
- setup.cfg
- setup.py

Lyncs-QUDA

- gauge_field.py
  - **auxiliary functions**

```python
def gauge_field(lattice, dofs=(4, 18), **kwargs):
    "Constructs a new gauge field"
    # TODO add option to select field type -> dofs
    # TODO reshape/shuffle to native order
    return GaugeField.create(lattice, dofs=dofs, **kwargs)


def gauge_scalar(lattice, dofs=18, **kwargs):
    "Constructs a new scalar gauge field"
    return gauge_field(lattice, dofs=(1, dofs), **kwargs)


def gauge_links(lattice, dofs=18, **kwargs):
    "Constructs a new gauge field of links"
    return gauge_field(lattice, dofs=(4, dofs), **kwargs)

def gauge_tensor(lattice, dofs=18, **kwargs):
    "Constructs a new gauge field with tensor structure"
    return gauge_field(lattice, dofs=(6, dofs), **kwargs)


def gauge_coarse(lattice, dofs=2 * 48**2, **kwargs):
    "Constructs a new coarse gauge field"
    return gauge_field(lattice, dofs=(8, dofs), **kwargs)


def momentum(lattice, **kwargs):
    return gauge_field(lattice, dofs=(4, 10), **kwargs)
```

Lyncs-QUDA

- gauge_field.py
  - auxiliary functions
  - **class definition**
    - **data field**

```
class GaugeField(LatticeField):
    "Mimics the quda::LatticeField object"

    @LatticeField.field.setter
    def field(self, field):
        LatticeField.field.fset(self, field)
        if self.reconstruct == "INVALID":
            raise TypeError(f"Unrecognized field dofs {self.dofs}")
```

Lyncs-QUDA

- gauge_field.py
  - auxiliary functions
  - class definition
    - data field
    - **meta data**

```python
class GaugeField(LatticeField):
    "Mimics the quda::LatticeField object"

    @LatticeField.field.setter
    def field(self, field):
        LatticeField.field.fset(self, field)
        if self.reconstruct == "INVALID":
            raise TypeError(f"Unrecognized field dofs {self.dofs}")

                                ⋮

    @property
    def quda_reconstruct(self):
        "Quda enum for reconstruct type of the field"
        return getattr(lib, f"QUDA_RECONSTRUCT_{self.reconstruct}")

    @property
    def ncol(self):
        "Number of colors"
        if self.reconstruct == "NO":
            dofs = self.dofs_per_link
            ncol = sqrt(dofs / 2)
            assert ncol.is_integer()
            return int(ncol)
        return 3
```

Lyncs-QUDA

- gauge_field.py
  - auxiliary functions
  - class definition
    - data field
    - meta data
    - **QUDA object**

```python
@property
def quda_params(self):
    "Returns an instance of quda::GaugeFieldParams"
    params = lib.GaugeFieldParam()
    lib.copy_struct(params, super().quda_params)
    params.reconstruct = self.quda_reconstruct
    params.geometry = self.quda_geometry
    params.link_type = self.quda_link_type
    params.gauge = to_pointer(self.ptr)
    params.create = lib.QUDA_REFERENCE_FIELD_CREATE
    params.location = self.quda_location
    params.t_boundary = self.quda_t_boundary
    params.order = self.quda_order
    params.nColor = self.ncol
    return params

@property
def quda_field(self):
    "Returns an instance of quda::GaugeField"
    self.activate()
    if self._quda is None:
        self._quda = make_shared(lib.GaugeField.Create(self.quda_params))
    return self._quda
```

Lyncs-QUDA

- gauge_field.py
  - auxiliary functions
  - class definition
    - data field
    - meta data
    - QUDA object
    - **methods**

QUDA

```
/**
   @brief Compute the plaquette of the gauge field

   @param[in] U The gauge field upon which to compute the plaquette
   @return double3 variable returning (plaquette, spatial plaquette,
   temporal plaquette) site averages normalized such that each
   plaquette is in the range [0,1]
 */
double3 plaquette(const GaugeField &U);
```

