An application user experience: Lattice QCD & QUDA

Balint Joo – OLCF

Oak Ridge Leadership Computing Facility
User Meeting 2021 (Virtual)

Thursday, June 24, 2021

joob AT ornl.gov

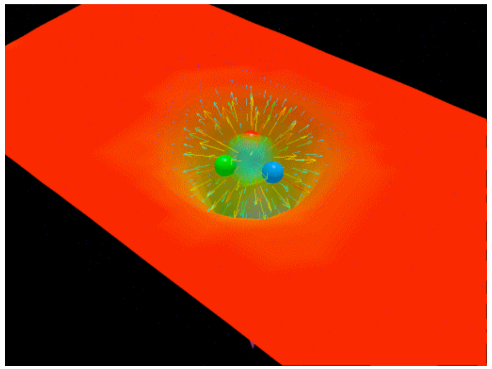
ORNL is managed by UT-Battelle LLC for the US Department of Energy

Rough outline

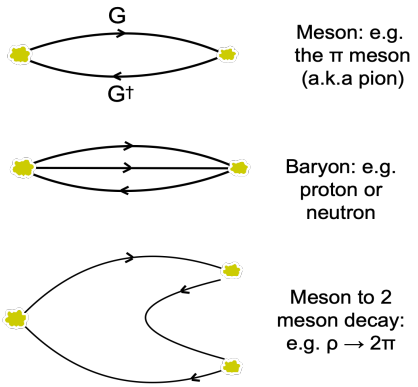
- About Lattice QCD and the Software Stack
- Porting QUDA
  - QUDA portability working group
  - the search for CUDA-idioms using HIPIFY
  - some refactorings
    - Abstracting Kernel launches & tuning
    - Easy abstractions: RNG & FFT
    - Trickier: Streams & Events
  - Some noteworthy issues...
- Where we stand today?
- Where to we go next?

Lattice QCD & Lattice QCD Codes

- Lattice QCD (LQCD) is the only known, non-perturbative, model-independent, method to carry out calculations in the strong coupling regime of QCD – the theory of the strong nuclear force.
- LQCD calculations are important to the missions of the DOE SC NP and HEP offices
- Several QCD codes out there: Chroma, CPS, MILC, GRID, BQCD, OpenQCD, etc.
- Key computational motifs:
  - Hybrid Molecular Dynamics Monte Carlo
  - Sparse linear solves on a regular grid (nearest/next to nearest neighbor stencil operators)
  - Tensor contractions (via batched ZGEMMs) for analysis

