QCD on GPUs: Progress and Prospects

Ron Babich
NVIDIA Research

USQCD Algorithms and Computing Workshop
Livermore Valley Open Campus
November 10, 2011

Acknowlegments

- Collaborators and QUDA developers:
    - Kipton Barros (now at LANL)
    - Richard Brower (Boston U.)
    - Michael Clark (NVIDIA)
    - Justin Foley (Utah)
    - Joel Giedt (Rensselaer)
    - Steven Gottlieb (Indiana)
    - Bálint Joó (Jefferson Lab)
    - James Osborn (Argonne)
    - Claudio Rebbi (Boston U.)
    - Guochun Shi (NCSA)

Overview

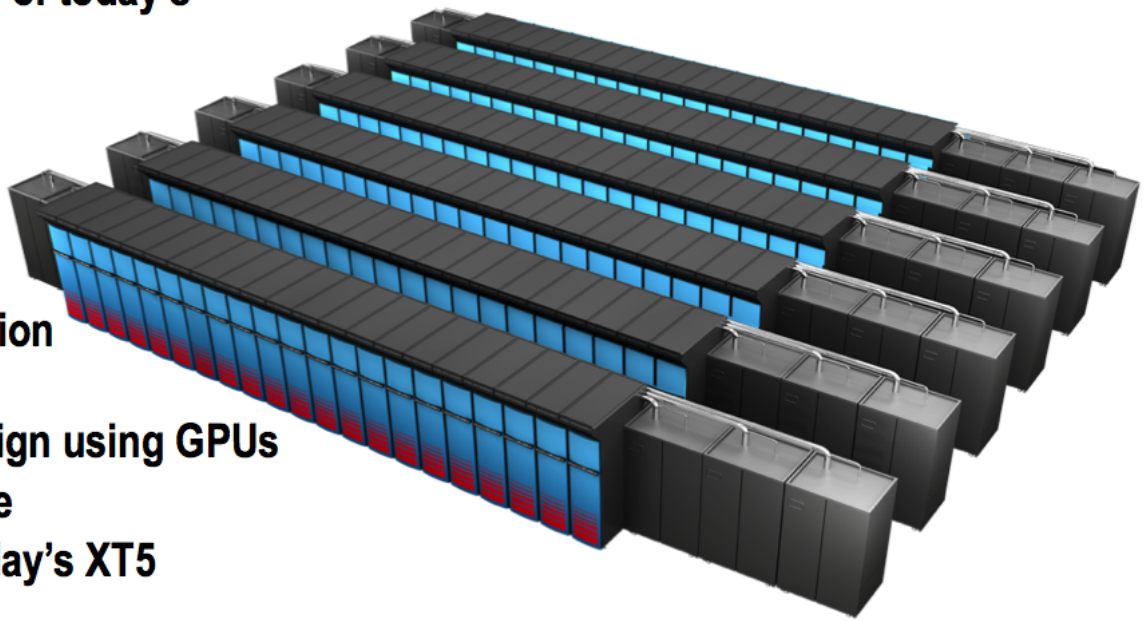
- Motivation
- What's been done
- Where we might go in the next 1-3 years

Motivation: Big machines on the way...

ORNL's "Titan" 20 PF System Goals

~18,000 GPUs

- Designed for science from the ground up
- Operating system upgrade of today's Linux Operating System
- Gemini interconnect
  - 3-D Torus
  - Globally addressable memory
  - Advanced synchronization features
- New accelerated node design using GPUs
- 10-20 PF peak performance
  - 9x performance of today's XT5
- Larger memory
- 3x larger and 4x faster file system



