LATTICE QCD ON MODERN GPU SYSTEMS

Mathias Wagner
1. Generate an ensemble of gluon field configurations “gauge generation”
   Produced in sequence, with hundreds needed per ensemble
   Strong scaling required with 100-1000 TFLOPS sustained for several months
   50-90% of the runtime is in the linear solver
   O(1) solve per linear system
   Target $16^4$ per GPU

2. “Analyze” the configurations
   Can be farmed out, assuming ~10 TFLOPS per job
   Task parallelism means that clusters reign supreme here
   80-99% of the runtime is in the linear solver
   Many solves per system, e.g., $O(10^6)$
   Target $24^4$-$32^4$ per GPU

\[
D_{ij}^{\alpha\beta}(x, y; U) \psi_j^\beta(y) = \eta_i^\alpha(x)
\]

or $Ax = b$

Simulation Cost $\sim a^{-6} V^{5/4}$
Finite difference operator in LQCD is known as Dslash
Assign a single space-time point to each thread
V = XYZT threads, e.g., V = 24^4 => 3.3x10^6 threads
Looping over direction each thread must
- Load the neighboring spinor (24 numbers x8)
- Load the color matrix connecting the sites (18 numbers x8)
- Do the computation
- Save the result (24 numbers)
Each thread has (Wilson Dslash) 0.92 naive arithmetic intensity
QUDA

• Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, TIFR, tmLQCD, etc.

• Provides:
  - Various solvers for all major fermionic discretizations, with multi-GPU support
  - Additional performance-critical routines needed for gauge-field generation

• Maximize performance
  - Exploit physical symmetries to minimize memory traffic
  - Mixed-precision methods
  - Autotuning for high performance on all CUDA-capable architectures
  - Eigenvector and deflated solvers (Lanczos, EigCG, GMRES-DR)
  - Multigrid solvers for optimal convergence Multi-source solvers
  - Domain-decomposed (Schwarz) preconditioners for strong scaling
  - Strong-scaling improvements

• A research tool for how to reach the exascale
QUDA - LATTICE QCD ON GPUS
http://lattice.github.com/quda, BSD license

QUDA v1.0.0

mathiaswagner released this on Jan 10

Version 1.0.0 - 10 January 2020

- Add support for CUDA 10.2: QUDA 1.0.0 is supported on CUDA 7.5-10.2 using either GCC or clang compilers. CUDA 10.x and either GCC >= 6.x or clang >= 6.x are highly recommended.

- Significant improvements to the CMake build system and removal of the legacy configure build.
QUDA CONTRIBUTORS
10+ years - lots of contributors

Ron Babich (NVIDIA)
Simone Bacchio (Cyprus)
Michael Baldhauf (Regensburg)
Kip Barros (LANL)
Rich Brower (Boston University)
Nuno Cardoso (NCSA)
Kate Clark (NVIDIA)
Michael Cheng (Boston University)
Carleton DeTar (Utah University)
Justin Foley (Utah -> NIH)
Joel Giedt (Rensselaer Polytechnic Institute)
Arjun Gambhir (William and Mary)
Steve Gottlieb (Indiana University)
Kyriakos Hadjiyiannakou (Cyprus)

Dean Howarth (LLNL)
Bálint Joó (Jlab)
Hyung-Jin Kim (BNL -> Samsung)
Bartek Kostrzewa (Bonn)
Claudio Rebbi (Boston University)
Hauke Sandmeyer (Bielefeld)
Guochun Shi (NCSA -> Google)
Mario Schröck (INFN)
Alexei Strelchenko (FNAL)
Jiqun Tu (Columbia -> NVIDIA)
Alejandro Vaquero (Utah University)
Mathias Wagner (NVIDIA)
Evan Weinberg (NVIDIA)
Frank Winter (Jlab)
AUTOTUNING
RECOMPILE AND RUN

Autotuning provides performance portability

Code from 2008 runs unchanged* (now rewritten - same perf better maintainability)

faster than Volta
QUDA’S AUTOTUNER
ensuring optimal kernel performance

virtual C++ class “Tunable” that is derived for each kernel you want to autotune

By default Tunable classes will autotune 1-d block size, shared memory size, grid size
  Derived specializations do 2-d and 3-d block size tuning