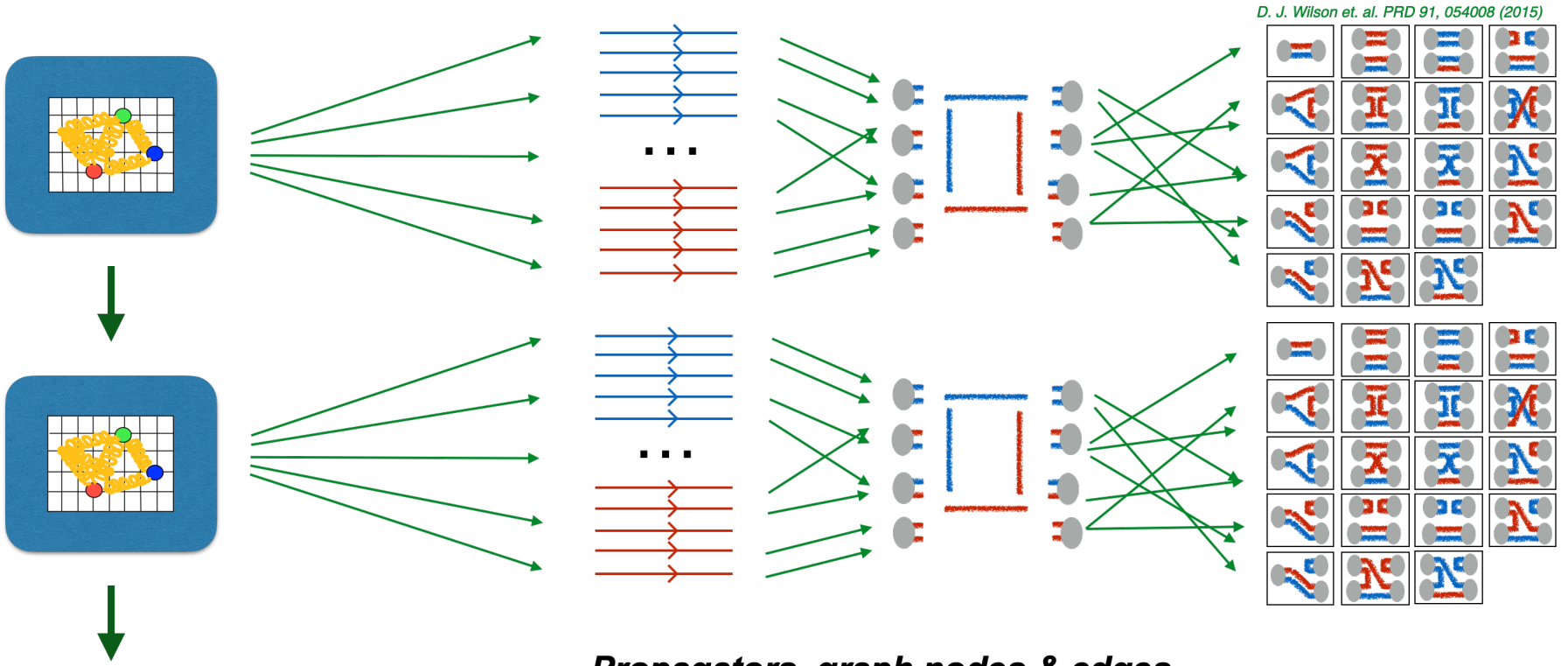


Flux tubes between 3 quarks. Produced by and courtesy of Dr Derek Leinweber, Centre for the Subatomic Structure of Matter (CSSM) and Department of Physics, University of Adelaide, 5005 Australia Copyright © 2003, 2004.

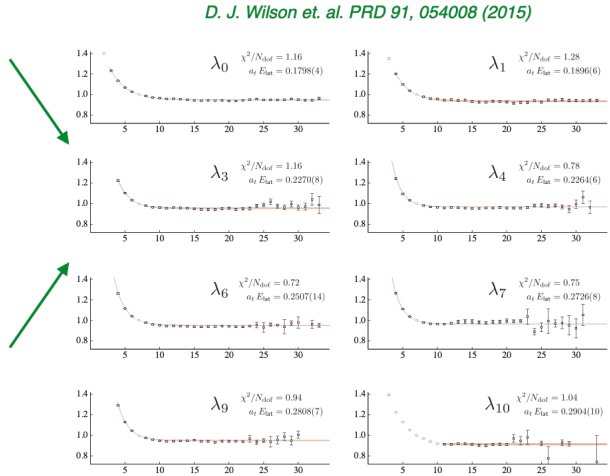


Simple quark line diagrams. Lines are 'valence quark propagators' Contractions with Dirac matrices, giving spin-parity to the state, occur at the yellow vertices

The typical LQCD workflow



D. J. Wilson et. al. PRD 91, 054008 (2015)



D. J. Wilson et. al. PRD 91, 054008 (2015)

Configuration Generation

- fews streams of large jobs
  - ~100-1000 GPUs
- Aim for O(1K-10K) trajectories
- Solver: matrix smoothly evolves
- Strong Scaling Limited

Propagators, graph nodes & edges eigenvectors etc.

- Many intermediate size jobs
  - O(10) GPUs each
- e.g. **O(1M) solves/config** for spectroscopy
- Solver: same matrix, many right hand sides
- Throughput limited