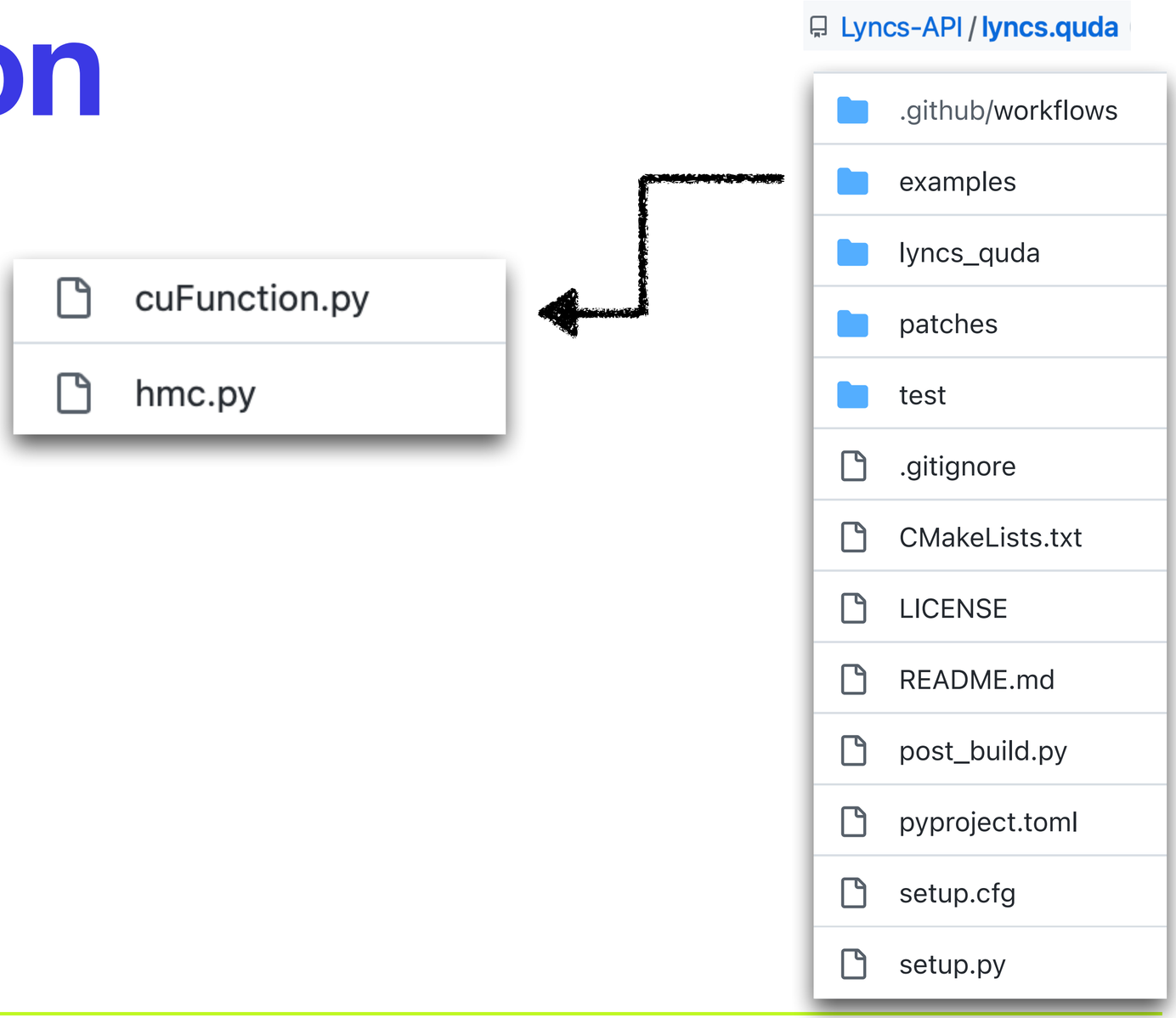
lyncs_quda

```
def plaquette(self):
    """
    Computes the plaquette of the gauge field

    Returns
    -------
    tuple(total, spatial, temporal) plaquette site averaged and
        normalized such that each plaquette is in the range [0,1]
    """
    plaq = lib.plaquette(self.extended_field(1))
    return plaq.x, plaq.y, plaq.z
```

HMC Simulation

- The script can be found in examples/



HMC Simulation

1. Pick an initial gauge configuration
2. Generate its conjugate momenta according to Gaussian distribution
3. Perform MD integration
4. Accept/Reject a proposed configuration according to Metropolis criteria
5. Go back to (2)

HMC Simulation

- HMCHelper class
  - generate an initial gauge
  - generate random momentum
  - update fields, etc…
- Integrator class
- HMC class
  - perform a HMC step
- main()

```python
from lyncs_quda import gauge_field, momentum, lib, MPI

@dataclass
class HMCHelper:
    beta: float = 5
    lattice: tuple = (4, 4, 4, 4)
                ⋮
    def random_gauge(self):
        "Returns a random mom"
        out = gauge_field(self.lattice)
        out.gaussian(10)
        return out
                ⋮
    def update_field(self, field, mom, coeff):
        "Updates the Gauge field"
        return field.update_gauge(mom, coeff)

    def update_mom(self, field, mom, coeff):
        "Updates the momentum"
        return field.compute_paths(
            self.paths, self.coeffs, out=mom, add_coeff=coeff, force=True
        )
```