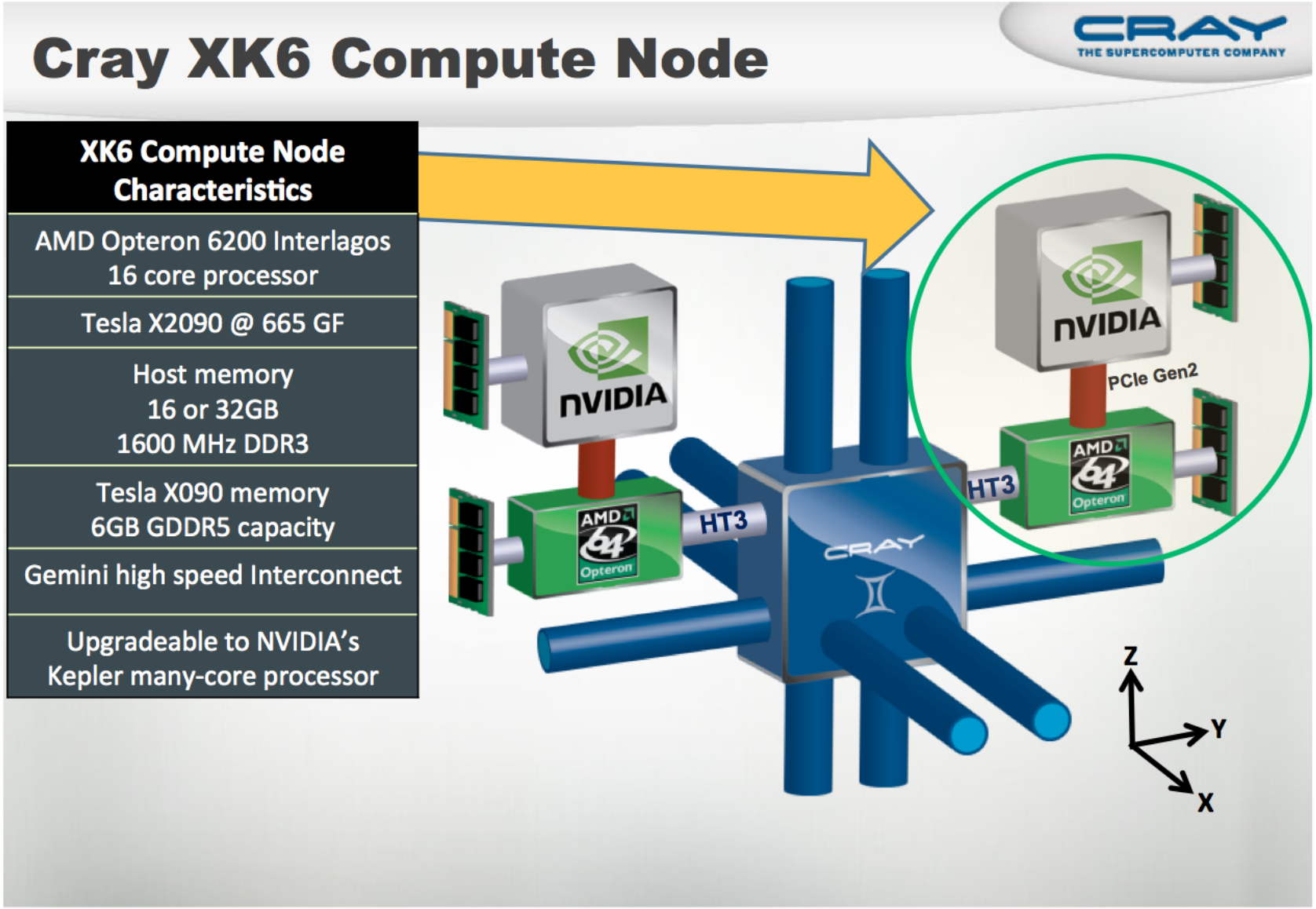
8 OLCF

Disclaimer: No contract with vendor is in place

OAK RIDGE National Laboratory

Source: http://www.olcf.ornl.gov/wp-content/uploads/2011/07/TitanWebinar.pdf

Motivation: Big machines on the way...



Slide courtesy of Cray, Inc.

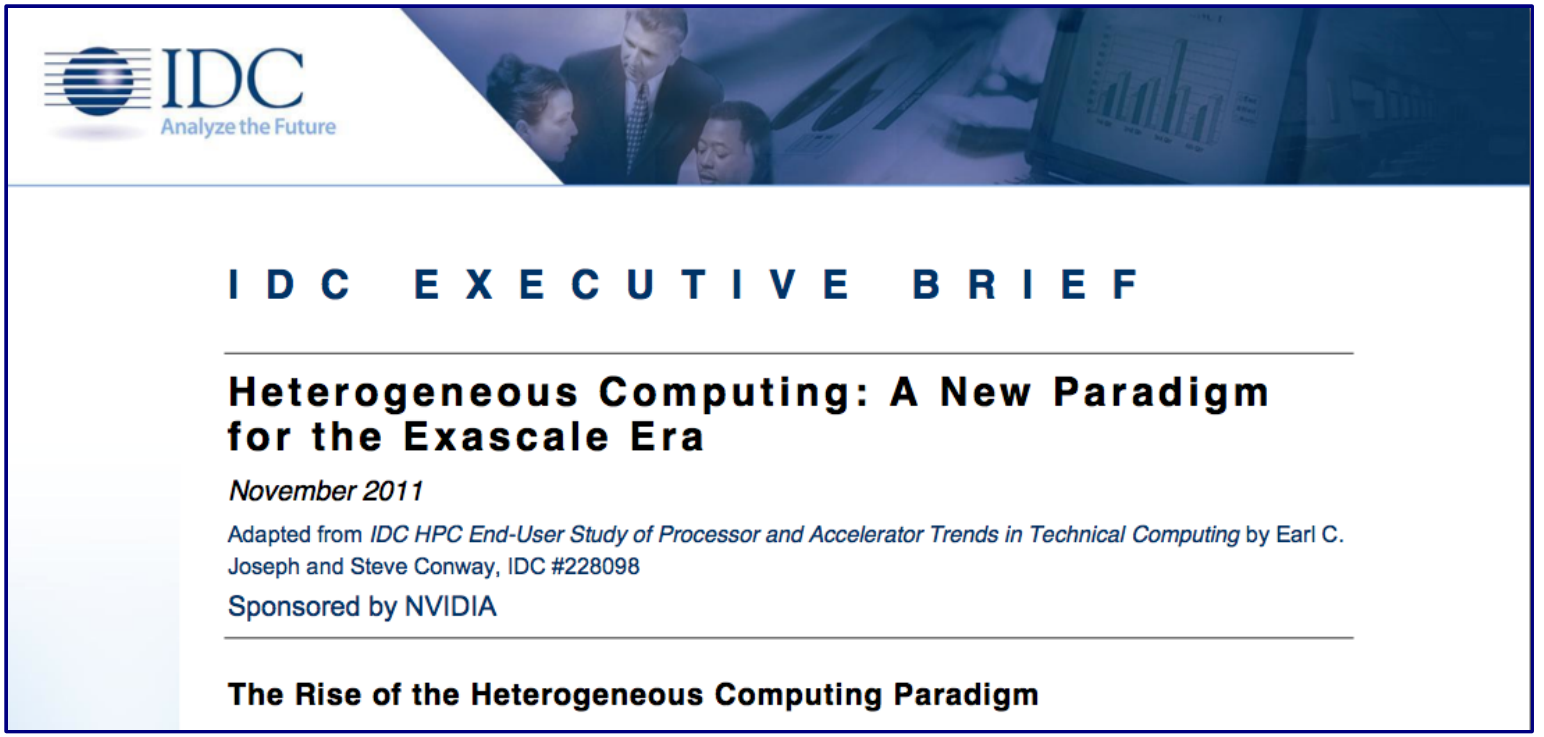
Disclaimer: No contract with vendor is in place

GPUs on the Top 500

- In the latest “Top 500” list (June 2011), three of the top five machines feature GPUs.

