SIMULATeQCD - a SImple MUlti-GPU LATtice code for QCD calculations

Luis Altenkort[1], Dennis Bollweg[2], David Clarke[1], Lukas Mazur[3], Olaf Kaczmarek[1], Rasmus Larsen[4], Christian Schmidt[1], Philipp Scior[2]
[1] Bielefeld University, [2] Brookhaven National Lab, [3] Paderborn Center for Parallel Computing, [4] University of Stavanger
HotQCD Collaboration

Introduction

Increasing GPU power across a competitive market of various GPU manufacturers and GPU-based supercomputers pushes lattice programmers to develop code usable for multiple APIs. In this poster we showcase SIMULATeQCD [1], a Simple Multi-GPU Lattice code for QCD calculations, developed and used by the HotQCD collaboration for large-scale projects on both NVIDIA and AMD GPUs. Our code has been made publicly available on GitHub [1]. We explain our design strategy, give a list of available features and modules, and provide our most recent benchmarks on state-of-the-art supercomputers.

Design strategy

`SIMULATeQCD` is a multi-GPU, multi-node lattice code written using `C++17` and utilizing the OOP paradigm and modern `C++` features.

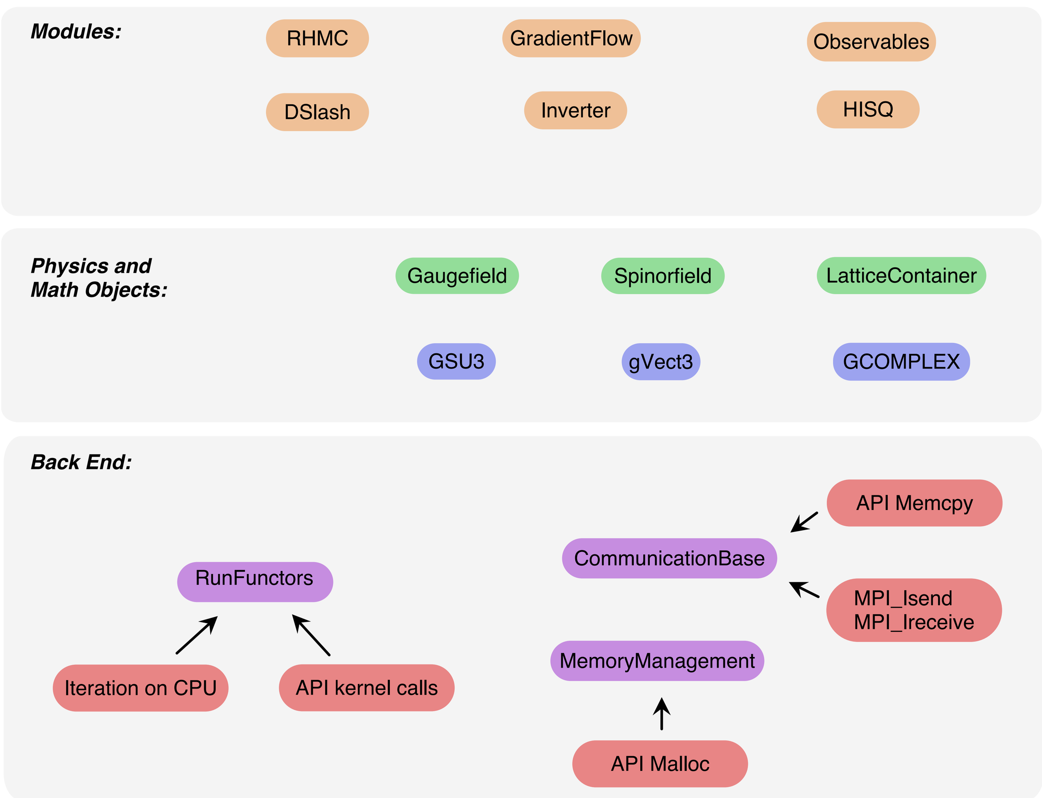

Figure 1: Sketch of inheritance hierarchy. Modules inherit from physics and math objects, which in turn inherit from the back end. Image taken from [3].