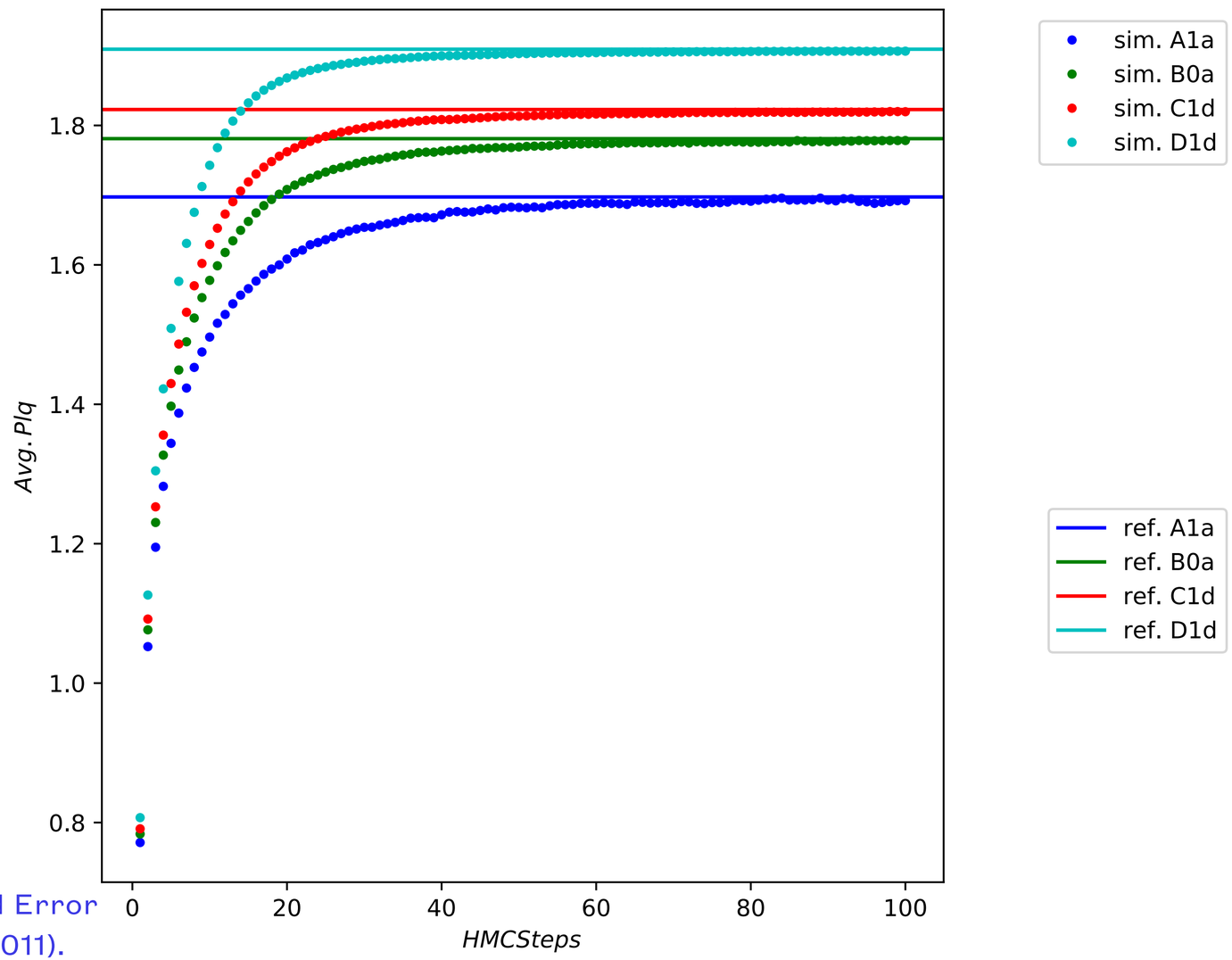
HMC Simulation

- Run the script via
  - `python examples/hmc.py --lattice-size 48  --beta 6.475`
- Current Options

```
Options:
  --beta FLOAT                      target action's beta
  --lattice-size INTEGER            Size of hypercubic lattice
  --lattice-dims INTEGER...         Size of asymmetric lattice
  --procs INTEGER...                Cartesian topology of the communicator
  --integrator [LeapFrog|MN2|MN2p|OMF4]
                                    Integrator for HMC
  --t-steps INTEGER                 Number of time steps trajectory
  --n-trajs INTEGER                 Number of trajectories
  --start [unity|random]            Initial field
```

Results
First sanity check

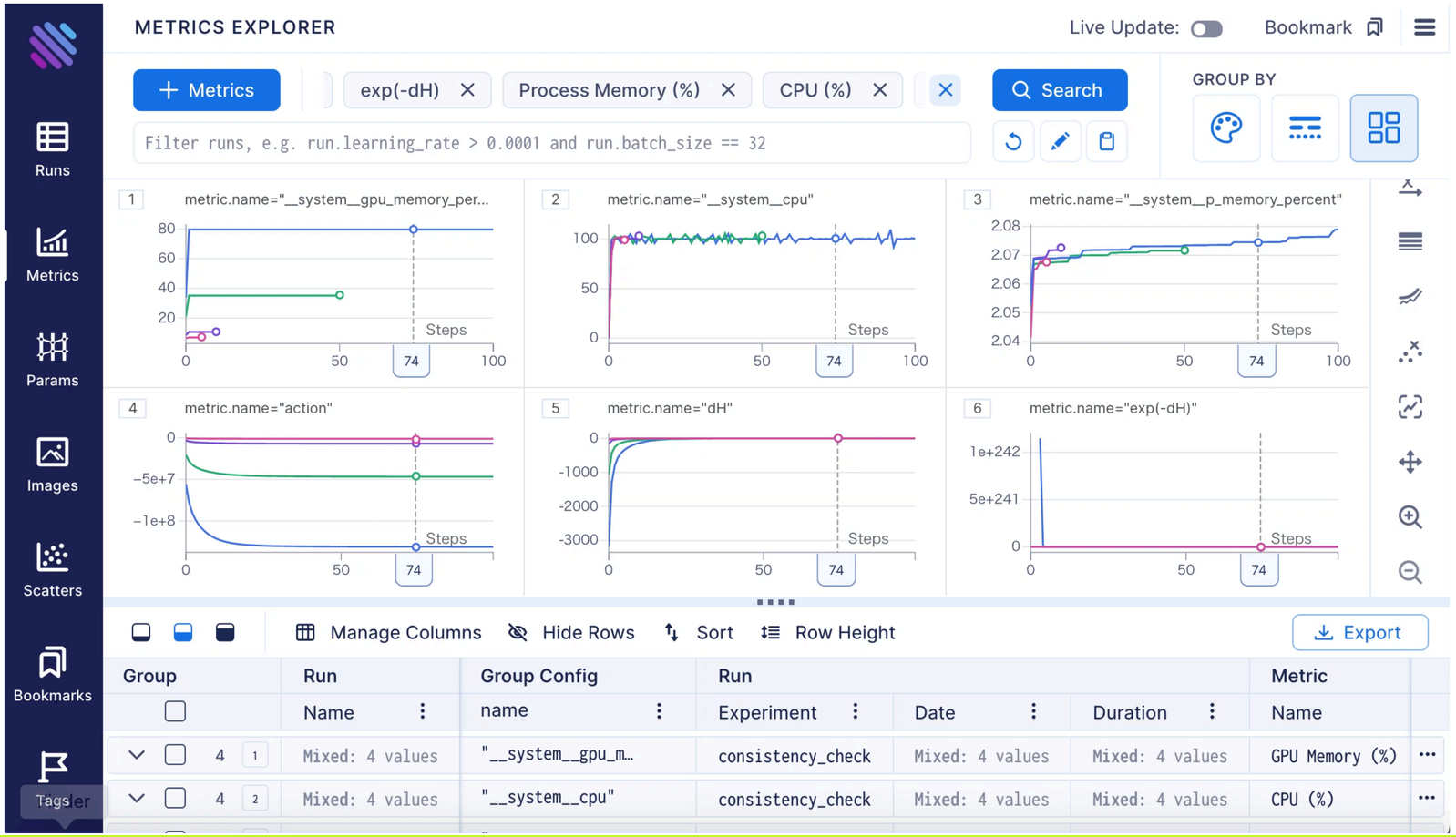
- check if plaquette thermalize to the values from Ref. [1]
- Ensembles (L^3 x T, β)
  - A1a (16^3x16, 5.789):
  - B0a (24^3x24, 6)
  - C1d (32^3x64, 6.136)
  - D1d (48^3x48, 6.475)



[1] S. Schaefer, R. Sommer, and F. Virotta, Critical Slowing down and Error Analysis in Lattice QCD Simulations, Nuclear Physics B 845, 93 (2011).