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	548352	8162.00	8773.63	9898.56
2	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
3	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
4	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
5	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.61

The future is heterogeneous


IDC
Analyze the Future

IDC EXECUTIVE BRIEF

Heterogeneous Computing: A New Paradigm for the Exascale Era

November 2011

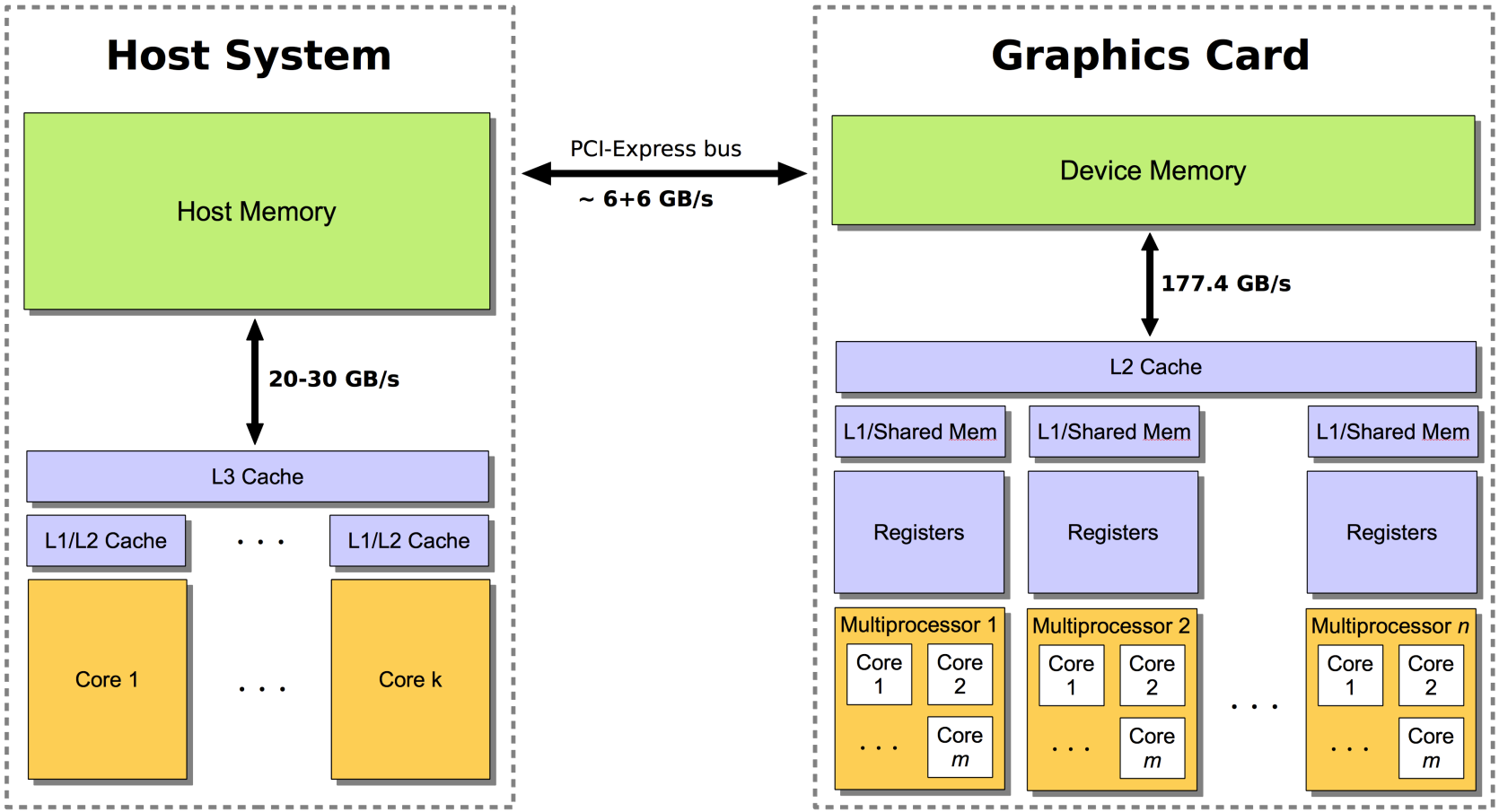
Adapted from *IDC HPC End-User Study of Processor and Accelerator Trends in Technical Computing* by Earl C. Joseph and Steve Conway, IDC #228098

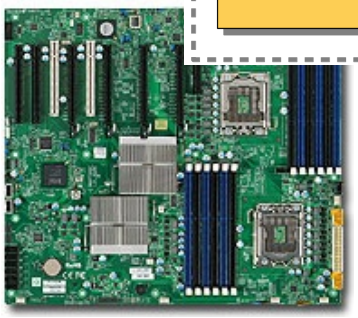
Sponsored by NVIDIA

The Rise of the Heterogeneous Computing Paradigm

- Driven by power-efficiency considerations: Use the right processor for the right workload (latency vs. throughput).
- GPU clusters are a harbinger of things to come and thus a key algorithmic test bed: high compute density, constrained by memory and interconnect bandwidth.

GPU memory hierarchy





(GeForce GTX 480)

QUDA Overview

- "QCD on CUDA" – developed in CUDA C/C++.
- Provides optimized solvers and other routines for the following fermion actions:
    - Wilson and clover-improved Wilson
    - Twisted mass
    - Improved staggered (asqtad/HISQ)
    - Domain wall
- Details, mailing list, and source code repository available here: http://lattice.github.com/quda
- Chroma can be built to use the Wilson/clover code with a simple configure flag, likewise for MILC and asqtad/HISQ.
- Straightforward to call directly (e.g., alongside QDP/C)

QUDA Capabilities

- **CG** and **BiCGstab** solvers for all actions.
- Wilson, clover, twisted-mass, and improved staggered code includes:
    - **Multi-GPU** support, using either MPI or QMP for communication.
    - **Multi-shift CG** solver.
    - **Domain-decomposed GCR** solver (not proven yet for staggered).
- Improved staggered code also includes:
    - Asqtad link fattening (HISQ in progress).
    - Asqtad fermion force (HISQ in progress).
    - Gauge force for 1-loop improved Symanzik action.