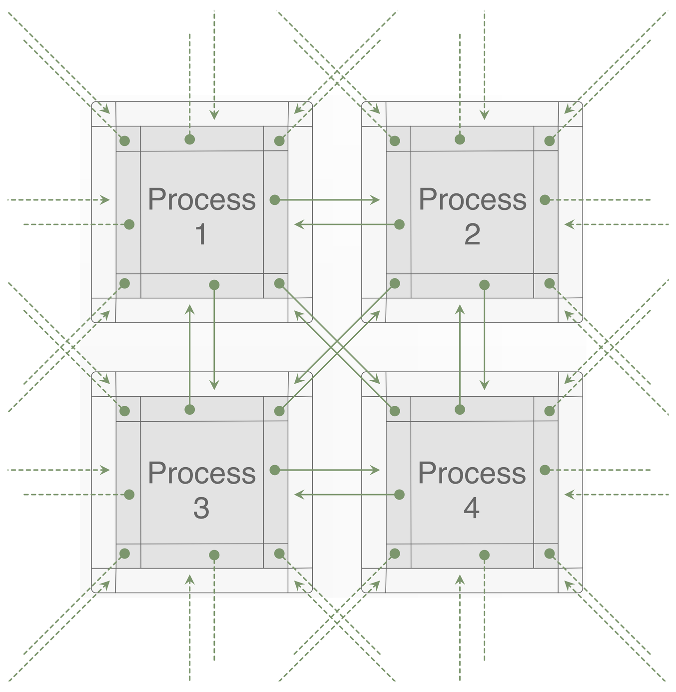

Figure 2: Schematic halo exchange for four processes in two dimensions, each process containing a sublattice. The bulk is indicated by the dark gray squares, and the halo is indicated by light grey squares. Image taken from [2].

Benchmarks

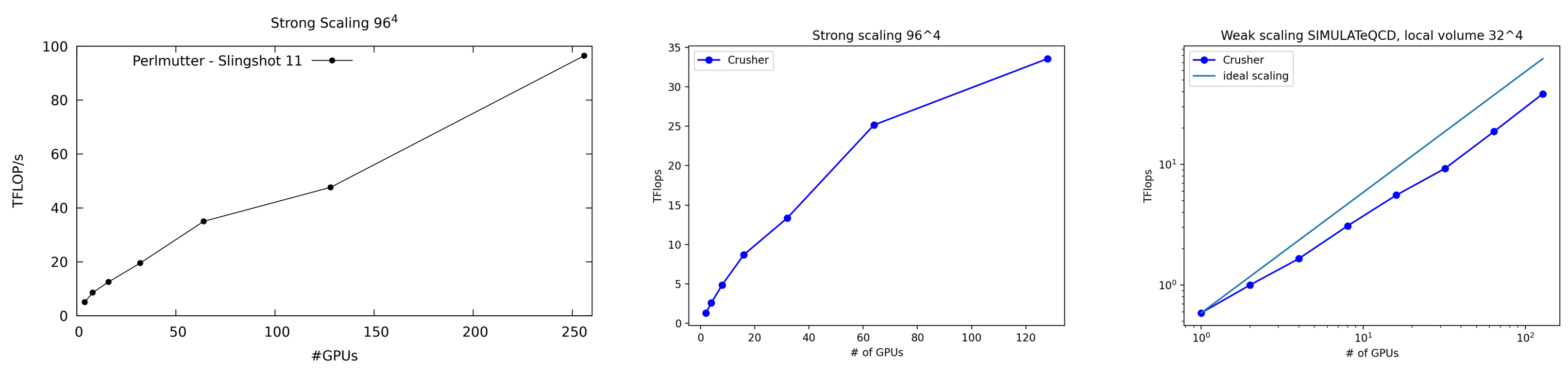

Figure 3: Strong scaling results for the HISQ $\not{D}$-operator on Perlmutter (left) and Crusher (middle) and weak scaling results on Crusher (right).

Perlmutter: NVIDIA A100 (4 GPUs per Node), slingshot interconnect.

Crusher: AMD MI250X (4 GPUs/8 GPDs per Node), slingshot interconnect.

References

[1] SIMULATeQCD public code repository, https://github.com/LatticeQCD/SIMULATeQCD.

[2] L. Mazur, Topological Aspects in Lattice QCD, Ph.D. thesis, Bielefeld U. (2021). doi:10.4119/unibi/2956493.

[3] D. Bollweg, L. Altenkort, D. A. Clarke, O. Kaczmarek, L. Mazur, C. Schmidt, P. Scior, H.-T. Shu, HotQCD on multi-GPU Systems, PoS LATTICE2021 (2022) 196, arXiv:2111.10354.

