SPEEDING UP CONJUGATE GRADIENT SOLVERS BY 10X

Mathias Wagner, Developer Technology Engineer
GTC 2017 - S7387
HPC BEYOND MOORE’S LAW

going wide

CPUs and GPUs becoming wider

increase in flops is driven by more cores

also applies for CPUs (Server to mobile)

need sufficient amount of parallelism to fill architectures
HPC BEYOND MOORE’S LAW

staying local

CPUs and GPUs becoming wider

increase in flops is driven by more cores

need sufficient amount of parallelism to fill architectures
Lattice Quantum Chromodynamics

Classic Conjugate Gradient Solver

Block Krylov solvers

Efficient implementation

Time to solution improvements in the wild
LATTICE QCD: SOME BASICS

QCD partition function
4 dimensional grid (=Lattice)
  quarks live on lattice sites
  gluons live on the links
typical sizes: $24^3 \times 6$ to $256^4$
parallelization over lattice sites
  $(10^5$ to $10^9)$

$$Z_{QCD}(T, \mu) = \int \mathcal{D}A \mathcal{D}\bar{\Psi} \mathcal{D}\Psi e^{-S_E(T,\mu)}$$

includes integral over space and time
4D gauge field of $3 \times 3$
complex vars
Spinor: 4 SU3 vector
SU3 vector: 3 complex vars
QUDA - LATTICE QCD ON GPUS

http://lattice.github.com/quda

QUDA is a library for performing calculations in lattice QCD on GPUs. http://lattice.github.com/quda — Edit

Branch: develop

Latest commit f3e2aa7 a day ago

- Include
- Lib
- Tests
- .Gitignore
- CMakeLists.txt

4,621 commits
49 branches
19 releases
16 contributors

include

In ColorSpinorParam, if staggered fermions then set field dimension t...

lib

Correctly set volumeCB for parity subset references - need to check p...

tests

Requesting --test 1 with staggered_dslash_test now tests MdagM operator

.gitignore

Updates to .gitignore and renamed multigrid_benchmark to multigrid_ben...

CMakeLists.txt

added some comments to CMakeLists.txt
QUDA AUTHORS

Green: Contributors to this talk

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

Dean Howarth (Rensselaer Polytechnic Institute)
Báltint Joó (Jlab)
Hyung-Jin Kim (BNL -> Samsung)
Claudio Rebbi (Boston University)
Guochun Shi (NCSA -> Google)
Mario Schröck (INFN)
Alexei Strelchenko (FNAL)
Alejandro Vaquero (Utah University)
Mathias Wagner (NVIDIA)
Evan Weinberg (Boston University)
Frank Winter (Jlab)
THE OLD WORK HORSE
procedure CG

\[
\begin{align*}
    r^{(0)} &= b - Ax^{(0)} \\
    p^{(0)} &= r^{(0)} \\
    \text{for } k = 1, 2, \ldots \text{ until converged do} & \text{ do} \\
    z^{(k-1)} &= Ap^{(k-1)} \\
    \alpha^{(k-1)} &= \frac{|r^{(k-1)}|^2}{\langle p^{(k-1)}, z^{(k-1)} \rangle} \\
    x^{(k)} &= x^{(k-1)} + \alpha^{(k-1)} p^{(k-1)} \\
    r^{(k)} &= r^{(k-1)} - \alpha^{(k-1)} z^{(k-1)} \\
    \beta^{(k-1)} &= \frac{|r^{(k-1)}|^2}{|r^{(k)}|^2} \\
    p^{(k)} &= r^{(k)} + \beta^{(k-1)} p^{(k-1)} \\
    \text{end for} & \text{ end procedure}
\end{align*}
\]
CONJUGATE GRADIENT

procedure CG
\[
\begin{align*}
    r^{(0)} &= b - Ax^{(0)} \\
    p^{(0)} &= r^{(0)} \\
    \text{for } k &= 1, 2, \ldots \text{ until converged do} \\
    z^{(k-1)} &= Ap^{(k-1)} \\
    \alpha^{(k-1)} &= \frac{|r^{(k-1)}|^2}{\langle p^{(k-1)}, z^{(k-1)} \rangle} \\
    x^{(k)} &= x^{(k-1)} + \alpha^{(k-1)} p^{(k-1)} \\
    r^{(k)} &= r^{(k-1)} - \alpha^{(k-1)} z^{(k-1)} \\
    \beta^{(k-1)} &= \frac{|r^{(k-1)}|^2}{|r^{(k)}|^2} \\
    p^{(k)} &= r^{(k)} + \beta^{(k-1)} p^{(k-1)} \\
\end{align*}
\]
end for
end procedure
procedure CG
\[ r^{(0)} = b - Ax^{(0)} \]
\[ p^{(0)} = r^{(0)} \]
for \( k = 1, 2, \ldots \) until converged do
\[ z^{(k-1)} = Ap^{(k-1)} \]
\[ \alpha^{(k-1)} = \frac{|r^{(k-1)}|^2}{\langle p^{(k-1)}, z^{(k-1)} \rangle} \]
\[ x^{(k)} = x^{(k-1)} + \alpha^{(k-1)} p^{(k-1)} \]
\[ r^{(k)} = r^{(k-1)} - \alpha^{(k-1)} z^{(k-1)} \]
\[ \beta^{(k-1)} = \frac{|r^{(k-1)}|^2}{|r^{(k)}|^2} \]
\[ p^{(k)} = r^{(k)} + \beta^{(k-1)} p^{(k-1)} \]
end for
end procedure
CONJUGATE GRADIENT

