EFFECTIVE USE OF MIXED PRECISION FOR HPC

Kate Clark, Smoky Mountain Conference 2019
AGENDA

Why Mixed Precision

Lattice Quantum Chromodynamics

Mixed Precision and Krylov Solvers

Mixed Precision and Multigrid

Tensor cores

Future Challenges

Summary
WHY MIXED PRECISION?

There are many reasons to consider different precisions
- Reduce memory traffic
- Reduce network traffic
- Reduce memory footprint
- Accelerated hardware in current architecture
- Suitable numerical properties for the algorithm at hand

Accelerate or even *improve* the algorithm without compromising the quality of science
LATTICE QCD
QUANTUM CHROMODYNAMICS

The strong force is one of the basic forces of nature (along with gravity, EM and weak)

It’s what binds together the quarks and gluons in the proton and the neutron (as well as hundreds of other particles seen in accelerator experiments)

Responsible for the particle zoo seen at sub-nuclear scales (mass, decay rate, etc.)

QCD is the theory of the strong force
It’s a beautiful theory...

\[ \langle \Omega \rangle = \frac{1}{Z} \int [dU] e^{- \int d^4 x L(U)} \Omega(U) \]

...but
Theory is highly non-linear $\Rightarrow$ cannot solve directly
Must resort to numerical methods to make predictions

Lattice QCD
Discretize spacetime $\Rightarrow$ 4-d dimensional lattice of size $L_x \times L_y \times L_z \times L_t$
Finite spacetime $\Rightarrow$ periodic boundary conditions
PDEs $\Rightarrow$ finite difference equations

Consumer of 10-20% of public supercomputer cycles
Traditionally highly optimized on every HPC platform for the past 30 years
STEPS IN AN LQCD CALCULATION

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 \( \sim a^{-6} V^{5/4} \)
LATTICE QCD IN A NUTSHELL

$$\langle \Omega \rangle = \frac{1}{Z} \int [dU] e^{-\int d^4 x L(U)} \Omega(U)$$

Multi GPU Parallelization

Experiment

Theory

Large Hadron Collider

Brookhaven National Laboratory

Davies et al.
QUDA

- “QCD on CUDA” - [http://lattice.github.com/quda](http://lattice.github.com/quda) (C++14, open source, BSD license)
- Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, TIFR, etc.
- Various solvers for all major fermionic discretizations, with multi-GPU support
- Maximize performance
  - Mixed-precision methods (runtime specification of precision for maximum flexibility)
  - Exploit physical symmetries to minimize memory traffic
  - Autotuning for high performance on all CUDA-capable architectures
  - Domain-decomposed (Schwarz) preconditioners for strong scaling
  - Eigenvector and deflated solvers (Lanczos, EigCG, GMRES-DR)
  - Multi-RHS solvers
  - Multigrid solvers for optimal convergence
- A research tool for how to reach the exascale (and beyond)
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 (BU)
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)
Alejandro Vaquero (Utah University)
Mathias Wagner (NVIDIA)
Evan Weinberg (NVIDIA)
Frank Winter (Jlab)
MIXED PRECISION AND
KRYLOV SOLVERS
MAPPING THE DIRAC OPERATOR TO CUDA

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 reduces memory traffic
  Exact SU(3) matrix compression (18 => 12 or 8 real numbers)
  Use reduced precision
MULTI GPU BUILDING BLOCKS

- Halo packing Kernel
- Interior Kernel
- Halo communication
- Halo update Kernel
SINGLE GPU PERFORMANCE

“Wilson Clover” stencil (Chroma)

Tesla V100
CUDA 10.1
GCC 7.3

GFLOPS

double
single

L

8 12 16 20 24 28 32 36 40

0

400 800 1200 1600
QUDA’S 16-BIT FIXED POINT FORMAT
In production since 2009

Link field - Defines the sparse matrix elements
  SU(3) matrices that live between all adjacent sites on the 4-d grid
  All elements $\in [-1,1]$ => very natural to use 16-bit fixed-point representation

Fermion field - The vector that appears in the linear solver
  Each 4-d grid point consists of a 12-component complex vector
  No a priori bounds on the elements
  Use per-site $L_{\text{inf}}$ norm to normalize the site vector and use 16-bit fixed point
  Retain global dynamic range with local 16-bit mantissa

Low precision used only as a storage type and computation done in FP32
SINGLE GPU PERFORMANCE
“Wilson Clover” stencil (Chroma)