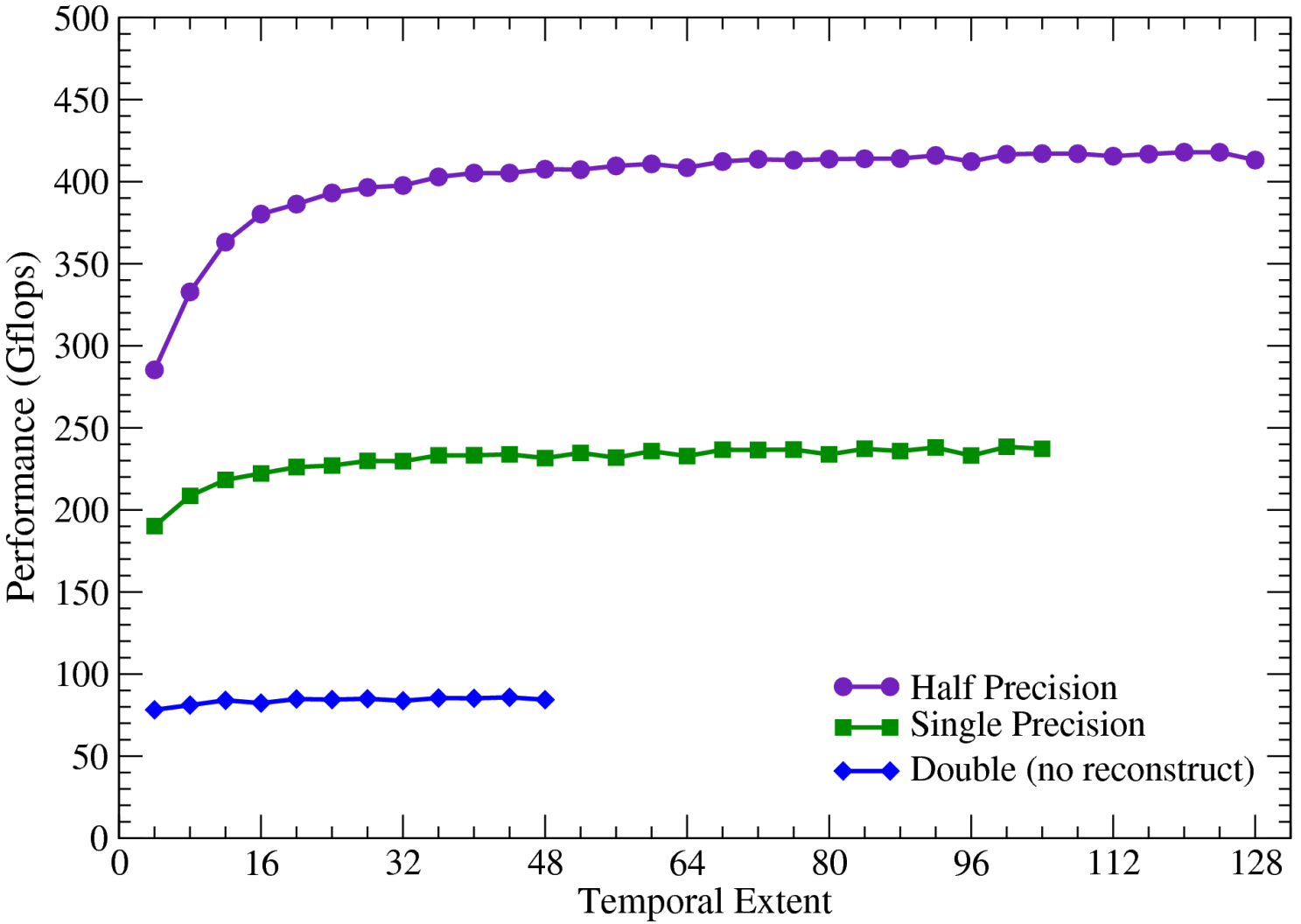
Performance results

- Results are for the even/odd preconditioned clover-improved Wilson matrix-vector product,