procedure CG

\[ r^{(0)} = b - Ax^{(0)} \]
\[ p^{(0)} = r^{(0)} \]

for \( k = 1, 2, \ldots \) until converged do

\[ z^{(k-1)} = Ap^{(k-1)} \]
\[ \alpha^{(k-1)} = \frac{|r^{(k-1)}|^2}{\langle p^{(k-1)}, z^{(k-1)} \rangle} \]
\[ x^{(k)} = x^{(k-1)} + \alpha^{(k-1)} p^{(k-1)} \]
\[ r^{(k)} = r^{(k-1)} - \alpha^{(k-1)} z^{(k-1)} \]
\[ \beta^{(k-1)} = \frac{|r^{(k-1)}|^2}{|r^{(k)}|^2} \]
\[ p^{(k)} = r^{(k)} + \beta^{(k-1)} p^{(k-1)} \]
end for
end procedure

matrix-vector operation dominates runtime

caxpy BLAS operations

Reductions
STAGGERED FERMION MATRIX (DSLASH)

Krylov space solve of fermion matrix dominates runtime within inversion application of sparse Matrix (Dslash) dominates (>80%)
Highly Improved Staggered Quarks (HISQ) use next and 3rd neighbor stencil within inversion application of sparse Matrix (Dslash) dominates (>80%) Krylov space solve.

$w_x = D_{x,x'} v_{x'} = \sum_{\mu=0}^{3} \left\{ U_{x,\mu} v_{x+\hat{\mu}} - U_{x-\hat{\mu},\mu}^\dagger v_{x-\hat{\mu}} \right\} + \left\{ N_{x,\mu} v_{x+3\hat{\mu}} - N_{x-3\hat{\mu},\mu}^\dagger v_{x-3\hat{\mu}} \right\}$

- $w_x$: complex 3-dim vector
- $D_{x,x'}$: 24 byte for fp32
- $v_{x'}$: complex 3x3 matrix
- $U_{x,\mu}$: 72 byte for fp32
- $N_{x,\mu}$: complex 3x3 matrix
- $v_{x+\hat{\mu}}$: 72 byte for fp32
STAGGERED FERMION MATRIX (DSLASH)

Krylov space solve of fermion matrix dominates runtime within inversion application of sparse Matrix (Dslash) dominates (>80%).

Highly Improved Staggered Quarks (HISQ) use next and 3rd neighbor stencil

\[ w_x = D_{x,x'} v_{x'} = \sum_{\mu=0}^{3} \left[ \left\{ U_{x,\mu} v_{x+\hat{\mu}} - U_{x-\hat{\mu},\mu}^\dagger v_{x-\hat{\mu}} \right\} + \left\{ N_{x,\mu} v_{x+3\hat{\mu}} - N_{x-3\hat{\mu},\mu}^\dagger v_{x-3\hat{\mu}} \right\} \right] \]

each site (x) loads 1024 bytes for links and 384 bytes for vectors, stores 24 bytes: total 1432 bytes / site

performs 1146 flop: arithmetic intensity: 0.8 flop/byte
STAGGERED FERMION MATRIX (DSLASH)

Krylov space solve of fermion matrix dominates runtime within inversion application of sparse Matrix (Dslash) dominates (>80%)

Highly Improved Staggered Quarks (HISQ) use next and 3rd neighbor stencil

\[ w_x = D_{x,x'} v_{x'} = \sum_{\mu=0}^{3} \left[ \left\{ U_{x,\mu} v_{x+\hat{\mu}} - U_{x-\hat{\mu},\mu}^\dagger v_{x-\hat{\mu}} \right\} + \left\{ N_{x,\mu} v_{x+3\hat{\mu}} - N_{x-3\hat{\mu},\mu}^\dagger v_{x-3\hat{\mu}} \right\} \right] \]

each site (x) loads 1024 bytes for links and 384 bytes for vectors, stores 24 bytes: total 1432 bytes / site

performs 1146 flop: arithmetic intensity: 0.8 flop/byte

sensitive to memory bandwidth
THE NEW TRACTOR
BLOCK KRYLOV SPACE SOLVERS

Share the Krylov space

BlockCG solver suggested by O’Leary in early 80’s
retooled BlockCG by Dubrulle 2001
In exact precision converges in N / rhs iterations
BLOCK KRYLOV SPACE SOLVERS
Share the Krylov space