Tuned parameters are stored in a std::map and dumped to disk for later reuse

Built in performance metrics and profiling

User just needs to
  State resource requirements: shared memory per thread and/or per block, total number of threads
  Specify a tuneKey which gives each kernel a unique entry and break any degeneracy
GENERATIONAL COMPARISON

$F_{\mu\nu}$ kernel - batched 3x3 multiplication

GFLOPS

K80

4-21% of peak

P100

9-35% of peak

V100

6-37% of peak
MIXED PRECISION
LINEAR SOLVERS

QUDA supports a wide range of linear solvers
CG, BiCGstab, GCR, Multi-shift solvers, etc.

Condition number inversely proportional to mass
Light (realistic) masses are highly singular
Naive Krylov solvers suffer from critical slowing down at decreasing mass

Entire solver algorithm must run on GPUs
Time-critical kernel is the stencil application
Also require BLAS level-1 type operations

while (|r_k| > ε) {
    β_k = (r_k,r_k)/(r_{k-1},r_{k-1})
    p_{k+1} = r_k - β_k p_k
    q_{k+1} = A p_{k+1}
    α = (r_k,r_k)/(p_{k+1},q_{k+1})
    r_{k+1} = r_k - α q_{k+1}
    x_{k+1} = x_k + α p_{k+1}
    k = k+1
}

conjugate gradient
MIXED-PRECISION CG

Apply Dslash in sloppy precision (single, half)
Reliable residual replacement in high precision
Ensures accuracy of final result

Half-precision storage:
- Stencil elements $\in [-1,1]$ (Link):
  - 16-bit fixed point
- Grid elements (Spinor):
  - 16-bit fixed point (24 numbers)
  - float (exponent, 1 number)
Use fp32 for actual arithmetics
**MIXED-PRECISION CG**

**double-half**
- Maintain solution vectors in high precision
  - Including the partial accumulator
- When true residual is injected, re-project the direction vector
- Use Polak-Ribière formula
  \[ \beta_k := \frac{z_{k+1}^T (r_{k+1} - r_k)}{z_k^T r_k} \]

**double-half alt**
- Residual replacement strategy of van der Worst and Ye
**MIXED-PRECISION CG**

**double-half**
- Maintain solution vectors in high precision
  - Including the partial accumulator
- When true residual is injected, re-project the direction vector
- Use Polak-Ribière formula
  \[ \beta_k := \frac{z_{k+1}^T (r_{k+1} - r_k)}{z_k^T r_k} \]

**double-half alt**
- Residual replacement strategy of van der Worst and Ye
MIXED-PRECISION CG

double-half
- Maintain solution vectors in high precision
  - Including the partial accumulator
- When true residual is injected, re-project the direction vector
- Use Polak-Ribière formula
  \[ \beta_k = \frac{z_{k+1}^T (r_{k+1} - r_k)}{z_k^T r_k} \]

double-half alt
- Residual replacement strategy of van der Worst and Ye
MIXED-PRECISION CG

mixed precision:

apply Dslash in sloppy precision (single, half)

reliable residual replacement in high precision

ensures double-precision accuracy of final result

virtually identical iteration count
EIGENSOLVERS

Multiple workflows require repeated solution with different RHS with the same matrix

Multigrid not amenable to all linear operators

Eigenvector deflation is a robust alternative applicable to all operators
  Deflate out low modes from linear operator to accelerate the solver
  Cost of eigensolver is amortized if we solve enough RHS
  Aside: also use deflation to accelerate multigrid

Memory overheads can be limiting factor
  May require storage of 1000s of vectors, ideally in fast memory
MIXED-PRECISION DEFLATION

V=48³x12, HISQ operator, physical light quarks, tol 10⁻¹⁰, 2xV100

Configuration provided by HotQCD collaboration (Mukherjee et al.)
DEFLATION STABILIZES LOW PRECISION

V=48³x12, HISQ operator, physical light quarks, tol 10⁻¹⁰

Solver Iterations

Out of memory

Tri-precision solver
Outer - Inner - Eigenvector

Number of eigenvectors
DEFLATION STABILIZES LOW PRECISION

\( V=48^3 \times 12 \), HISQ operator, physical light quarks, tol 10\(^{-10} \)