Graph Contractions

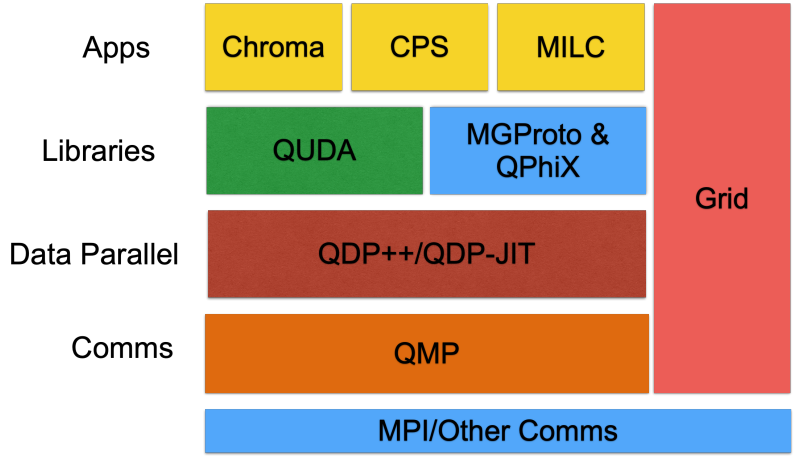
- Single Node / GPU
- O(10,000) Graphs per config
- Graph Optimization and Throughput problem

Correlation Function Fitting and Analysis

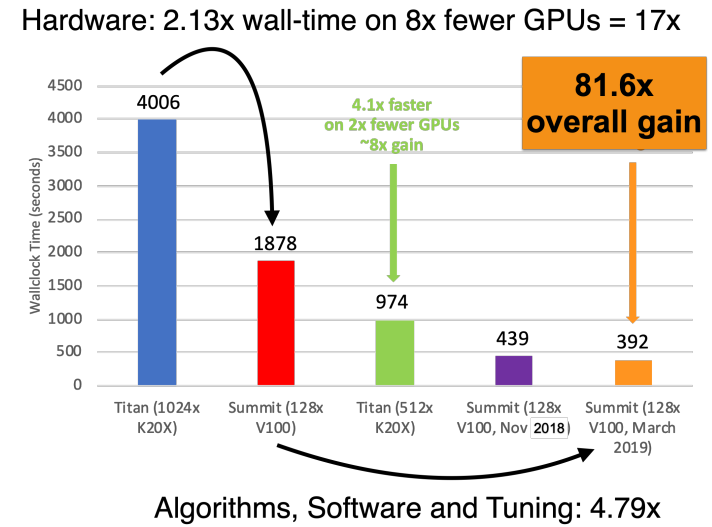
- workstations

LQCD Software in the US

- Layered approach based on design developed in SciDAC1-4
- Applications make use of libraries
  - Optimized Solvers/Dirac Operators
    - QUDA, MGProto, BFM, etc..
  - Data Parallel Productivity Layer (QDP++)
  - Parallel I/O and Communications wrapping Layer (QMP/QIO)
- QUDA Library for GPUs is a critical component
  - provides state of the art solvers for LQCD originally on NVIDIA GPUs. Porting to AMD GPUs is the focus of this talk.
  - lead developer Kate Clark and her team are Dev-Tech-s at NVIDIA
- On GPUs accelerated solvers are not enough! Even small amount of serial code can cause Amdahl's law bottleneck.
  - QDP-JIT is a version of QDP++ productivity layer for GPUs
  - Uses JIT Compilation through the LLVM framework to generate PTX code for NVIDIA GPUs and now also AMDGCN ISA for ROCm



The USQCD software layers



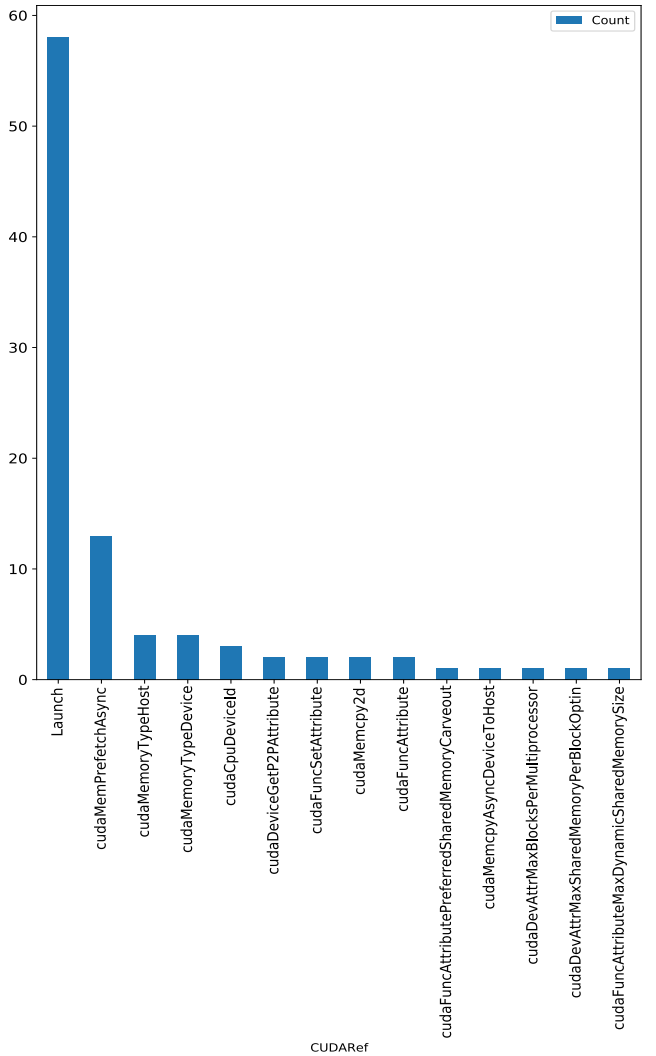
The "Money Plot": Performance gains from Titan o Summit, incorporating Multigrid and force Gradient Integrators into the gauge generation