BlockCG solver suggested by O’Leary in early 80’s
retooled BlockCG by Dubrulle 2001
In exact precision converges in N / rhs iterations

Application in QCD:
Nakamura et. (modified block BiCGStab)
Birk and Frommer (block methods,
including block methods for multi shift)
BLOCK CG
share Krylov space between multiple rhs

procedure BLOCKCG
\[ R^{(0)} = B - AX^{(0)} \]
\[ P^{(0)} = R^{(0)} \]

for \( k = 1, 2, \ldots \) until converged do
\[ Z^{(k-1)} = AP^{(k-1)} \]
\[ \alpha^{(k-1)} = \left[ (P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)} \]
\[ X^{(k)} = X^{(k-1)} + P^{(k-1)} \alpha^{(k-1)} \]
\[ R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)} \]
\[ \beta^{(k-1)} = \left[ (R^{(k-1)})^H R^{(k-1)} \right]^{-1} R^{(k)} R^{(k)} \]
\[ P^{(k)} = R^{(k)} - P^{(k-1)} \beta^{(k-1)} \]
end for
end procedure
**BLOCK CG**

share Krylov space between multiple rhs

```
procedure BLOCKCG

\[ R^{(0)} = B - AX^{(0)} \]
\[ P^{(0)} = R^{(0)} \]

for \( k = 1, 2, \ldots \) until converged do

\[ Z^{(k-1)} = AP^{(k-1)} \]
\[ \alpha^{(k-1)} = \left[ (P^{(k-1)})^H Z^{(k-1)} \right]^{-1} \left( R^{(k-1)} \right)^H R^{(k-1)} \]
\[ X^{(k)} = X^{(k-1)} + P^{(k-1)} \alpha^{(k-1)} \]
\[ R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)} \]
\[ \beta^{(k-1)} = \left[ (R^{(k-1)})^H R^{(k-1)} \right]^{-1} \left( R^{(k)} \right)^H R^{(k)} \]
\[ P^{(k)} = R^{(k)} - P^{(k-1)} \beta^{(k-1)} \]

end for

end procedure
```
**BLOCK CG**

share Krylov space between multiple rhs

**procedure BLOCKCG**

\[ R^{(0)} = B - AX^{(0)} \]
\[ P^{(0)} = R^{(0)} \]

for \( k = 1, 2, \ldots \) until converged do

\[ Z^{(k-1)} = AP^{(k-1)} \]
\[ \alpha^{(k-1)} = \left[ (P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)} \]
\[ X^{(k)} = X^{(k-1)} + P^{(k-1)} \alpha^{(k-1)} \]
\[ R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)} \]
\[ \beta^{(k-1)} = \left[ (R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)} \]
\[ P^{(k)} = R^{(k)} - P^{(k-1)} \beta^{(k-1)} \]

end for

end procedure
**BLOCK CG**

share Krylov space between multiple rhs

---

**procedure** `BLOCKCG`

\[ R^{(0)} = B - AX^{(0)} \]
\[ P^{(0)} = R^{(0)} \]

**for** \( k = 1, 2, \ldots \) until converged **do**

\[ Z^{(k-1)} = AP^{(k-1)} \]
\[ \alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)}\right]^{-1} (R^{(k-1)})^H R^{(k-1)} \]
\[ X^{(k)} = X^{(k-1)} + P^{(k-1)} \alpha^{(k-1)} \]
\[ R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)} \]
\[ \beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)}\right]^{-1} (R^{(k)})^H R^{(k)} \]
\[ P^{(k)} = R^{(k)} - P^{(k-1)} \beta^{(k-1)} \]

**end for**

**end procedure**
REDUCED ITERATION COUNT

HISQ, $32^3 \times 8$, Gaussian random source

<table>
<thead>
<tr>
<th>residual</th>
<th>1.00E+00</th>
<th>1.00E-01</th>
<th>1.00E-02</th>
<th>1.00E-03</th>
<th>1.00E-04</th>
<th>1.00E-05</th>
<th>1.00E-06</th>
<th>1.00E-07</th>
<th>1.00E-08</th>
<th>1.00E-09</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterations</td>
<td>0</td>
<td>500</td>
<td>1000</td>
<td>1500</td>
<td>2000</td>
<td>2500</td>
<td>3000</td>
<td>3500</td>
<td>4000</td>
<td>4500</td>
</tr>
</tbody>
</table>
REDUCED ITERATION COUNT