- **double-single-single**
- **double-half-single**

**Tri-precision solver**

**Outer - Inner - Eigenvector**

**Solver Iterations**

Out of memory
DEFLATION STABILIZES LOW PRECISION

V=48³×12, HISQ operator, physical light quarks, tol 10⁻¹⁰

Solver Iterations

0 16 32 64 128 256 512 1024

0 4000 8000 12000 16000

- double-single-single
- double-half-single
- double-half-half

Tri-precision solver
Outer - Inner - Eigenvector
DEFLATION STABILIZES LOW PRECISION

$V=48^3 \times 12$, HISQ operator, physical light quarks, tol $10^{-10}$

Solver Iterations

- double-single-single
- double-half-single
- double-half-half
- double-quarter-half

Tri-precision solver
Outer - Inner - Eigenvector

Number of eigenvectors

Double, single, single
Double, half, single
Double, half, half
Double, quarter, half
**MIXED-PRECISION DEFLATION**

V = $48^3 \times 12$, HISQ operator, physical light quarks, tol $10^{-10}$, 2xV100

---

Configuration provided by HotQCD collaboration (Mukherjee et al)

---

- CG double
- defCG double-half-single
- CG double-single
- defCG double-half-half
- CG double-half
- defCG double-quarter-half
- CG double-quarter

---

**Final Solver Speedup**

- 512 vectors
- 1024 vectors
- 1024 vectors

---

62x speedup

---

Similar setup times
STRONG SCALING

Multiple meanings
- Same problem size, more nodes, more GPUs
- Same problem, next generation GPUs
- Multigrid - strong scaling within the same run (not discussed here)

To tame strong scaling we have to understand the limiters
- Bandwidth limiters
- Latency limiters
MULTI-GPU BUILDING BLOCKS

- Halo packing Kernel
- Interior Kernel
- Halo communication
- Halo update Kernel
DGX-1 nodes
  8x V100 GPUs connected through NVLink
  4x EDR for inter-node communication
  Optimal placement of GPUs and NIC

Balanced GPU / IB configuration

understand limiters on a single node first
MULTI-GPU PROFILE
overlapping comms and compute

32^4 local volume, single precision

P2P copies
REDUCING LOCAL PROBLEM SIZE

single GPU performance

GFlop/s vs. Lattice length

- half
- single
- double

strong scaling
REDUCING LOCAL PROBLEM SIZE

single GPU performance

GFlop/s vs. Lattice length for half, single, and double precision. The graph shows strong scaling at the smallest reasonable problem size.
## STRONG SCALING PROFILE

**overlapping comms and compute**

<table>
<thead>
<tr>
<th>8x49.26ns</th>
<th>+249.20ns</th>
<th>+249.3ns</th>
<th>+249.32ms</th>
<th>+249.34ms</th>
<th>+249.36ms</th>
<th>+249.38ms</th>
<th>+249.4ms</th>
<th>+249.42ms</th>
<th>+249.44ms</th>
<th>+249.46ms</th>
</tr>
</thead>
</table>

### Packing kernel
- p...
- dslashG...
- dslashG...
- p...

### Halo kernels (fused)
- ds...
- ds...
- p...

### Interior kernel
- p...

### P2P copies
- ...
- ...
- ...
- ...
- ...
- ...

### DGX-1, 1x2x2x2 partitioning

**16^4 local volume, half precision**
STRONG SCALING PROFILE

Latencies ate my scaling

16^4 local volume, half precision
REDUCING API OVERHEADS

Packing kernel writes to remote GPU using CUDA IPC

16^4 local volume, half precision
NVSHMEM

Implementation of OpenSHMEM1, a Partitioned Global Address Space (PGAS) library

NVSHMEM features
- Symmetric memory allocations in device memory
- Communication API calls on CPU (standard and stream-ordered)
- Kernel-side communication (API and LD/ST) between GPUs
- NVLink and PCIe support (intranode)
- InfiniBand support (internode)
- Interoperability with MPI and OpenSHMEM libraries

1 SHMEM from Cray's “shared memory” library, https://en.wikipedia.org/wiki/SHMEM
DSLASH NVSHMEM IMPLEMENTATION

First exploration

Keep general structure of packing, interior and exterior Dslash

Use `nvshmem_ptr` for intra-node remote writes (fine-grained)
   Packing buffer is located on remote device
Use `nvshmem_putmem_nbi` to write to remote GPU over IB (1 RDMA transfer)