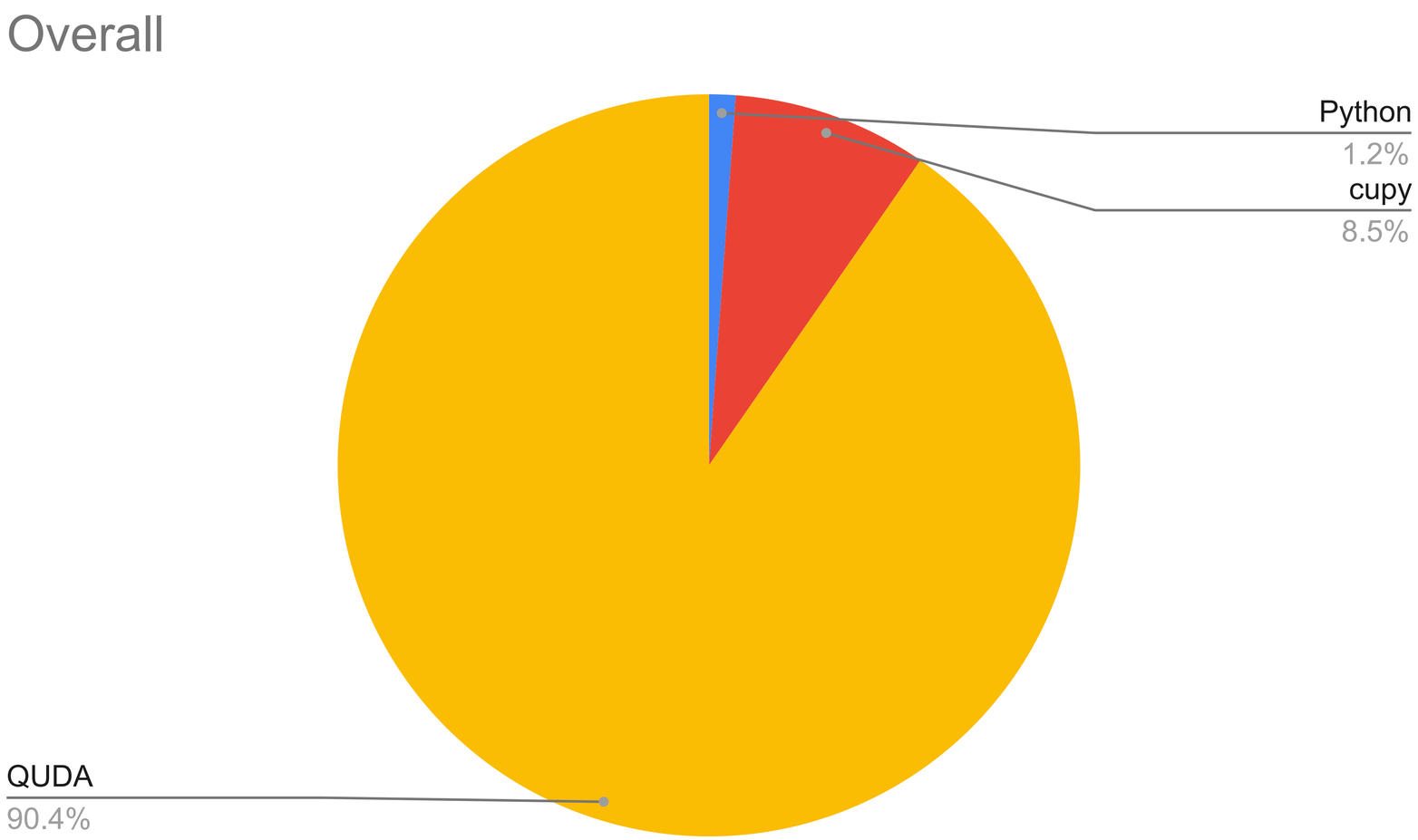
Results

- Other stats tracked by a Python library *aim*



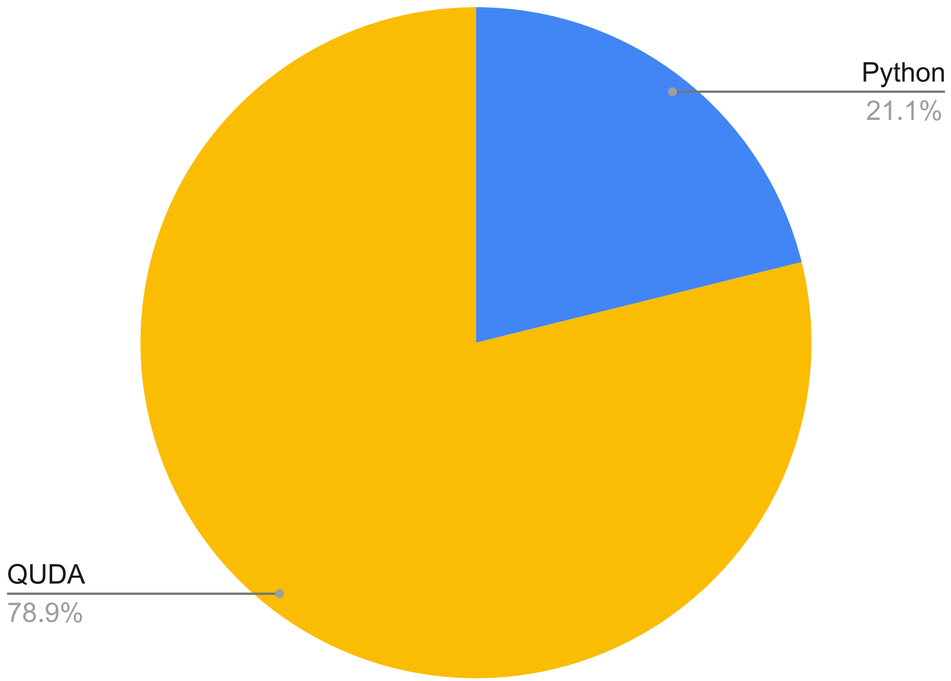
Profiling

- Cyclamen at The Cyprus Institute
  - CPU: Intel® Xeon® Gold 6130
  - GPU: Tesla P100 x 2
- Ensemble:
  - D1d: 48^4
- Profiler:
  - cProfile
  - snakeviz