HISQ, $32^3 \times 8$, Gaussian random source

1.00E-09 1.00E-08 1.00E-07 1.00E-06 1.00E-05 1.00E-04 1.00E-03 1.00E-02 1.00E-01

0 500 1000 1500 2000 2500 3000 3500 4000 4500

residual

iterations
REDUCED ITERATION COUNT

HISQ, 32³x8, Gaussian random source
REDUCED ITERATION COUNT

HISQ, $32^3 \times 8$, Gaussian random source

<table>
<thead>
<tr>
<th>Residual (10^{-n})</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00E-09</td>
<td>0</td>
</tr>
<tr>
<td>1.00E-08</td>
<td>500</td>
</tr>
<tr>
<td>1.00E-07</td>
<td>1000</td>
</tr>
<tr>
<td>1.00E-06</td>
<td>1500</td>
</tr>
<tr>
<td>1.00E-05</td>
<td>2000</td>
</tr>
<tr>
<td>1.00E-04</td>
<td>2500</td>
</tr>
<tr>
<td>1.00E-03</td>
<td>3000</td>
</tr>
<tr>
<td>1.00E-02</td>
<td>3500</td>
</tr>
<tr>
<td>1.00E-01</td>
<td>4000</td>
</tr>
<tr>
<td>1.00E+00</td>
<td>4500</td>
</tr>
</tbody>
</table>
REDUCED ITERATION COUNT

HISQ, $32^3 \times 8$, Gaussian random source

<table>
<thead>
<tr>
<th>Residual</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00E-09</td>
<td>1.00E-08</td>
<td>1.00E-07</td>
<td>1.00E-06</td>
<td>1.00E-05</td>
<td>1.00E-04</td>
</tr>
</tbody>
</table>

Plot showing the residual over iterations for different iteration counts. The plot indicates a reduction in residual as the number of iterations increases.
REDUCED ITERATION COUNT
HISQ, $32^3 \times 8$, Gaussian random source

![Graph showing residual iterations over iterations for different iteration counts.]

- Residuals decrease as iterations increase.
- The graph compares the performance of different iteration counts (1, 2, 4, 8, 12, 16).
- The x-axis represents iterations ranging from 0 to 4500.
- The y-axis represents residual values ranging from $1.00E-09$ to $1.00E+00$.
REDUCED ITERATION COUNT

HISQ, 32^3 x 8, Gaussian random source

Residual iterations vs. iterations for different reduction counts (1, 2, 4, 8, 12, 16).
BLOCK CGRQ

Prevent numerical instabilities through orthogonalization

\[ A \in \mathbb{C}^{L \times L}; \ R, B, X, Q \in \mathbb{C}^{L \times N}; \ C, S, \beta \in \mathbb{C}^{N \times N} \]

procedure \textsc{blockcgrq}(X^{L \times N}, B^{L \times N})

\[
R = B - AX \\
QC = R \\
S = I^{N \times N} \\
P = 0
\]

while not converged do

\[
P = Q + PS^\dagger \\
\beta = (P^\dagger AP)^{-1} \\
X = X + P \beta C \\
QS = Q - AP \beta \\
C = SC
\]

end while

end procedure
ORTHOGONALIZATION

THIN QR

simple approach: Gram-Schmidt or modified Gram-Schmidt
becomes prohibitively expensive

**THINQR**

<table>
<thead>
<tr>
<th>Gram-Matrix:</th>
<th>$B = R^H R$</th>
<th>$m \times m$ dot products of length $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cholesky Decomposition</td>
<td>$S^H S = B$</td>
<td>of $m \times m$ matrix</td>
</tr>
<tr>
<td>apply to vectors</td>
<td>$Q = RS^{-1}$</td>
<td>axpy $m \times m$ (output, input)</td>
</tr>
</tbody>
</table>

relies on BLAS operations and reductions. Cholesksky Decompostion on CPU.
INCREASED COST PER ITERATION

**DSLASH**

- apply Dslash to $m$ rhs
- naively scales linear
- constant time per rhs

**BLAS**

- caxpy couples $m \times m$ input vectors via $m \times m$ matrix $a$
- Quadratic scaling with $m$
- used for orthogonalization
- linear scaling of cost per rhs

**REDUCTION**

- Instead of dot product between to vectors we now need to evaluate $m \times m$ dot products
- Quadratic scaling with $m$
- used for orthogonalization
- linear scaling of cost per rhs
| # rhs | 0    | 0.333 | 0.667 | 1    | 1.333 | 1.667 | 2    | 2.333 | 2.667 | 3    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   | 16   |
|-------|------|-------|-------|------|-------|-------|------|-------|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   | 16   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