Porting Strategy: QUDA + QDP-JIT

- The plan was to port QUDA and QDP-JIT for Frontier
  - Chroma is 99% written in terms of QDP++/QDP-JIT constructs calling out to QUDA for solvers.
  - QUDA ports to new exascale architectures also benefit the MILC and CPS codes.
- QUDA developers and vendors formed a Portability working group
  - QUDA development team from NVIDIA, developers from ORNL, ANL, LLNL, FNAL and COE staff from the Frontier COE from AMD, and the Aurora COE from Intel
  - NVIDIA's interest: refactor for use with std. C++ features/NVHPC
    - move towards using pSTL to express parallelism
    - clean up code, apply modern C++ idioms
    - Working group lead: K. Clark
  - OLCF/LLNL and AMD interest: Frontier, El Capitan and HIP/ROCm
    - B. Joo (OLCF), D. Howarth (LLNL), D. McDougall (AMD), C. Robeck (AMD)
  - Argonne and Intel Interests: Aurora, SYCL/DPC++ and OpenMP-offload
    - J. Osborn (ALCF), X-Y Jin (ALCF), A. Strelchenko (FNAL), P. Steinbrecher (Intel)
  - WG rules: no NDA, no dissing other companies or their products!
- The QDP-JIT work was undertaken by Frank Winter (QDP-JIT lead)

Initial Discovery using hipexamine-perl.sh

- Ran hipexamine-perl.sh over the source
  - false positives: QUDA routines starting with "cuda"
  - Majority of true cuda-refs
    - kernel launch
    - Memory (alloc, H2D-D2H tranfers, async, shared memory, P2P)
    - Streams
    - Events
    - Dependent external libs (CUB, cuFFT, cuRAND)
    - warp-shuffle operations (exist in hip but name incompatible)
    - scattered __CUDA_ARCH__ and other CUDA specific #ifdefs
  - some funnies:
    - no host `sincos()` with hipcc (GNU extensions), device OK
    - no host `rsqrt()` with hipcc, device OK
- We submitted 2 Pull Requests on GitHub to HIPIFY
  - better whitelisting/blacklisting (e.g. excluding directories)
  - some more known false positives