$$M = (1 - A_{ee}^{-1} D_{eo} A_{oo}^{-1} D_{oe})$$

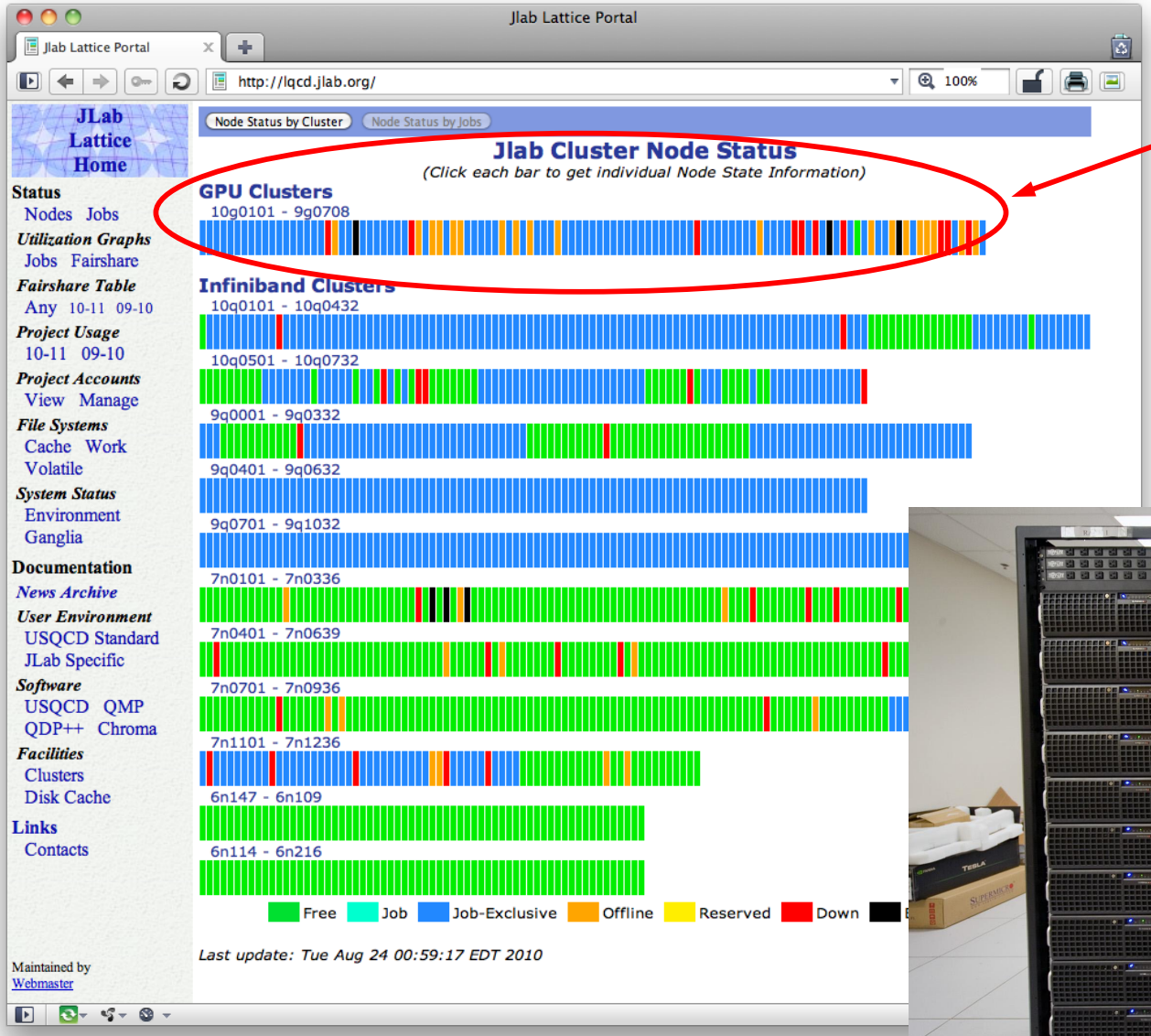
- Runs were done on a single GeForce GTX 480.
- For reference, a standard dual-socket node with recent (Westmere) quad-core Xeons would sustain around **20 Gflops** in single precision for a (fairly) well-optimized Wilson-clover Dslash. *2x improvement perhaps possible with better cache blocking.*
- We'll compare results for double, single, and half precision. In this case, half is a 16-bit quasi-fixed-point implementation, implemented via normalized texture reads.
- The spatial volume is held fixed at 24^3.

Matrix-vector performance

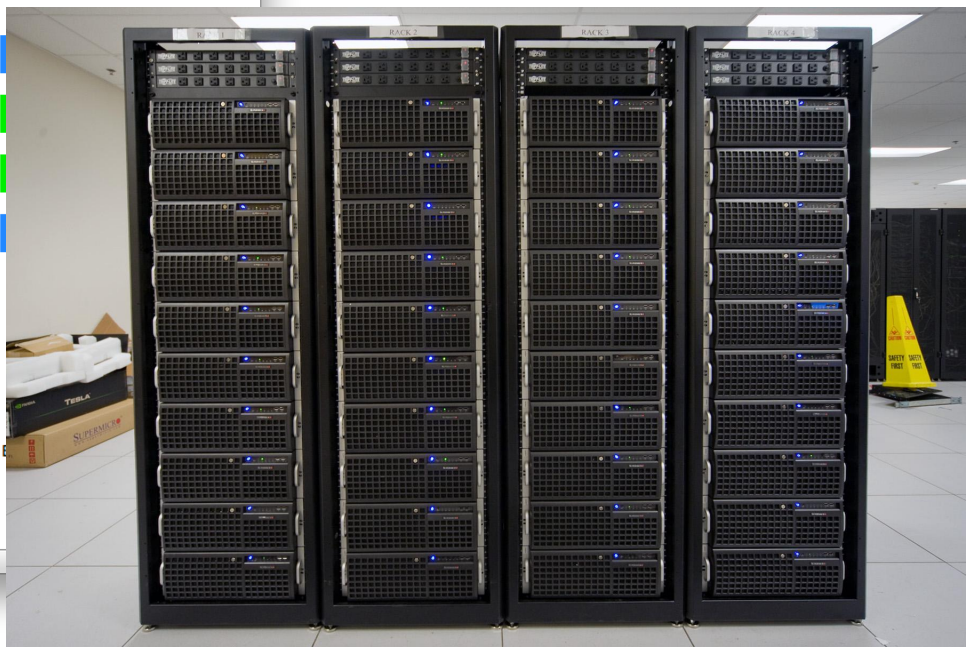


- Single and half performance are about 2.8x and 4.9x higher than double, respectively.

GPUs are in serious use for “analysis”



~ 500 GPUs dedicated to LQCD at Jefferson Lab



Can they also make a dent here?

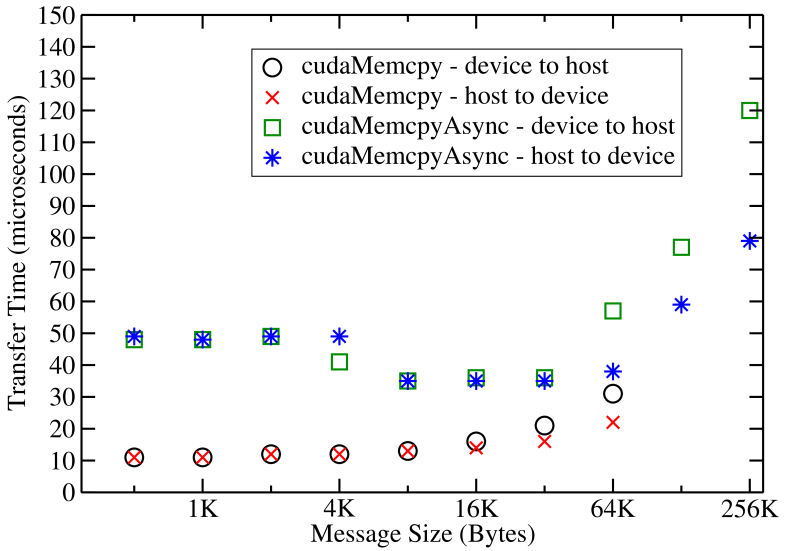
“Intrepid” - Argonne Leadership Computing Facility