COST PER ITERATION

for one rhs

naive
DSLASH FOR MULTIPLE RHS
QCD PERFORMANCE LIMITERS

QCD is memory bandwidth bound
Dslash arithmetic intensity for HISQ ~ 0.7-0.8

exploit SU(3) symmetry:
reconstruct gauge field from 8/12 floats
QCD PERFORMANCE LIMITERS

QCD is **memory bandwidth bound**
Dslash arithmetic intensity for HISQ ~ 0.7-0.8

exploit SU(3) symmetry:
reconstruct gauge field from 8/12 floats

**Constant gauge field (A) for multiple rhs**
Reuse gauge field for multiple rhs
QCD is **memory bandwidth bound**

Dslash arithmetic intensity for HISQ ~ 0.7-0.8

exploit SU(3) symmetry:
reconstruct gauge field from 8/12 floats

**Constant gauge field (A) for multiple rhs**
Reuse gauge field for multiple rhs
For roofline model: Assume 100% cache reuse in x-direction, 50% in y-direction
WHY DON’T WE SEE EXPECTED SCALING

1 rhs Dslash agrees with roofline

Kernel Performance Is Bound By Memory Bandwidth

For device "Quadro GP100" the kernel’s compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the Device memory.
## WHY DON’T WE SEE EXPECTED SCALING

1 rhs Dslash agrees with roofline

<table>
<thead>
<tr>
<th>L2 Cache</th>
<th>Reads</th>
<th>6450241</th>
<th>669.848 GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Writes</td>
<td>124429</td>
<td>12.922 GB/s</td>
</tr>
<tr>
<td>Total</td>
<td>6574670</td>
<td>682.77</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unified Cache</th>
<th>Local Loads</th>
<th>0</th>
<th>0 B/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Local Stores</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td></td>
<td>Global Loads</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td></td>
<td>Global Stores</td>
<td>124416</td>
<td>12.92 GB/s</td>
</tr>
<tr>
<td></td>
<td>Texture Reads</td>
<td>6635520</td>
<td>689.089 GB/s</td>
</tr>
<tr>
<td>Unified Total</td>
<td>6759936</td>
<td>702.01</td>
<td></td>
</tr>
</tbody>
</table>

### Device Memory

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>5014415</th>
<th>520.74 GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Writes</td>
<td>152277</td>
<td>15.814 GB/s</td>
</tr>
<tr>
<td>Total</td>
<td>5166692</td>
<td>536.554</td>
<td>GB/s</td>
</tr>
</tbody>
</table>
WHY DON’T WE SEE EXPECTED SCALING

1 rhs Dslash agrees with roofline

<table>
<thead>
<tr>
<th></th>
<th>L2 Cache</th>
<th>Unified Cache</th>
<th>Device Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reads</td>
<td>Writes</td>
<td>Reads</td>
</tr>
<tr>
<td></td>
<td>6450241</td>
<td>124429</td>
<td>5014415</td>
</tr>
<tr>
<td></td>
<td>669.848 GB/s</td>
<td>12.922 GB/s</td>
<td>520.74 GB/s</td>
</tr>
<tr>
<td></td>
<td>6574670</td>
<td></td>
<td>5166692</td>
</tr>
<tr>
<td></td>
<td>682.77 GB/s</td>
<td></td>
<td>536.554 GB/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6574670</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CACHE BASED MULTI RHS D/SLASH

Check extreme cases with 16 rhs

Kernel Performance Is Bound By Memory Bandwidth
For device "Quadro GP100" the kernel’s compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the Texture memory.
CACHE BASED MULTI RHS DSLASH

Check extreme cases with 16 rhs

Kernel Performance Is Bound By Memory Bandwidth

For device "Quadro GP100" the kernel's compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the Texture memory.

High number of memory operations
# CACHE BASED MULTI RHS D/LASH

Check extreme cases with 16 rhs

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(in MB)</td>
<td>(in MB)</td>
<td>(in MB)</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>46681396</td>
<td>1990669</td>
<td>48672065</td>
</tr>
<tr>
<td></td>
<td>703.125 GB/s</td>
<td>29.984 GB/s</td>
<td>733.109 GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(in MB)</th>
<th>(in MB)</th>
<th>(in MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Cache</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Local Loads</td>
<td>Local Stores</td>
<td>Global Loads</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 B/s</td>
<td>0 B/s</td>
<td>0 B/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1990656</td>
<td></td>
<td>106168320</td>
</tr>
<tr>
<td></td>
<td>29.984 GB/s</td>
<td></td>
<td>1,599.13 GB/s</td>
</tr>
<tr>
<td></td>
<td>Unifed Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>108158976</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,629.113 GB/s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reads</td>
<td>Writes</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td>(in MB)</td>
<td>(in MB)</td>
<td>(in MB)</td>
</tr>
<tr>
<td></td>
<td>16534840</td>
<td>2020709</td>
<td>18555549</td>
</tr>
<tr>
<td></td>
<td>249.051 GB/s</td>
<td>30.436 GB/s</td>
<td>279.488 GB/s</td>
</tr>
</tbody>
</table>
**CACHE BASED MULTI RHS DSLASH**