Tesla V100
CUDA 10.1
GCC 7.3

~1200 GB/s
~1300 GB/s
~1300 GB/s

GFLOPS

double
single
half

L

8 12 16 20 24 28 32 36 40
LINEAR SOLVERS

LQCD requires a range of sparse iterative 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) {
   \[ \beta_k = \frac{(r_k, r_k)}{(r_{k-1}, r_{k-1})} \]
   \[ p_{k+1} = r_k - \beta_k p_k \]
   \[ q_{k+1} = A p_{k+1} \]
   \[ \alpha = \frac{(r_k, r_k)}{(p_{k+1}, q_{k+1})} \]
   \[ r_{k+1} = r_k - \alpha q_{k+1} \]
   \[ x_{k+1} = x_k + \alpha p_{k+1} \]
   \[ k = k+1 \]
}
Traditional approach to mixed precision is to use iterative refinement
Disadvantage: each restart means we discard the Krylov space

Instead we use Reliable Updates*
As low-precision solver progresses, residual will drift
Occasionally replace iterated residual with high-precision residual
Retains the Krylov space information
Maintain a separate partial-solution accumulator

Aside: reductions always done in fp64 regardless of data precision

```
while (|r_k| > \varepsilon) {
    r_k = b - Ax_k
    solve Ap_k = r_k
    x_{k+1} = x_k + p_k
}
```

```
if (|r_k| < \delta |b|) {
    r_k = b - Ax_k
    b = r_k
    y = y + x_k
    x_k = 0
}
```

* Sleijpen and Van der Worst, 1996

KC, Babich, Barros, Brower, Rebbi (2009)
**STABLE** MIXED-PRECISION CG

Three key ingredients

CG convergence relies on gradient vector being orthogonal to residual vector
Re-project when injecting new residual *(Strzodka and Gödecke, 2006)*

α (stepsize) chosen to minimize $|e|_A$
True regardless of underlying precision of process
Solution correction is truncated if keep the solution vector in low precision
Always keep the (partial) solution vectors in high precision

β computation relies on $(r_i, r_j) = |r_i|^2 \delta_{i,j}$
Not true in finite precision
Polak-Ribière form is equivalent and self-stabilizing

\[ \beta_k = \frac{(r_k, (r_k - r_{k-1}))}{|r_{k-1}|^2} \]
MIXED-PRECISION MILC CG SOLVER

mass = 0.01 => \( \kappa \sim 10^4, \delta = 0.1 \)
MIXED-PRECISION MILC CG SOLVER

mass = 0.001, $\kappa \sim 10^6$, $\delta = 0.1$
MIXED-PRECISION MILC CG SOLVER

mass = 0.001, \( \kappa \sim 10^6 \), \( \delta = 0.1 \)

Looking at smarter reliable update triggers based on dynamic error estimates e.g. van der Vorst and Ye
MIXED-PRECISION MILC CG SOLVER

solution time in s

- double
- double-single
- double-half
HOW LOW CAN YOU GO?

Easily extend to 8-bit fixed-point format

(Aside) finally becoming compute bound

How much precision (information) is needed to converge the solver?

Condition number dictates the precision and range required

“Wilson Clover” stencil (Chroma)
HOW LOW CAN YOU GO?

“Chroma” linear system, 3x10^6 dof

![Graph showing CG iterations versus Condition Number (κ) for different data types: double-double, double-single, double-half, double-quarter. The x-axis represents Condition Number (κ) on a log scale, and the y-axis represents CG iterations on a log scale. The graph compares the performance of different data types in terms of CG iterations required for convergence.]
HOW LOW CAN YOU GO?

“Chroma” linear system, $3 \times 10^6$ dof

8-bit is a bridge too far
MIXED PRECISION AND MULTIGRID
WHY MULTIGRID?

Optimality

Time to Solution

Speed

Wilson-clover, Strong scaling on Titan (K20X), $V = 64^{3} \times 128$, $m_{\pi} = 197$ MeV

Babich et al 2010

Clark et al (2016)

Stability

Osborn et al 2010
ADAPTIVE GEOMETRIC MULTIGRID

Based on “Adaptive Smooth Aggregation Multigrid” (Brezina et al, 2003)
Adaptively find candidate null-space vectors
  Dynamically learn the null space and use this to define the prolongator
  Algorithm is self learning

Setup
1. Set solver to be simple smoother
2. Apply current solver to random vector \( v_i = P(D) \eta_i \)
3. If convergence good enough, solver setup complete
4. Construct prolongator using fixed coarsening \( (1 - P R) v_k = 0 \)
   - Typically use 4\(^4\) geometric blocks
   - Preserve chirality when coarsening \( R = \gamma_5 \ P^\dagger \gamma_5 = P^\dagger \)
5. Construct coarse operator \( (D_c = R \ D \ P) \)
6. Recurse on coarse problem
7. Set solver to be augmented V-cycle, goto 2