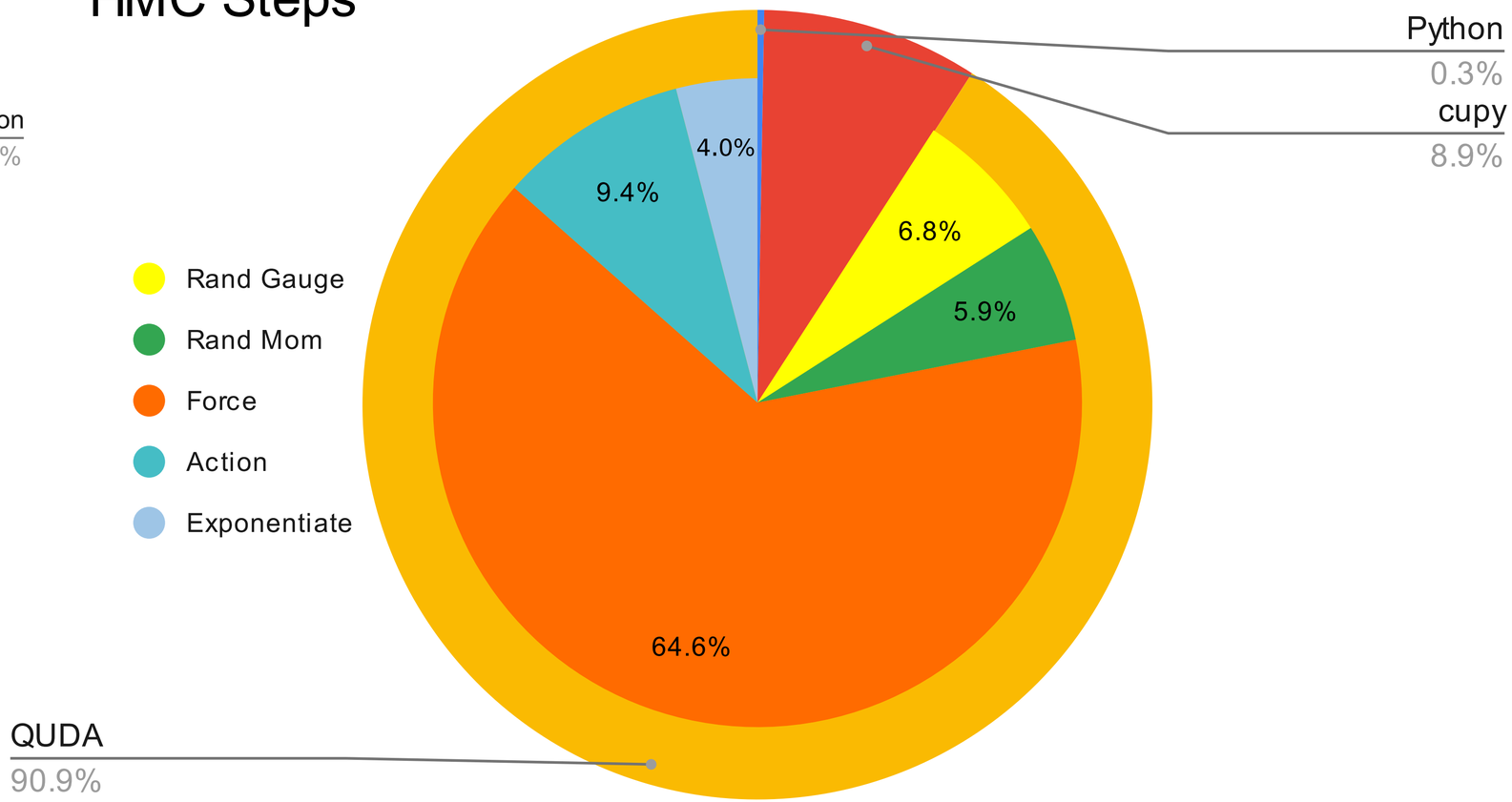


Profiling

Setup



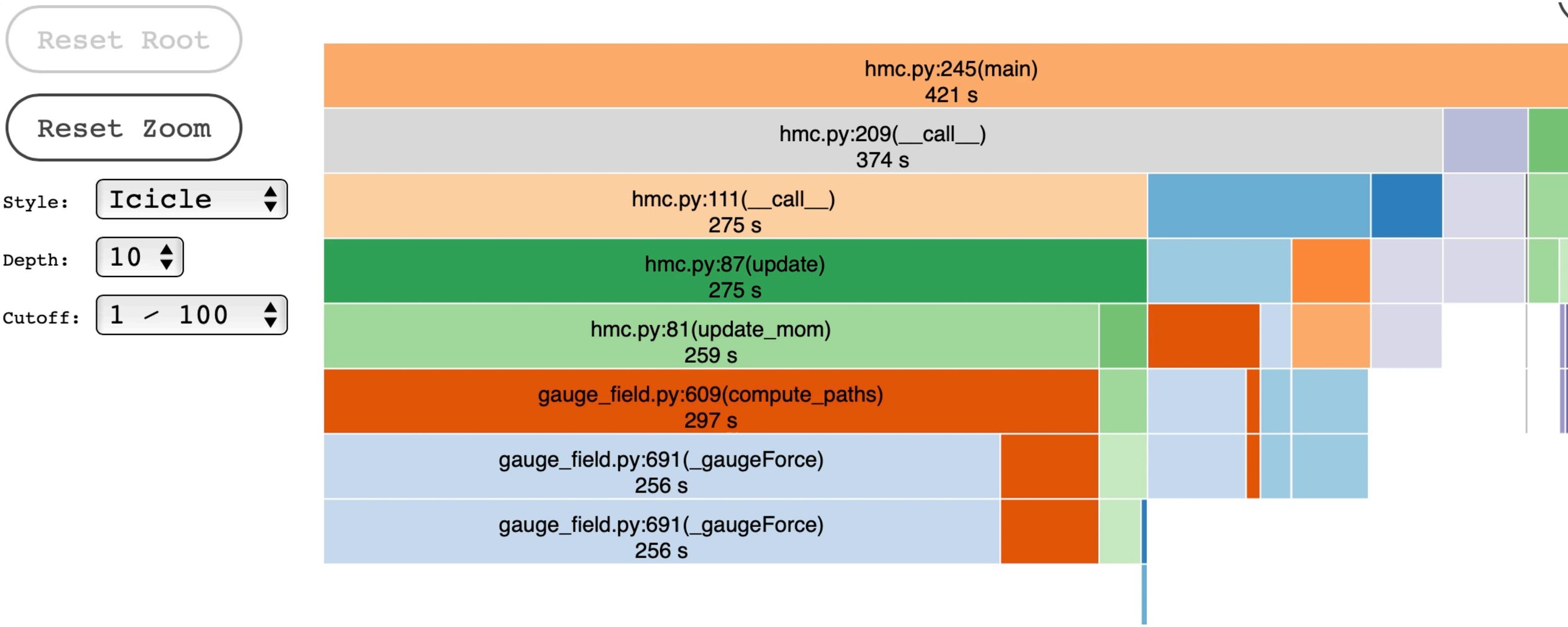
HMC Steps



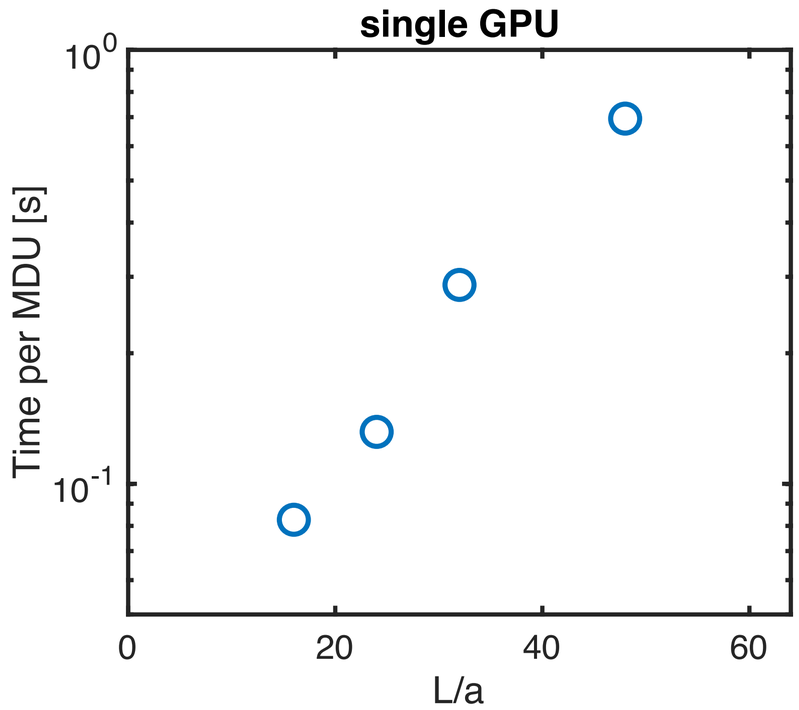
- Setup Time: 17.6s
- 4.1 % of total time (421s)

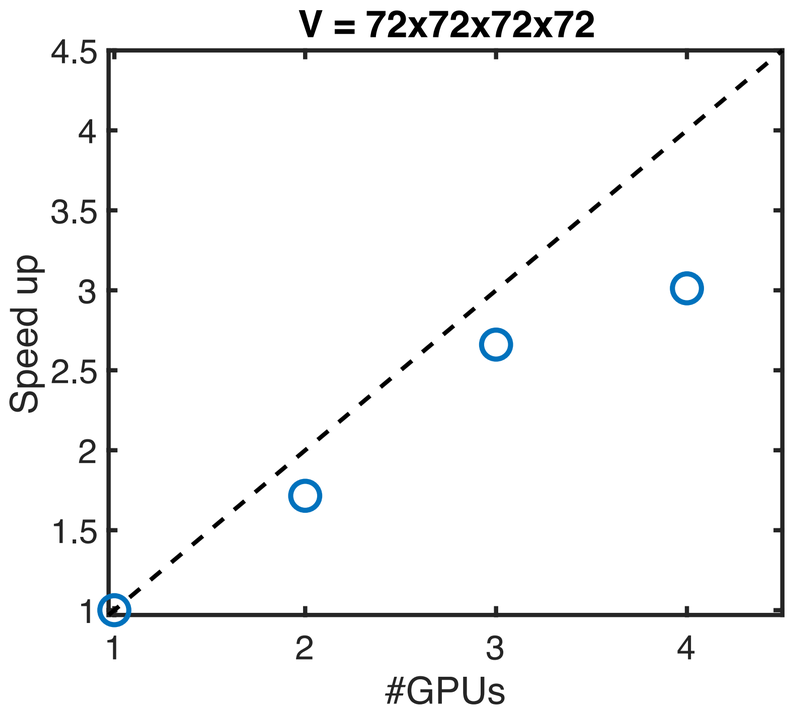
100 HMC Steps: 95.8% of total time

Profiling


Reset Root
Reset Zoom
Style:
Icicle
Depth:
10
Cutoff:
1 – 100
hmc.py:245(main)
421 s
hmc.py:209(__call__)
374 s
hmc.py:111(__call__)
275 s
hmc.py:87(update)
275 s
hmc.py:81(update_mom)
259 s
gauge_field.py:609(compute_paths)
297 s
gauge_field.py:691(_gaugeForce)
256 s
gauge_field.py:691(_gaugeForce)
256 s

Scaling





- JUWELS Booster at Jülich Supercomputing Centre
  - CPU: AMD EPYC 7402
  - GPU: NVIDIA A100 Tensor Core x 4

Outlook

- Support acceleration strategies such as multi grid and deflation
- Support other fermion types such as staggered fermions
- Support various compiler options of QUDA
- Update Lyncs-QUDA to the most recent version of QUDA that now includes support for AMD

Thank you!