Check extreme cases with 16 rhs

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L2 Cache</strong></td>
<td>46681396</td>
<td>1990669</td>
<td>48672065</td>
</tr>
<tr>
<td></td>
<td>703.125 GB/s</td>
<td>29.984 GB/s</td>
<td>733.109 GB/s</td>
</tr>
</tbody>
</table>

**Unified Cache**

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Loads</td>
<td>0</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td>Local Stores</td>
<td>0</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td>Global Loads</td>
<td>0</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td>Global Stores</td>
<td>1990656</td>
<td>29.984 GB/s</td>
<td>29.984 GB/s</td>
</tr>
<tr>
<td>Texture Reads</td>
<td>106168320</td>
<td>1,599.13 GB/s</td>
<td>1,599.13 GB/s</td>
</tr>
<tr>
<td><strong>Unified Total</strong></td>
<td>108158976</td>
<td>1,629.113 GB/s</td>
<td>1,629.113 GB/s</td>
</tr>
</tbody>
</table>

**Device Memory**

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16534840</td>
<td>2020709</td>
<td>18555549</td>
</tr>
<tr>
<td></td>
<td>249.051 GB/s</td>
<td>30.436 GB/s</td>
<td>279.488 GB/s</td>
</tr>
</tbody>
</table>

**Unified Cache is limiting performance**
REGISTER OPTIMIZATION

Hybrid implementation

Use y-dimension of CUDA blocks for rhs
One lattice site is scheduled on the same SM for all rhs
Each thread processes one rhs on one lattice site
Necessary for cache reuse of gauge field
REGISTER OPTIMIZATION

Hybrid implementation

Use y-dimension of CUDA blocks for rhs
One lattice site is scheduled on the same SM for all rhs
Each thread processes one rhs on one lattice site
Necessary for cache reuse of gauge field

Use y-dimension of CUDA blocks for rhs
One lattice site is scheduled on the same SM for all rhs
Each thread processes multiple rhs on one lattice site (reuse gauge field from registers)
Reduces cache pressure
REGISTER REUSE DSLASH

Check extreme cases with 16 rhs

Kernel Performance Is Bound By Instruction And Memory Latency
This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Quadro GP100". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.
REGISTER REUSE DSLASH

Check extreme cases with 16 rhs

Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Quadro GP100". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.
**REGISTER REUSE DSLASH**

Check extreme cases with 16 rhs

<table>
<thead>
<tr>
<th></th>
<th>L2 Cache</th>
<th>Unified Cache</th>
<th>Device Memory</th>
<th>System Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reads</td>
<td>Writes</td>
<td>Reads</td>
<td>Writes</td>
</tr>
<tr>
<td></td>
<td>36434573</td>
<td>1990669</td>
<td>16840673</td>
<td>2013116</td>
</tr>
<tr>
<td></td>
<td>772.596 GB/s</td>
<td>42.212 GB/s</td>
<td>357.107 GB/s</td>
<td>42.688 GB/s</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td></td>
<td>38425242</td>
<td></td>
<td>18853789</td>
<td></td>
</tr>
<tr>
<td></td>
<td>814.808 GB/s</td>
<td></td>
<td>399.795 GB/s</td>
<td></td>
</tr>
</tbody>
</table>

[Graph showing L2 Cache performance with metrics and bars for Idle, Low, Medium, High, Max]
**REGISTER REUSE DSLASH**

Check extreme cases with 16 rhs

<table>
<thead>
<tr>
<th>L2 Cache</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>36434573</td>
<td>772.596 GB/s</td>
</tr>
<tr>
<td>Writes</td>
<td>1990669</td>
<td>42.212 GB/s</td>
</tr>
<tr>
<td>Total</td>
<td>38425242</td>
<td>814.808 GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unified Cache</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Loads</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td>Local Stores</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td>Global Loads</td>
<td>0</td>
<td>0 B/s</td>
</tr>
<tr>
<td>Global Stores</td>
<td>1990656</td>
<td>42.212 GB/s</td>
</tr>
<tr>
<td>Texture Reads</td>
<td>50429952</td>
<td>1,069.368 GB/s</td>
</tr>
<tr>
<td>Unified Total</td>
<td>52420608</td>
<td>1,111.58 GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Memory</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>16840673</td>
<td>357.107 GB/s</td>
</tr>
<tr>
<td>Writes</td>
<td>2013116</td>
<td>42.688 GB/s</td>
</tr>
<tr>
<td>Total</td>
<td>18853789</td>
<td>399.795 GB/s</td>
</tr>
</tbody>
</table>