Specialized form of adaptive multigrid adapted for non-Hermitian LQCD problem with geometric coarsening

Babich, Branich, Brower, KC, Manteuffel, McCormick, Osborn, Rebbi (2009)
MIXED-PRECISION MULTIGRID SOLVER

Ideal algorithm for mixed precision
Each MG pass only reduces the residual / error by an order of magnitude
Deploy as a preconditioner for an outer Krylov solver

Method
Do entire MG cycle in 16-bit precision
Wrapped by a single-precision GCR solver
Use double-precision restarts to ensure convergence

LQCD adaptive MG requires a lot of “state”
33% reduction in peak memory - can run on less nodes

Absolutely zero effect on multigrid convergence

Coarse operator perf on Pascal

<table>
<thead>
<tr>
<th>Lattice length</th>
<th>fp32</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>225</td>
<td>225</td>
</tr>
<tr>
<td>4</td>
<td>450</td>
<td>450</td>
</tr>
<tr>
<td>6</td>
<td>675</td>
<td>675</td>
</tr>
<tr>
<td>8</td>
<td>900</td>
<td>900</td>
</tr>
</tbody>
</table>

GFLOPS
MIXED-PRECISION MULTIGRID SETUP

Need to evaluate $RAP$ - pseudo batched triple matrix product

$$= \sum$$  
Coarse operator off diagonals

$$= \sum$$  
Coarse operator diagonals

Need to employ fine-grained parallelization
- Perform each batched matrix product in parallel
- Each coarse matrix element has many contributions (many-to-one)

Cannot pose as a reduction so atomically update the coarse matrix elements

Floating point atomics are non-deterministic

Use 32-bit integer atomics for associativity and determinism
From Titan running 2016 code to Summit running 2019 code we see >82x speedup in HMC throughput

Multiplicative speedup coming from machine and algorithm

Highly optimized multigrid for gauge field evolution

Mixed precision an important piece of the puzzle
- double – outer defect correction
- single – GCR solver
- half – MG preconditioner
- int32 – deterministic parallel coarsening

Data from B. Joó (Jefferson Lab). Chroma w/ QDP-JIT (F. Winter, Jefferson Lab) and QUDA. B. Joó gratefully acknowledges funding through the US DOE SciDAC program (DE-AC05-06OR23177)
Unlike CG, Multigrid stable with an 8-bit smoother
ONGOING AND FUTURE WORK

Plumb 8-bit into the full MG hierarchy

Use tensor-core accelerated direct solve for coarse grid (cf Dongarra)
  Motivation is a latency optimization
  Current: iterative local communication and computation => latency bound
  Future: single all gather collective and local direct solve
TENSOR CORES
Multi-splitting Preconditioned CG

- Motivated by lack of network bandwidth in supercomputing centers (e.g., Summit)
- Block Jacobi preconditioner for (M)DWF that correctly applies the Dirichlet boundary condition for the red-black normal op
LATTICE QCD WITH TENSOR CORES

KC, Jung, Mawhinney, Tu

Significantly reduces outer iterations...

...but local preconditioner becomes prohibitively expensive
LATTICE QCD WITH TENSOR CORES

Modern GPU have high throughput tensor-core functionality

\[
\begin{align*}
D &= \begin{pmatrix}
A_{0,0} & A_{0,1} & \cdots & A_{0,15} \\
A_{1,0} & A_{1,1} & \cdots & A_{1,15} \\
\vdots & \vdots & & \vdots \\
A_{15,0} & A_{15,1} & \cdots & A_{15,15}
\end{pmatrix} + \begin{pmatrix}
B_{0,0} & B_{0,1} & \cdots & B_{0,15} \\
B_{1,0} & B_{1,1} & \cdots & B_{1,15} \\
\vdots & \vdots & & \vdots \\
B_{15,0} & B_{15,1} & \cdots & B_{15,15}
\end{pmatrix}
\end{align*}
\]