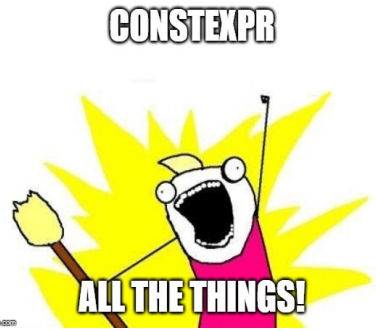


Major Restructurings

- QUDA API
  - `qudaMemcpy()`, `qudaLaunchKernel()`, etc…
- Kernel Launch Abstraction
  - `Kernel1D`, `Kernel2D`, `Kernel3D`, `Reduction`, `BlockReduction`, `TransformReduce`
  - These 'launchers' are templated on their 'dispatcher functors' and their arguments (similar in style to Kokkos)
  - All derive from '`Tunable`' class to autotune block and grid dimensions
- Device Independent Streams
  - pre-create all 8-9 streams used. Pass around index of stream
- Device Independent Events:
  - `using qudaEvent_t = void* ;`
  - manipulate with `qudaEvent()` functions in the QUDA API
  - cast to `cudaEvent_t` / `hipEvent_t` etc only in the back end API definitions
- Modern C++ techniques – mostly compile time
  - **`constexpr`** functions to remove/reduce macro use
    - e.g. fixed device properties, like warp size, minimum block size etc.
  - SFINAE (**`std::enable_if<>`**) to select various features
    - e.g. to pass arguments via regular function arguments or shared memory.

```
template <template <typename> class Functor,
          typename Arg,
          bool grid_stride = false>
__forceinline__ __device__
void Kernel1D_impl(const Arg &arg)
{
  Functor<Arg> f(arg); // instantiate

  // Compute 1D Thread Index
  auto i = threadIdx.x
         + blockIdx.x * blockDim.x;

  // Potentially grid strided execution
  while (i < arg.threads.x) {

    f(i);  // Execute!

    // Deal with grid stride
    if (grid_stride)
        i += gridDim.x * blockDim.x;
    else
        break;
  }
}
```