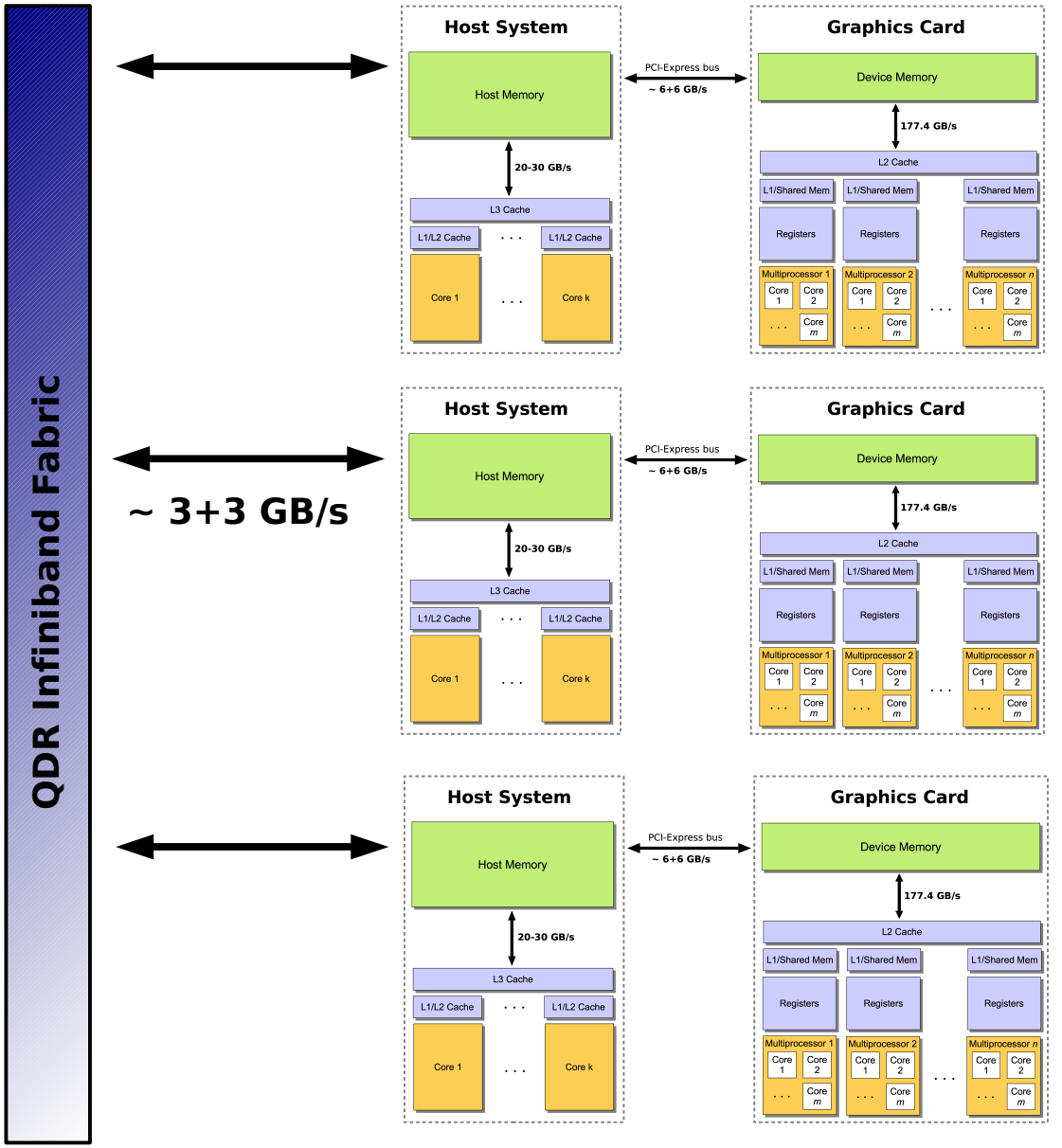
QPACE – NIC Juelich

“Jaguar” - Oak Ridge Leadership Computing Facility

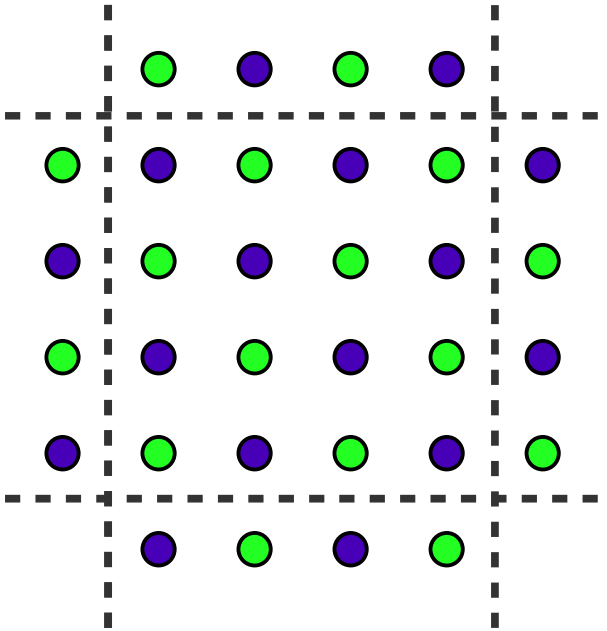
Challenges to scaling up

- GPU-to-host and inter-node **bandwidth**
- GPU-to-host and inter-node **latency**