Tesla V100
FP64: 7.5 TFLOPS
FP32: 15 TFLOPS
Tensor: 125 TFLOPS

\[
\begin{bmatrix}
1 - \kappa_b^2 M_\phi^\dagger D_w^\dagger M_5^{-1} M_\phi^\dagger D_w^\dagger M_5^{-1}
\end{bmatrix}
\begin{bmatrix}
1 - \kappa_b^2 M_5^{-1} D_w M_\phi M_5^{-1} D_w M_\phi
\end{bmatrix}
\]

fuse, fuse, fuse, fuse
## LATTICE QCD WITH TENSOR CORES

**KC, Jung, Mawhinney, Tu**

Increasingly beneficial as one strong scales

### 6144 GPUs (Summit)

<table>
<thead>
<tr>
<th>nodes</th>
<th>local volume</th>
<th>solver</th>
<th>inner iter.</th>
<th>(outer) iter.</th>
<th>r. n.</th>
<th>performance/node</th>
<th>time</th>
<th>speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>16 - 24 - 12 - 24</td>
<td>CG</td>
<td>–</td>
<td>42133</td>
<td>471</td>
<td>4.66</td>
<td>466.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>05</td>
<td>16903</td>
<td>195</td>
<td>1.56(01)/5.45(33)/37.29(53)</td>
<td>456.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>06</td>
<td>14860</td>
<td>173</td>
<td>1.56(01)/5.51(31)/37.60(58)</td>
<td>428.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>07</td>
<td>13787</td>
<td>161</td>
<td>1.56(01)/5.48(26)/37.49(60)</td>
<td>480.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>08</td>
<td>12922</td>
<td>151</td>
<td>1.56(01)/5.44(26)/37.55(63)</td>
<td>469.5</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>16 - 12 - 12 - 24</td>
<td>CG</td>
<td>–</td>
<td>42427</td>
<td>474</td>
<td>3.85</td>
<td>206.6</td>
<td>1.10x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>05</td>
<td>17625</td>
<td>203</td>
<td>1.26(01)/4.54(37)/36.21(52)</td>
<td>271.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>06</td>
<td>15425</td>
<td>179</td>
<td>1.27(01)/4.55(33)/36.26(57)</td>
<td>262.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>07</td>
<td>14409</td>
<td>168</td>
<td>1.26(01)/4.57(30)/36.39(60)</td>
<td>268.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>08</td>
<td>13597</td>
<td>159</td>
<td>1.27(01)/4.53(28)/36.35(63)</td>
<td>276.0</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>16 - 12 - 12 - 12</td>
<td>CG</td>
<td>–</td>
<td>42482</td>
<td>474</td>
<td>2.93</td>
<td>195.2</td>
<td>1.13x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>05</td>
<td>18250</td>
<td>210</td>
<td>1.00(01)/3.68(34)/34.62(45)</td>
<td>183.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>06</td>
<td>15959</td>
<td>185</td>
<td>1.01(01)/3.68(35)/34.79(54)</td>
<td>159.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>07</td>
<td>14985</td>
<td>174</td>
<td>1.01(01)/3.68(32)/35.06(58)</td>
<td>163.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSPCG</td>
<td>08</td>
<td>14287</td>
<td>167</td>
<td>1.00(01)/3.69(20)/34.76(61)</td>
<td>169.1</td>
<td></td>
</tr>
</tbody>
</table>
FUTURE CHALLENGES
HOW HIGH DO YOU NEED TO GO?

Present state of art LQCD grid size ~128³x256 => dof ~ 10^{10}

LQCD Hybrid Monte Carlo algorithms utilize a Metropolis acceptance test for accept/reject

Involves a difference measurement of \( \delta S = \psi^\dagger A_1^{-1} \psi - \psi^\dagger A_0^{-1} \psi \)

This is an extensive quality

Presently require solver tolerance of ~10^{-12} to ensure high Metropolis acceptance rate

Not long before we have to consider beyond double precision

Already putting double-double (pseudo-quad) precision in QUDA to cover this eventuality
SCALING FURTHER

NVSHMEM gets the CPU out of the way

Increasingly latency limited when we reduce precision (and GPUs get faster)
- Overhead from calling MPI routines
- Halo-region updates do not saturate the GPU

NVSHMEM: Implementation of OpenSHMEM, a Partitioned Global Address Space (PGAS) library
- Removing reliance on CPU for communication
- Parallelism for implicit compute - communication overlap
- Allows kernel-side communication (API and LD/ST) between GPUs
- Interoperability with MPI and OpenSHMEM libraries
- Improving performance while making it easier to program
MULTI-NODE STRONG SCALING
SuperPOD, Chroma stencil, $64^3 \times 128$ global volume

Disclaimer: Results from an first implementation in QUDA with a pre-release version of NVSHMEM
SUMMARY
SUMMARY

The state of the art for LQCD linear solver requires the use of mixed precision.

Optimal Hierarchical solvers can require three or more different precisions.

Some algorithms more amenable than others.

New algorithms possible that were unfeasible before.

Judicious choice of precision can lead to a significant speedup: more science.
RECOMPILE AND RUN

Code from 2008 runs unchanged