Lyncs-QUDA

- include
- .gitignore
- __init__.py
- dirac.py
- enum.py
- enums.py
- evenodd.py
- gauge_field.py
- lattice_field.py
- lib.py
- solver.py
- spinor_field.py
- struct.py
- structs.py
- testing.py
- time_profile.py

- parallel_test
- test_clover_field.py
- test_dirac.py
- test_enums.py
- test_evenodd.py
- test_field.py
- test_gauge_field.py
- test_lib.py
- test_solver.py
- test_spinor.py
- test_structs.py
- test_testing.py

- .github/workflows
- examples
- lyncs_quda
- patches
- test
- .gitignore
- CMakeLists.txt
- LICENSE
- README.md
- post_build.py
- pyproject.toml
- setup.cfg
- setup.py

Lyncs-QUDA

```python
def gauge_field(lattice, dofs=(4, 18), **kwargs):
    "Constructs a new gauge field"
    # TODO add option to select field type -> dofs
    # TODO reshape/shuffle to native order
    return GaugeField.create(lattice, dofs=dofs, **kwargs)


def gauge_scalar(lattice, dofs=18, **kwargs):
    "Constructs a new scalar gauge field"
    return gauge_field(lattice, dofs=(1, dofs), **kwargs)


def gauge_links(lattice, dofs=18, **kwargs):
    "Constructs a new gauge field of links"
    return gauge_field(lattice, dofs=(4, dofs), **kwargs)

class GaugeField(LatticeField):
    "Mimics the quda::LatticeField object"

    @LatticeField.field.setter
    def field(self, field):
        LatticeField.field.fset(self, field)
        if self.reconstruct == "INVALID":
            raise TypeError(f"Unrecognized field dofs {self.dofs}")
```

```python
@property
def dofs_per_link(self):
    if self.geometry == "SCALAR":
        dofs = prod(self.dofs)
    else:
        dofs = prod(self.dofs[1:])
    if self.iscomplex:
        return dofs * 2
    return dofs

@property
def reconstruct(self):
    "Reconstruct type of the field"
    dofs = self.dofs_per_link
    if dofs == 12:
        return "12"
    if dofs == 8:
        return "8"
    if dofs == 10:
        return "10"
    if sqrt(dofs / 2).is_integer():
        return "NO"
    return "INVALID"

@property
def quda_reconstruct(self):
    "Quda enum for reconstruct type of the field"
    return getattr(lib, f"QUDA_RECONSTRUCT_{self.reconstruct}")

@property
def ncol(self):
    "Number of colors"
    if self.reconstruct == "NO":
        dofs = self.dofs_per_link
        ncol = sqrt(dofs / 2)
        assert ncol.is_integer()
        return int(ncol)
    return 3
```

Lyncs-QUDA

```python
@property
def quda_params(self):
    "Returns an instance of quda::GaugeFieldParams"
    params = lib.GaugeFieldParam()
    lib.copy_struct(params, super().quda_params)
    params.reconstruct = self.quda_reconstruct
    params.geometry = self.quda_geometry
    params.link_type = self.quda_link_type
    params.gauge = to_pointer(self.ptr)
    params.create = lib.QUDA_REFERENCE_FIELD_CREATE
    params.location = self.quda_location
    params.t_boundary = self.quda_t_boundary
    params.order = self.quda_order
    params.nColor = self.ncol
    return params

@property
def quda_field(self):
    "Returns an instance of quda::GaugeField"
    self.activate()
    if self._quda is None:
        self._quda = make_shared(lib.GaugeField.Create(self.quda_params))
    return self._quda
```

```python
def exponentiate(self, coeff=1, mul_to=None, out=None, conj=False, exact=False):
    """
    Exponentiates a momentum field
    """

    if out is None:
        out = mul_to.new() if mul_to is not None else self.new(reconstruct="NO")
    if mul_to is None:
        mul_to = out.new()
        mul_to.unity()

    lib.updateGaugeField(
        out.quda_field, coeff, mul_to.quda_field, self.quda_field, conj, exact
    )
    return out

def update_gauge(self, mom, coeff=1, out=None, conj=False, exact=False):
    """
    Updates a gauge field with momentum field
    """
    return mom.exponentiate(
        coeff=coeff, mul_to=self, out=out, conj=conj, exact=exact
    )
```

HMC Simulation

```python
@dataclass
class HMCHelper:
    beta: float = 5
    lattice: tuple = (4, 4, 4, 4)
    # Log
    last_action: float = field(init=False, default=None)

    def action(self, field):
        "Returns the action computed on field"
        self.last_action = 2 * float(
            field.compute_paths(self.paths, self.coeffs).reduce(mean=False)
        )
        return self.last_action

                ⋮

    def random_gauge(self):
        "Returns a random mom"
        out = gauge_field(self.lattice)
        out.gaussian(10)
        return out

                ⋮

    def hamiltonian(self, field, mom):
        mom2 = self.mom2(mom)
        action = self.action(field)
        return mom2 + action
```

```python
@dataclass
class Integrator:
    steps: int
    time: float = 1.0
    order: int = field(init=False, default=1)
    pos_coeffs: tuple = field(init=False, default=(0, 1))
    mom_coeffs: tuple = field(init=False, default=(0.5, 0.5))

    def __call__(self, field, mom, helper):

        # Use negative time for reversed integration
        dtime = self.time / self.steps
        for step in range(self.steps):

            for fcoeff, mcoeff in zip(self.pos_coeffs, self.mom_coeffs):
                field, mom = helper.update(field, mom, fcoeff * dtime, mcoeff * dtime)

        return field, mom

                ⋮
```

