HotQCD on multi-GPUs

D. Bollweg
for the HotQCD Collaboration

Bielefeld University

Lattice 2021, MIT, 07/28/2021

Outline

HotQCD GPU Code

- GPU Code for lattice QCD (quenched, staggered or HISQ gauge field generation, gradient flow, etc ...).
- Started in late 2017/2018 by **Lukas Mazur** as a thesis project, quickly became the new standard within HotQCD.
- Written in C++ with CUDA (or HIP*) for GPU acceleration.
- Multi-GPU support, D2D communication via CUDA P2P, CUDA-aware MPI and regular MPI (user choice).
- In use in large-scale computing projects on Top500 systems including Summit (OLCF), Marconi100 (CINECA), JUWELS (JSC), Piz Daint (CSCS).

UNIVERSITÄT BIELEFELD
Faculty of Physics

Contributors:
Akio Tomiya, Alexei Bazavov, Anirban Lahiri, Anna-Lena Lorenz, Battogtokh Purev, Bent Buttwill, Christian Schmidt, David Clarke, Dennis Bollweg, Dibyendu Bala, Fabian Hesse, Frithjof Karsch, Guido Nicotra, Hai-Tao Shu, Hauke Sandmeyer, Heng-Tong Ding, Hiroshi Ohno, Jishnu Goswami, Kevin Zambello, Lorenzo Dini, Luis Altenkort, Lukas Mazur, Marcel Rodekamp, Marius Neumann, Markus Klappenbach, Min Lin, Mischa Jaensch, Mugdha Sarkar, Olaf Kaczmarek, Peter Petreczky, Philipp Scior, Prasad Hegde, Qing Yuan, Rasmus Larsen, Rishabh Thakkar, Sajid Ali, Shengtai Li, Simon Dentinger, Simran Singh, Swagato Mukherjee, Tristan Ueding, Wei-Ping Huang, Xiao-Dan Wang, Yu Zhang, ... **and many others!**

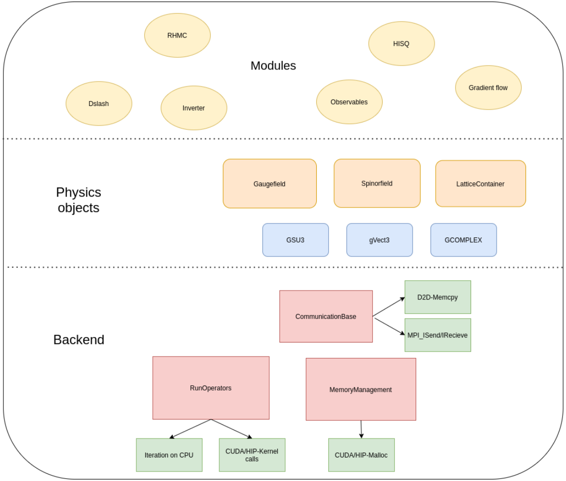
Code design

UNIVERSITÄT BIELEFELD
Faculty of Physics

We want our code to be:

- Intuitive & accessible for physics users,
- future-proof,
- performant.

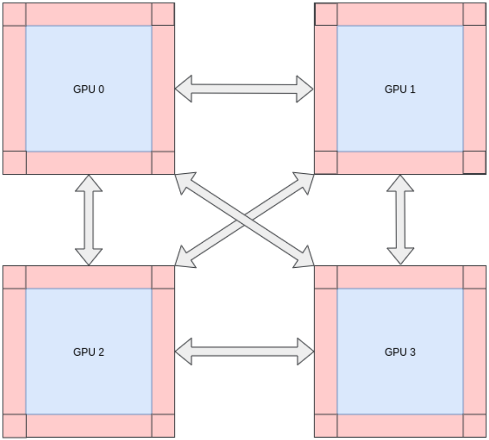
- Follow OOP paradigm.
- Separate low-level GPU code from high-level "physics" code.
- Expression templates and overloading to express calculations intuitively without sacrificing performance.
- Custom kernels via function objects.



Code design

Distribution onto multiple GPUs:

- Split lattice onto GPUs (blue).
- Extend local lattices by halos (red).
- Communicate halos for stencil computations.
- Overlap communication & computation.



Simple examples

```
Gaugefield<floatT, true, HaloDepth> gauge(...);

//simply multiply/add your physics objects
gauge = 2 * gauge;

gauge = gauge * 0.5 + gsu3_one<floatT>();


//update halos (if necessary)
gauge.updateAll();
```

```
//define custom kernel as function object
template<class floatT, size_t HaloDepth>
struct SimpleGaugeFunctor {
    gaugeAccessor<floatT> gAcc;

    SimpleGaugeFunctor(Gaugefield<floatT, true, HaloDepth> &gaugeIn) :
            gAcc(gaugeIn.getAccessor()) {}

    __host__ __device__ GSU3<floatT> operator()(gSiteMu thisLink) {

        GSU3<floatT> result = 2 * gAcc.getLink(thisLink);

        return result;
    }
};

//call custom kernel with iterateOver... methods:
gauge.iterateOverBulkAllMu(SimpleGaugeFunctor(gauge));
```