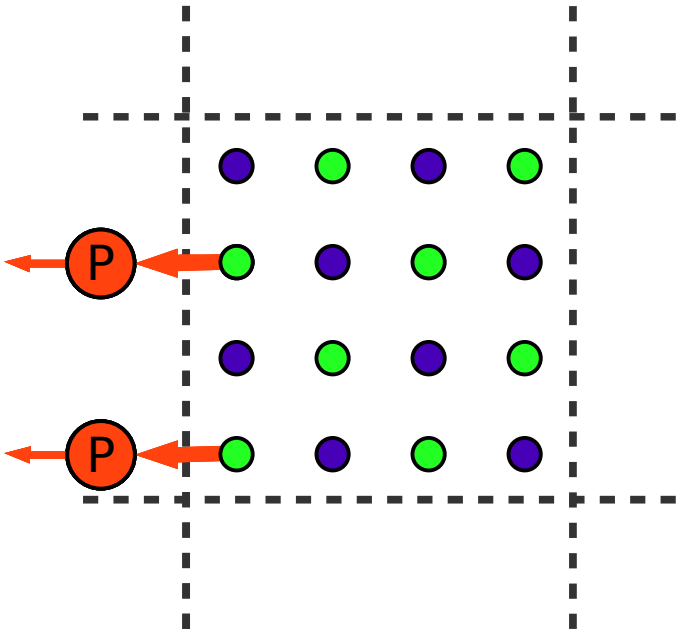


Parallelizing the Dslash



- For illustration, consider a 2D problem with a 4^2 local volume.
- Because we employ even/odd (red/black) preconditioning, only half the sites will be updated per “Dslash” operation.
- We'll take these to be the purple sites.

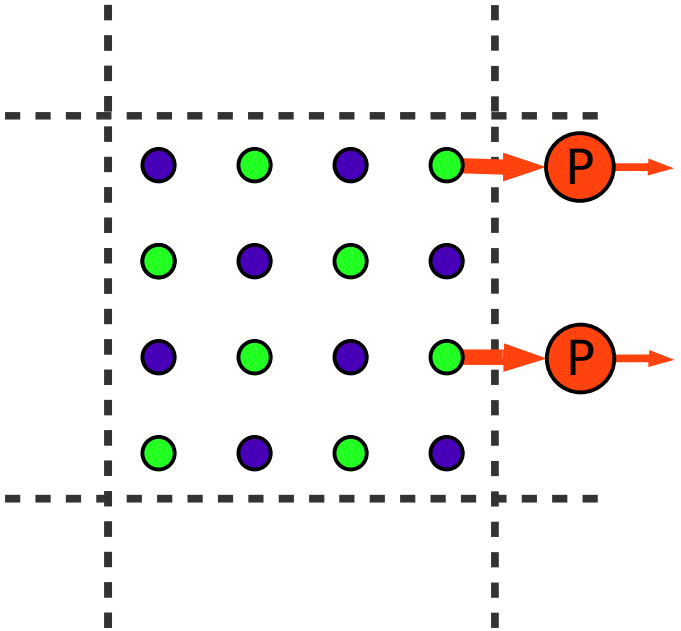
Parallelizing the Dslash



Step 1:

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.
- As part of the gather kernel, a “spin projection” step reduces the amount of data that must be transferred from 24 to 12 floats, at the cost of only 12 adds.

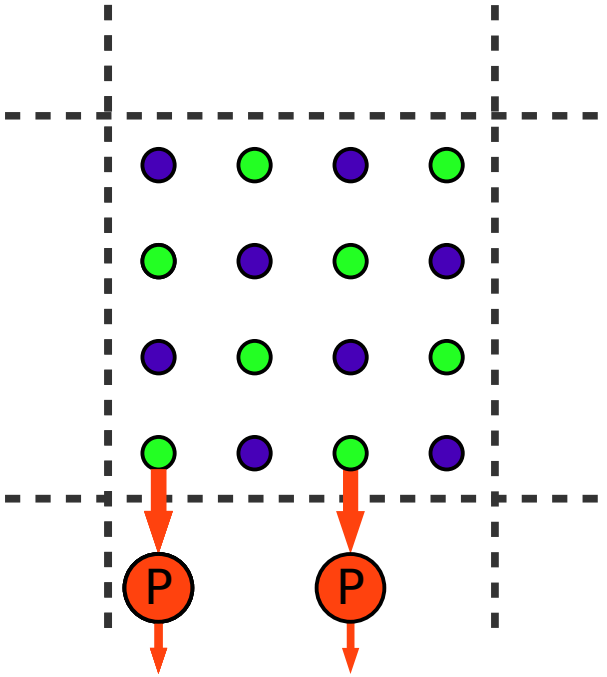
Parallelizing the Dslash



Step 1:

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.
- As part of the gather kernel, a “spin projection” step reduces the amount of data that must be transferred from 24 to 12 floats, at the cost of only 12 adds.

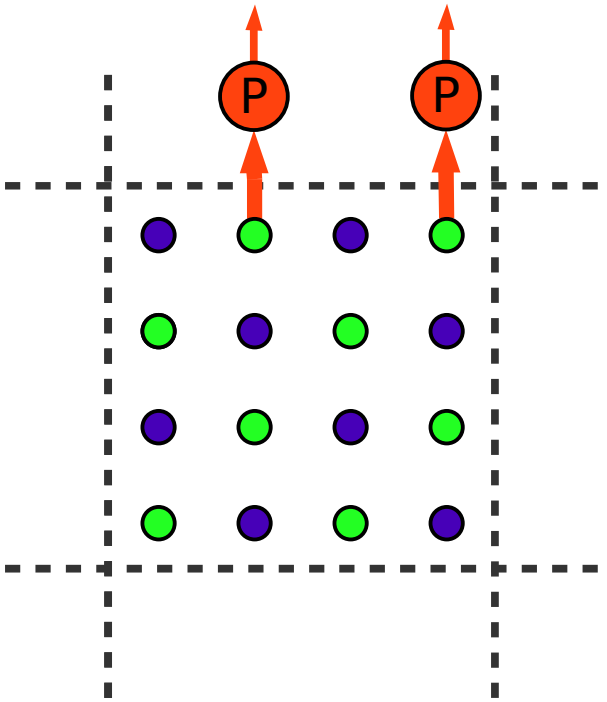
Parallelizing the Dslash



Step 1:

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.
- As part of the gather kernel, a “spin projection” step reduces the amount of data that must be transferred from 24 to 12 floats, at the cost of only 12 adds.