1D Kernel in QUDA

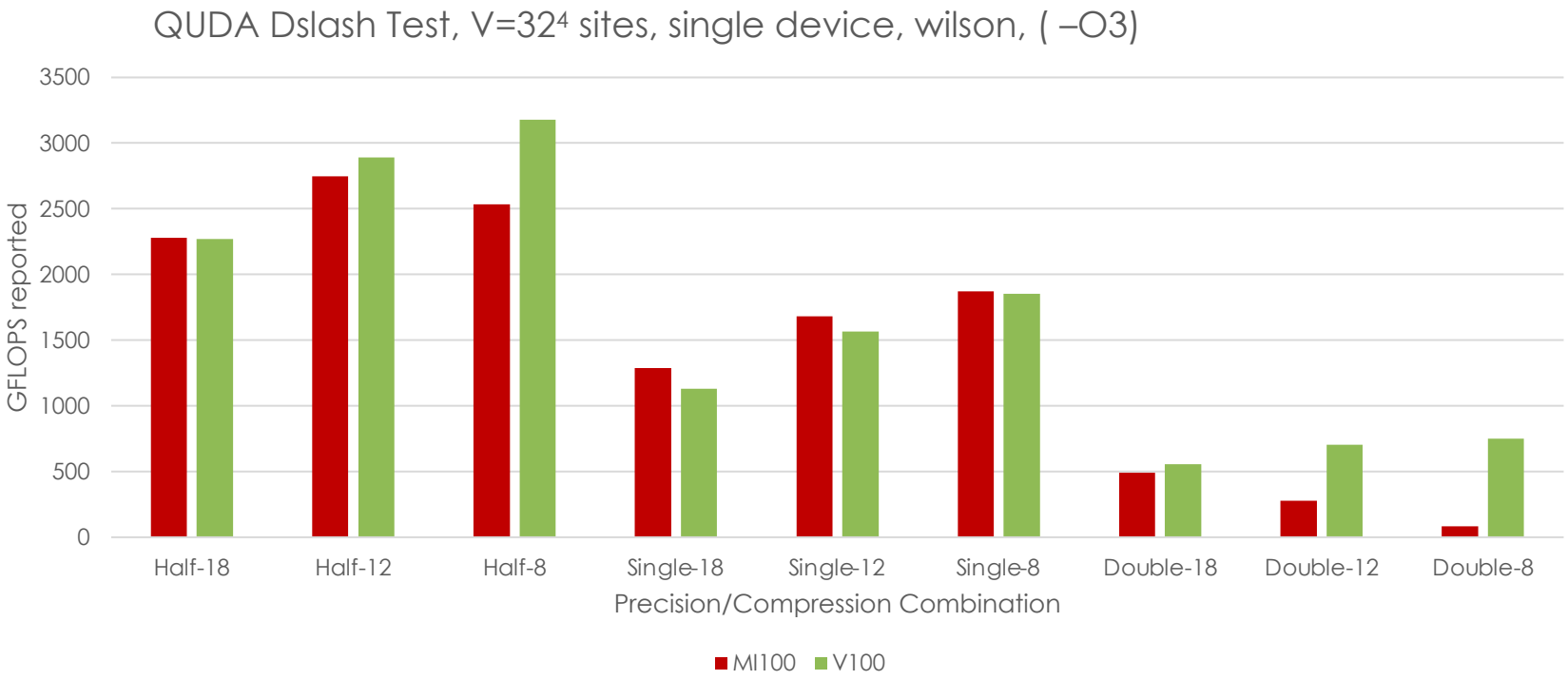


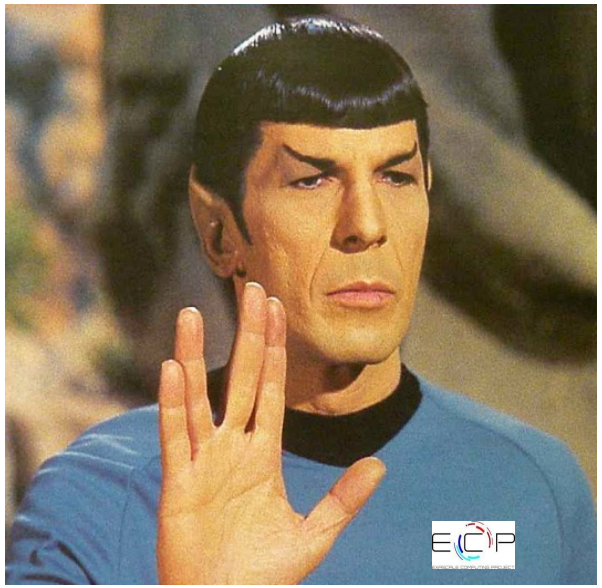
CppCon17 talk by Ben Deane and Jason Turner

But its not all roses... (it's tulips) – Bumps along the way

- There have been some rough edges with the compilers since ROCm-3.4
  - missing functions (e.g. related to IPC/P2P) – now all added
  - often could work around issues by building our own 'upstream' compiler
    - from upstream LLVM or from AMD amd-stg-open branch on their GitHub
  - the occasional mysterious segfault
    - fixed by putting limits on shared memory region size.
  - the occasional weird mis-compilation
    - the curious case of the non-terminating loop – ticket filed but really hard to repro.
  - the 4 hours it took to compile one file (now reduced with compiler flags)
    - full build (double, single, half precisions, 8/12/18 compression + Multigrid) now about 40 min
  - ...
- As of ROCm-4.2 all our issues are resolved, worked around and potentially have tickets with AMD
- **All new systems have rough edges. We go through it, so you don't have to suffer (excessively)!**

Some Preliminary Numbers from Spock





- Performances as reported by QUDA, using HIP events for timing
  - not yet gone back for a ROCprof based check on bandwidth
  - those of you who have seen my talk this morning may recall event based timing, rocprof timestamps, and roctrace, all adding different overheads.
- HIP DP results decrease dramatically for gauge field compression
  - needs root cause (higher register pressure? spills? device trig functions not being used?)

Status Summary and Next Steps

- The Chroma stack now builds on AMD systems using QUDA and QDP-JIT
  - also many build improvements: autoconf->CMake for USQCD stack
  - Spack packaging also in development (I need to learn about Spack + ROCm)
- Future steps:
  - multi-device/multi-node testing and profiling (we have just begun this)
  - root-cause performance issues and fix
  - run our ECP FOMs
- Overall experience
  - We have come a long way in terms of capabilities and stability and will have Chroma ready for Frontier.
  - The staff at HPE, AMD and locally at OLCF are super capable and helpful: if you have issues please get in touch with us at *help AT olcf.ornl.gov*

Acknowledgements

- None of this would have been possible without the work of the QUDA portability working group:
  - K. Clark (NVIDIA), D. Howarth (LLNL), X-Y Jin (ALCF), B. Joo (OLCF), D. McDougall (AMD), J. C. Osborn (ALCF), C. Robeck (AMD), P. Steinbrecher (Intel), A. Strelchenko (FNAL) and especially without the refactoring leadership of K. Clark
- I would like to gratefully acknowledge funding under ECP Application Integration

- I didn't talk about it here, but the work on QDP-JIT and Analysis code porting is also proceeding in parallel (work of F. Winter, E. Romero, J. Chen and R. Edwards at JLab) as is work on MILC, GRID and the other USQCD codes all via ECP LatticeQCD AD