Unified Cache is no longer limiting performance
MULTI-SRC DSLASH ON PASCAL

Volume 24, HISQ, tuned gauge reconstruct

- double reg
- single reg
- half reg
- double
- single
- half
- single roofline
- half roofline

GFLOPS

# rhs
QUDA AUTOTUNER
runtime tuning of launch parameters and more

How to decide how to divide between tex and register reuse?
E.g. for 16 rhs possible combinations are (1,16), (2,8), (4,4), (8,2), (16,1)
QUDA AUTOTUNER
runtime tuning of launch parameters and more

How to decide how to divide between tex and register reuse?
E.g. for 16 rhs possible combinations are (1,16), (2,8), (4,4), (8,2), (16,1)

QUDA already uses an autotuner at runtime to determine optimal launch parameters
  For 1st call of each kernel and parameters like volume
  Cached on disk for reuse in subsequent calculations
How to decide how to divide between tex and register reuse?
E.g. for 16 rhs possible combinations are (1,16), (2,8), (4,4), (8,2), (16,1)

QUDA already uses an autotuner at runtime to determine optimal launch parameters
   For 1st call of each kernel and parameters like volume
   Cached on disk for reuse in subsequent calculations

Can also be used for auxiliary parameters
   here: number of rhs in registers
### COST PER ITERATION

for one rhs

<table>
<thead>
<tr>
<th># rhs</th>
<th>naive</th>
<th>multi rhs Dslash</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.333</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.667</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.333</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.667</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.333</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2.667</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3.000</td>
<td></td>
</tr>
</tbody>
</table>

*Note: The chart shows the relative cost (normalized to one naive 1 rhs) for different numbers of right-hand sides (rhs). The green line represents the naive method, and the red line represents the multi rhs Dslash method. The x-axis represents the number of rhs, and the y-axis represents the relative cost.*
OPTIMIZING LINEAR ALGEBRA
EXPLOIT GPU ARCHITECTURE
to overcome quadratically scaling

\[ y_i = \sum a_{ij} x_j + y_i \]

CUDA supports two dimensional grid blocks:
easy to exploit locality for texture cache / shared memory

GP100 Pascal Whitepaper
Figure 8. Pascal GP100 SM Unit

Designed for High-Performance Double Precision

Double precision arithmetic is at the heart of many HPC applications such as linear algebra, numerical simulation, and quantum chemistry. Therefore, one of the key design goals for GP100 was to significantly improve the delivered performance for these use cases.

Each SM in GP100 features 32 double precision (FP64) CUDA Cores, which is one-half the number of FP32 single precision CUDA Cores. A full GP100 GPU has 1920 FP64 CUDA Cores. This 2:1 ratio of single precision (SP) units to double precision (DP) units aligns better with GP100's new datapath configuration, allowing the GPU to process DP workloads more efficiently. Like previous GPU architectures, GP100 supports full IEEE 754-2008 compliant single precision and double precision arithmetic, including support for the fused multiply-add (FMA) operation and full speed support for denormalized values.

Note: Kepler GK110 had a 3:1 ratio of SP units to DP units.
EXPLOIT GPU ARCHITECTURE
to overcome quadratically scaling

\[ y_i = \sum a_{ij} x_j + y_i \]

CUDA supports two dimensional grid blocks:
easy to exploit locality for texture cache / shared memory

(0,0) (1,0) (2,0) (3,0)
(0,1) (1,1) (2,1) (3,1)
(0,3) (1,2) (2,2) (3,2)
(0,4) (1,3) (2,3) (3,4)
EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

\[ y_i = \sum a_{ij} x_j + y_i \]

CUDA supports two dimensional grid blocks:
easy to exploit locality for texture cache / shared memory

\[
\begin{align*}
y_0(0) &= a_{00} x_0(0) + a_{01} x_1(0) + \ldots \\
y_1(0) &= a_{10} x_0(0) + a_{11} x_1(0) + \ldots \\
y_2(0) &= a_{20} x_0(0) + a_{21} x_1(0) + \ldots \\
y_3(0) &= a_{30} x_0(0) + a_{31} x_1(0) + \ldots 
\end{align*}
\]
EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

\[ y_i = \sum a_{ij} x_j + y_i \]