Performance - Multi-RHS $\not{D}$

$\sim$60% of (HISQ) RHMC run time is spent performing matrix inversions (CG) dominated by $\not{D}\psi_x$ computation.

$$\not{D}\psi_x = \sum_{\mu=0}^{4} \left[\left(V_{x,\mu}\chi_{x+\hat{\mu}} - V^{\dagger}_{x-\hat{\mu},\mu}\chi_{x-\hat{\mu}} \right) + \left(W_{x,\mu}\chi_{x+3\hat{\mu}} - W^{\dagger}_{x-3\hat{\mu},\mu}\chi_{x-3\hat{\mu}} \right) \right]$$

$$V_{x,\mu}: \quad 3 \times 3 \text{ complex matrix}, \quad W_{x,\mu}: \quad U(3) \text{ matrix}$$

- 1146 FLOP/site, 1560 byte/site $\rightarrow$ FLOP/byte $\sim$ 0.73. $\not{D}\psi_x$ computation is bandwidth bound!
- Arithmetic intensity can be increased by applying the gauge field to multiple right-hand sides (rhs) at once. 10 rhs: FLOP/byte $\sim$ 2.19.
- Further improvement by using link-compressed $W_{x,\mu}$.

Performance - JUWELS Booster

- Setup: $96^3 \times 16$ lattice, single precision.
- up to 5.5 TFLOP/s $\sim 37\%$ of A100 peak FP32.
- up to 1.36 TB/s memory throughput.
- up to 19 TFLOP/s on a single Booster node (4xA100).

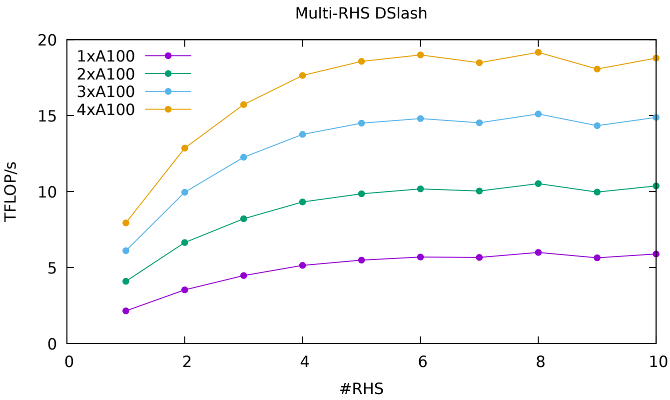


Figure: Multi-GPU performance of Multi-RHS $\not{D}$.

Performance - Multi-GPU scaling

UNIVERSITÄT BIELEFELD
Faculty of Physics

- RHMC setup: $64^3 \times 16$ global lattice size, $T = 135$MeV, phys. quark masses, single precision.
- HISQ-specific smearing and force kernels achieve nearly perfect scaling.
- RHMC scaling determined by $\not{D}$, both achieve very good on-node scaling.

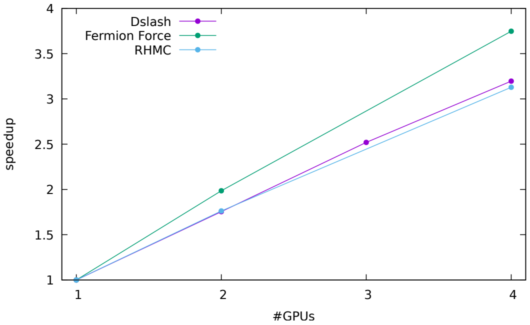


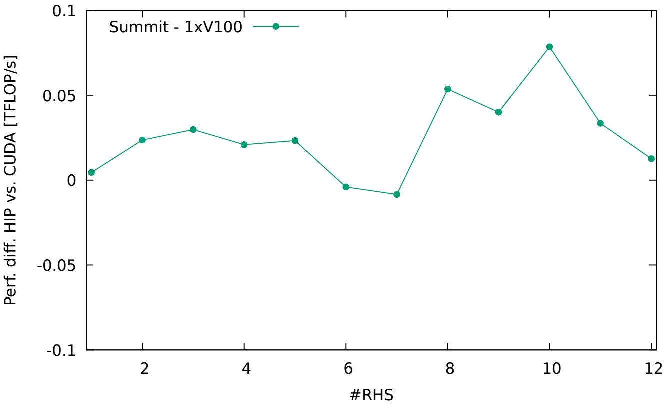
Figure: Multi-GPU scaling of $\not{D}$, force kernels and RHMC.

UNIVERSITÄT BIELEFELD
Faculty of Physics

Approach for incorporating HIP:

- No replacement for CUDA back end, both will be kept in the code.
- Wrap CUDA API functions with macros in central wrapper header.
- Hipify the code.
- Due to code design, only few lines in back end code are changed.

- Virtually no performance difference between HIP & CUDA back ends on Summit!
- Benchmarks on AMD still to come.



Summary & Outlook

- We are developing a lattice QCD GPU code based on modern C++ with CUDA and HIP* back ends.
- Our code achieves good performance and on-node scaling across recent supercomputer architectures.
- Preparations for the exascale: HIP on NVIDIA GPUs looks promising, benchmarks on AMD GPUs are planned.
- Preparations for a future open source release are ongoing.