HMC Simulation

```python
@dataclass
class HMC:
    helper: HMCHelper
    integrate: Integrator

    # Metrics
    accepted: int = field(init=False, default=0)
    steps: int = field(init=False, default=0)
    denergy: float = field(init=False, default=0)
    last_accepted: bool = field(init=False, default=False)
    last_action: float = field(init=False, default=0)

    def __call__(self, field):
        mom = self.helper.random_mom()
        energy = self.helper.hamiltonian(field, mom)
        action = self.helper.last_action
        field1, mom1 = self.integrate(field, mom, self.helper)
        energy1 = self.helper.hamiltonian(field1, mom1)
        action1 = self.helper.last_action
        self.denergy = energy1 - energy

        self.steps += 1
        self.last_accepted = random() < exp(-self.denergy)
        self.accepted += self.last_accepted

        if self.last_accepted:
            self.last_action = action1
            return field1
        self.last_action = action
        return field
```

```python
def main(**kwargs):
    args = Namespace(**kwargs)

    lattice = (
        args.lattice_dims if prod(args.lattice_dims) != 0 else (args.lattice_size,) * 4
    )
    lib.set_comm(procs=args.procs)

    helper = HMCHelper(args.beta, lattice)
    integr = HMC_INTEGRATORS[args.integrator]
    integr = integr(args.t_steps)
    hmc = HMC(helper, integr)

    if args.start == "random":
        field = helper.random_gauge()
    elif args.start == "unity":
        field = helper.random_unity()
    else:
        raise ValueError("Unknown start")


    with tqdm(range(args.n_trajs)) as pbar:
        for step in pbar:
            field = hmc(field)
            pbar.set_description(f"plaq: {hmc.last_plaquette}")

            for key, val in hmc.stats.items():
                run.track(val, name=key)
```

Results

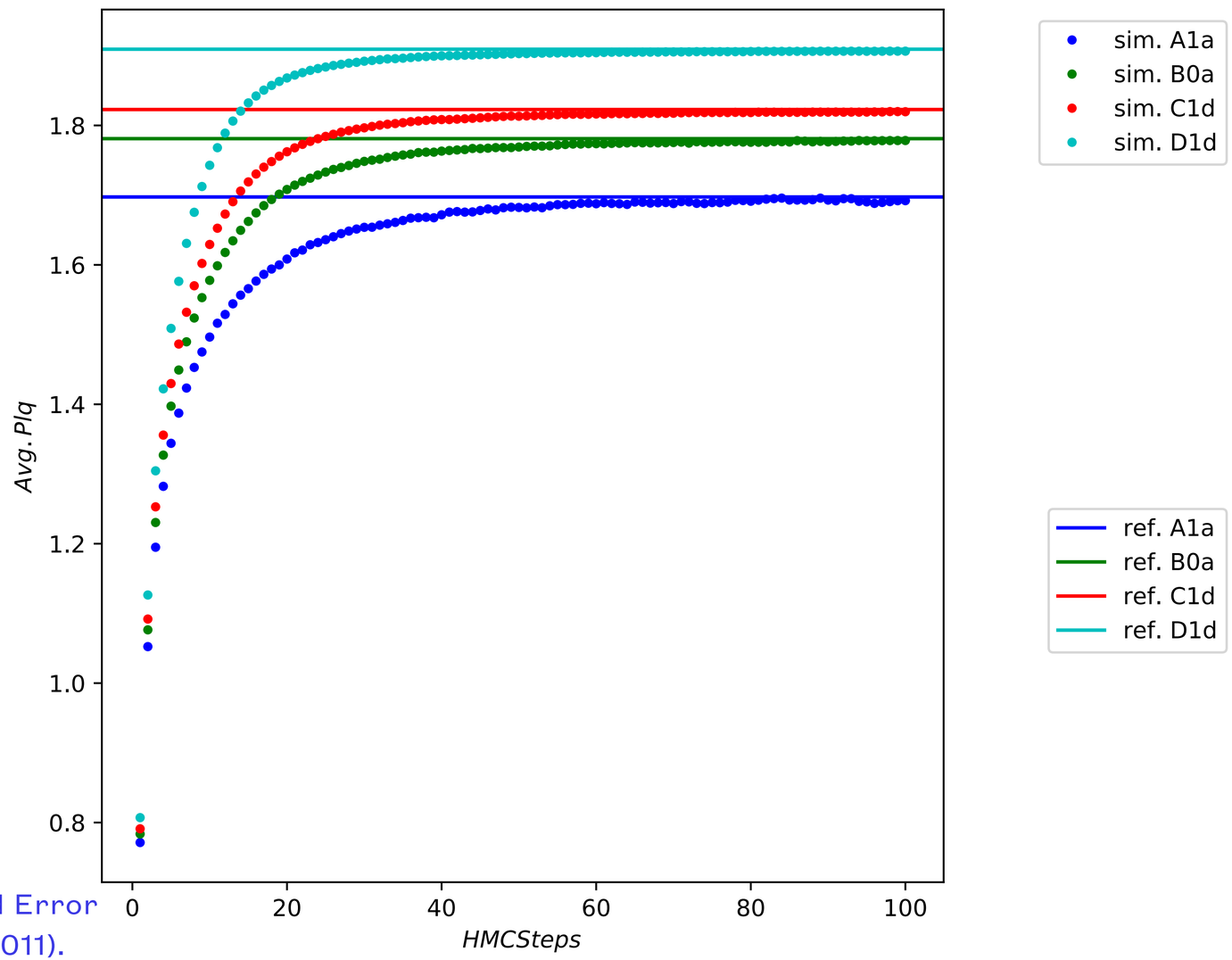
MACHINES

- Cyclamen at The Cyprus Institute
  - CPU: Intel® Xeon® Gold 6130
  - GPU: Tesla P100 x 2
- JUWELS Booster at Jülich Supercomputing Centre
  - CPU: AMD EPYC 7402
  - GPU: NVIDIA A100 Tensor Core x 4

Results

Comparison of Average Plaquettes

- Compared against values in Ref. [1]
- Ensembles (L^3 x T, β)
  - A1a (16^3x16, 5.789):
    - 1.697388(57)
  - B0a (24^3x24, 6)
    - 1.781044(6)
  - C1d (32^3x64, 6.136)
    - 1.822828(4)
  - D1d (48^3x48, 6.475)
    - 1.909347(2)



[1] S. Schaefer, R. Sommer, and F. Virotta, Critical Slowing down and Error Analysis in Lattice QCD Simulations, Nuclear Physics B 845, 93 (2011).