Need to make sure writes are visible: `nvshmem_barrier_all_on_stream`
   or barrier kernel that only waits for writes from neighbors
NVSHMEM DSLASH
first exploration

16\textsuperscript{4} local volume, half precision
NVSHMEM + FUSING KERNELS
no extra packing and barrier kernels needed
ÜBER KERNEL

über kernel (packing + interior + exterior)
MULTI-NODE STRONG SCALING
DGX SuperPOD (DGX2 nodes: 16xV100 (32GB), 8xEDR IB)

Wilson Dslash, $64^3 \times 128$ global volume, half precision

16$^4$ local volume, half precision
MULTI-NODE STRONG SCALING
DGX SuperPOD (DGX2 nodes: 16xV100 (32GB), 8xEDR IB)

Wilson Dslash, $64^3 \times 128$ global volume, half precision

sweet spot for simulations

16^4 local volume, half precision
APPLICATION SCALING
MILC NERSC BENCHMARK OVERVIEW

- MILC NERSC Benchmark comes in 4 lattice sizes
  - small $18^3 \times 36$, medium $36^3 \times 72$, large $72^3 \times 144$, x-large $144^3 \times 288$
- Benchmark runs the RHMC algorithm
  - Dominated by the multi-shift CG sparse linear solver (stencil operator)
  - Also have auxiliary “Force” and “Link” computations
- Since 2012 MILC has built-in QUDA support
  - Enabled through a Makefile option
  - All time-critical functions off loaded to QUDA
MILC HMC SCALING ON DGX-2

NERSC MEDIUM BENCHMARK $36^3 \times 72$

- Running with MPI
- other part scales reasonably (not limited by communication)
- solver part needs improvements
MILC SOLVER SCALING ON DGX-2

NERSC MEDIUM BENCHMARK $36^3 \times 72$

- multi-shift solver
- mixed precision: double-single
- refinement: double-half
- MPI version overlaps BLAS + comms
- NVSHMEM scales beyond 32 GPUs
MULTIPLICATIVE SPEEDUP
HMC typically dominated by solving the Dirac equation, but
  Few solves per linear system
  Can be bound by heavy solves (c.f. Hasenbusch mass preconditioning)

Multigrid setup must run at speed of light
  Reuse and evolve multigrid setup where possible
  Use the same null space for all masses (setup run on lightest mass)
  Evolve null space vectors as the gauge field evolves (Lüscher 2007)
  Update null space when the preconditioner degrades too much on lightest mass
CHROMA HMC-MG ON SUMMIT

From Titan running 2016 code to Summit running 2019 code we see >82x speedup in HMC throughput

Multiplicative speedup coming from machine and algorithm innovation

Highly optimized multigrid for gauge field evolution
NODE PERFORMANCE OVER TIME
Multiplicative speedup through software and hardware

Time to solution is measured time to solution for solving the Wilson operator against a random source on a 24x24x24x64 lattice, $\beta=5.5$, $M_\pi = 416$ MeV. One node is defined to be 3 GPUs.
QUDA ROADMAP
On to QUDA 2.0

Multi-rhs block solvers for all stencils
Contraction framework
Improved strong scaling through NVSHMEM
Beyond just regular QCD

Longer term: Investigate how well QUDA runs on C++17 pSTL

Post feature requests here: https://github.com/lattice/quda/issues
MULTIPLE RIGHT-HAND SIDES
48³x12, HISQ, single precision, one code

Volta
Pascal
Maxwell
Kepler
Fermi

Gflop/s

# rhs

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

0 750 1,500 2,250 3,000

3.5x
2.5x
QUDA - LATTICE QCD ON GPUS

Widely used for Lattice QCD applications on GPUs

State of the art solvers using mixed precision

- Multigrid
- Deflation
- Block-Krylov solver

All components for gauge field evolution

Portable high-performance kernels through auto-tuning and careful optimization
Tuned Multi-GPU scaling

GPU centric communication with NVSHMEM takes CPU limitations out

Multiplicative speedup from hardware and software: more science