CUDA supports two dimensional grid blocks:

easy to exploit locality for texture cache / shared memory

\[ y_0(0) = a_{00} x_0(0) + a_{01} x_1(0) + \ldots \]
\[ y_1(0) = a_{10} x_0(0) + a_{11} x_1(0) + \ldots \]
\[ y_2(0) = a_{20} x_0(0) + a_{21} x_1(0) + \ldots \]
\[ y_3(0) = a_{30} x_0(0) + a_{31} x_1(0) + \ldots \]
EXPLOIT GPU ARCHITECTURE
to overcome quadratically scaling

\[ y_i = \sum a_{ij} x_j + y_i \]

CUDA supports two dimensional grid blocks:
easy to exploit locality for texture cache / shared memory

\[
\begin{align*}
y_0(0) &= a_{00}x_0(0) + a_{01}x_1(0) + \ldots \\
y_1(0) &= a_{10}x_0(0) + a_{11}x_1(0) + \ldots \\
y_2(0) &= a_{20}x_0(0) + a_{21}x_1(0) + \ldots \\
y_3(0) &= a_{30}x_0(0) + a_{31}x_1(0) + \ldots \\
\end{align*}
\]

cache reuse
EXPLOIT GPU ARCHITECTURE
to overcome quadratically scaling

\[ y_i = \sum a_{ij} x_j + y_i \]

CUDA supports two dimensional grid blocks:
easy to exploit locality for texture cache / shared memory

\[ y_0(0) = a_{00} x_0(0) + a_{01} x_1(0) + \ldots \]
\[ y_1(0) = a_{10} x_0(0) + a_{11} x_1(0) + \ldots \]
\[ y_2(0) = a_{20} x_0(0) + a_{21} x_1(0) + \ldots \]
\[ y_3(0) = a_{30} x_0(0) + a_{31} x_1(0) + \ldots \]

\[ y_0(1) = a_{00} x_0(1) + a_{01} x_1(1) + \ldots \]
\[ y_1(1) = a_{10} x_0(1) + a_{11} x_1(1) + \ldots \]
\[ y_2(1) = a_{20} x_0(1) + a_{21} x_1(1) + \ldots \]
\[ y_3(1) = a_{30} x_0(1) + a_{31} x_1(1) + \ldots \]

cache reuse
MULTI-BLAS

caxpy

Block size

Time

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

0 0.001 0.002 0.003 0.004 0.005

caxpy  Linear  Quadratic
MULTI-BLAS

caxpy

Time

GB/s

GFLOPS

Block size

Block size

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

0.001  0.002  0.003  0.004  0.005

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

0  50  100  150  200  250  300  350  400  450  500  550  600  650  700  750  800  850  900  950  1000  1050  1100  1150  1200  1250  1300  1350  1400  1450  1500  1550  1600  1650  1700  1750  1800  1850  1900  1950  2000  2050  2100  2150  2200

GFLOPS

GB/s
MULTI-REDUCTION

cdot

- Time
- Linear
- Quadratic
MULTI-REDUCTION

cdot

Time  Linear  Quadratic

GB/s

0.00  1000  2000  3000  4000

GFLOPS

0.00  450.00  900.00  1350.00  1800.00

GB/s

0  1000  2000  3000  4000

GFLOPS

0.00  450.00  900.00  1350.00  1800.00

GB/s

0  1000  2000  3000  4000
COST PER ITERATION

for one rhs

<table>
<thead>
<tr>
<th># rhs</th>
<th>naive</th>
<th>multi rhs Dslash</th>
<th>full multi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2.333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2.667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3.333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>4.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4.333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4.667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>5.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>5.333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>5.667</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BLOCKCGQ IN THE WILD
ACTUAL PHYSICS PROJECT

G-2 project of the MILC collaboration

Calculations inherently need to solve multiples of 8 rhs

Use a two-step solution process (sloppy-solve + refine)

7x speedup for production runs
SUMMARY
BLOCKCG RQ
more parallelism and lots of locality to exploit
Reuse gauge field for Dslash
Reuse vectors in BLAS and reductions

more parallelism and lots of locality to exploit
BLOCKCG RQ

more parallelism and lots of locality to exploit

Reuse gauge field for Dslash

Reuse vectors in BLAS and reductions

avoid quadratical scaling in linear algebra and orthogonalization

creates more parallelism to saturate wider architectures

squeezes 4x number of FLOPS out of a P100
TIME TO SOLUTION

Combined effect of reduced iteration count and cost per iteration

about a factor 3 improvement in cost/iteration

additional (free) flops in BLAS/reduction drive reduced iteration count
TIME TO SOLUTION

Combined effect of reduced iteration count and cost per iteration

- Immediate drop in for CG solver
- No overhead costs

About a factor 3 improvement in cost/iteration

Additional (free) flops in BLAS/reduction drive reduced iteration count

Reduced iteration count depends on rhs, target residual and matrix condition