Functors: abstracting away the technical

```
template<class floatT, bool onDevice, size_t HaloDepth, CompressionType comp>
struct plaquetteKernel{

  gaugeAccessor<floatT, comp> gAcc;

  plaquetteKernel(Gaugefield<floatT, onDevice, HaloDepth, comp> &gauge) : gAcc(gauge.getAccessor()) {}

  __device__ __host__ floatT operator()(gSite site) {

    typedef GIndexer<All, HaloDepth> GInd;

    GSU3<floatT> temp;

    floatT result = 0;
    for (int nu = 1; nu < 4; nu++) {
      for (int mu = 0; mu < nu; mu++) {
        GSU3<floatT> tmp = gAcc.template getLinkPath<All, HaloDepth>(site, nu, mu, Back(nu));
        result += tr_d(gAcc.template getLinkPath<All, HaloDepth>(site, Back(mu)), tmp);
      }
    }
    return result;
  }
};
```

Listing 1: Functor `PlaquetteKernel`. Allows arbitrary precision `floatT`, to run on GPU with `onDevice==True`, for arbitrary `HaloDepth`, and arbitrary `CompressionType`. This functor takes a `Gaugefield` object as argument, whose elements are accessed in memory through the `gaugeAcessor gAcc`, set to point to the `Gaugefield` accessor in the initializer list. The argument of `operator()` indicates that this functor will be iterated over sites. We indicate with `All` that we run over both even and odd parity sites. `getLinkPath` multiplies all links starting at `site`, following a path in the specified directions.

```
const int halodepth = 0;
const bool useGPU = true;
double plaq;

typedef GIndexer<All, halodepth> GInd;

Gaugefield<double, useGPU, halodepth> gauge

LatticeContainer<useGPU, double> latContainer
latContainer.adjustSize(GInd::getLatData().vol4);

latContainer.template iterateOverBulk<All, halodepth>(plaquetteKernel<floatT, HaloDepth>(gauge));
latContainer.reduce(plaq, GInd::getLatData().vol4);

plaq /= (GInd::getLatData().globalLattice().mult()*18);
```

Listing 2: Using an iterator to calculate the plaquette with the functor given above. We instantiate a `LatticeContainer`, which is needed for the reduction. Our iterator, `iterateOverBulk` will assign each `gSite` in the bulk of a sublattice to a GPU thread.

```
template<typename Accessor, typename Functor, typename CalcReadInd, typename CalcWriteInd>
__global__ void performFunctor(Accessor res, Functor op, CalcReadInd calcReadInd,
                               CalcWriteInd calcWriteInd, const size_t size_x) {
    size_t i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i >= size_x) {
        return;
    }
#ifdef USE_CUDA
    auto site = calcReadInd(blockDim, blockIdx, threadIdx);
#elif defined USE_HIP
    auto site = calcReadInd(dim3(blockDim), GetUint3(dim3(blockIdx)), GetUint3(dim3(threadIdx)));
#endif
    res.setElement(calcWriteInd(site), op(site));
}


template<typename CalcReadInd, typename CalcWriteInd, typename Functor>
void RunFunctors::iterateFunctor(Functor op, CalcReadInd calcReadInd, CalcWriteInd calcWriteInd,
                                  const size_t elems, ...) {
...
#ifdef USE_CUDA
    performFunctor<<< gridDim, blockDim, ... >>> (getAccessor(), op, calcReadInd, calcWriteInd, elems);
#elif defined USE_HIP
    hipLaunchKernelGGL(performFunctor, gridDim, blockDim, ...,
                       getAccessor(), op, calcReadInd, calcWriteInd, elems);
#endif
...
}
```

Listing 3: Sketch of iterator implementation. API-dependent constructions are wrapped inside `performFunctor` and `iterateFunctor` methods. The user can decide at compile time whether to use the CUDA or HIP API.

Supported actions and algorithms

- Highly improved staggered quark action (RHMC)
- Standard Wilson gauge action (heatbath and over-relaxation)
- Conjugate gradient
  - multi right-hand side inverter
  - multi shift inverter
- Gradient flow with adaptive step-size
- Multi-level and blocking algorithms
- Gauge fixing (over-relaxation)
- Correlator class
- more to be added . . .

UNIVERSITÄT BIELEFELD
Fakultät für Physik

CRC-TR 211
Strong-interaction matter under extreme conditions

PUNCH 4 NFDI