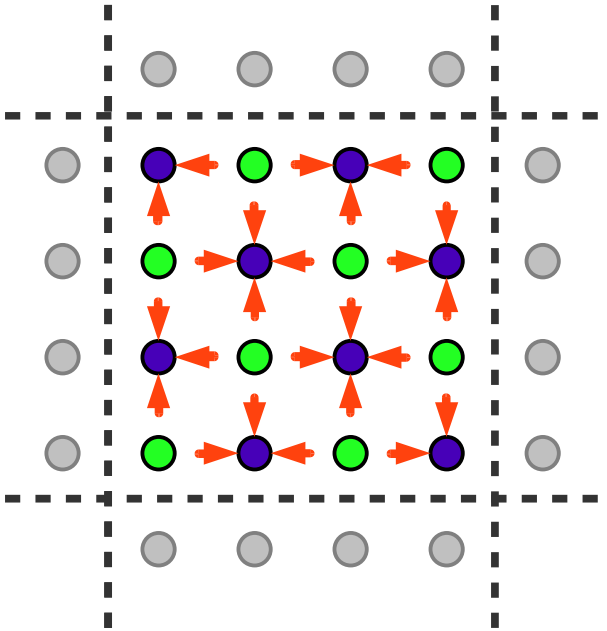
Parallelizing the Dslash



Step 1:

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.
- As part of the gather kernel, a “spin projection” step reduces the amount of data that must be transferred from 24 to 12 floats, at the cost of only 12 adds.

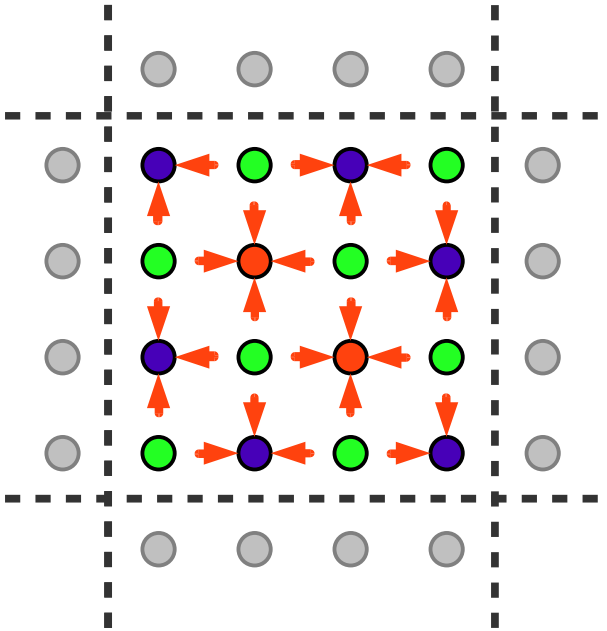
Parallelizing the Dslash



Step 2:

- An “interior kernel” updates all local sites to the extent possible. Sites along the boundary receive contributions from local neighbors.

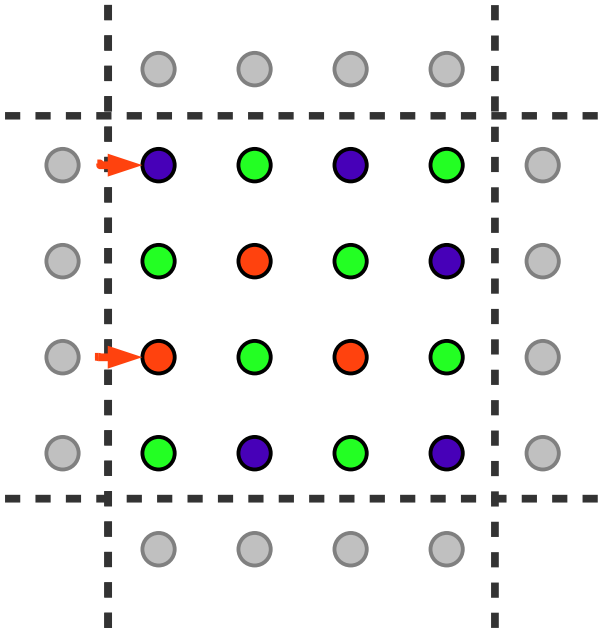
Parallelizing the Dslash



Step 2:

- An “interior kernel” updates all local sites to the extent possible. Sites along the boundary receive contributions from local neighbors.
- To finish off a site, we must apply the clover term (local 12x12 complex matrix-vector multiply). This is done for a given site once contributions from all neighbors have been accumulated.

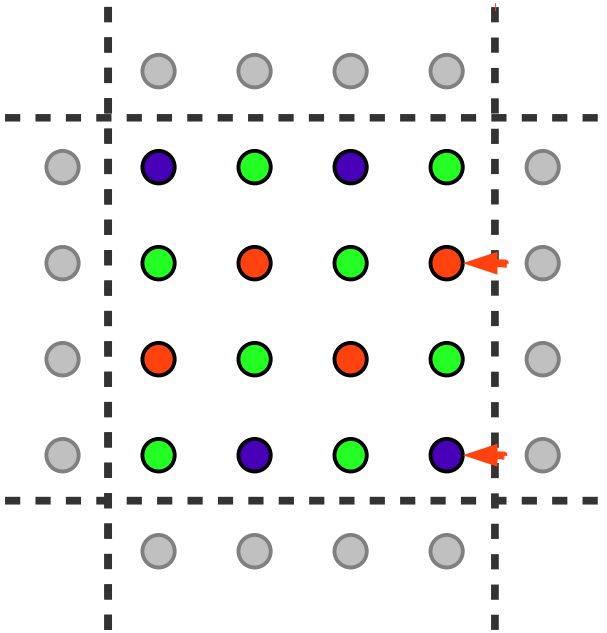
Parallelizing the Dslash



Step 3:

- Boundary sites are updated by a series of kernels (one per direction).
- Note that corner sites (and edges/faces in higher dimensions) introduce a data dependency between kernels, which must therefore execute sequentially.
- A given boundary kernel must also wait for it's “ghost zone” to arrive.

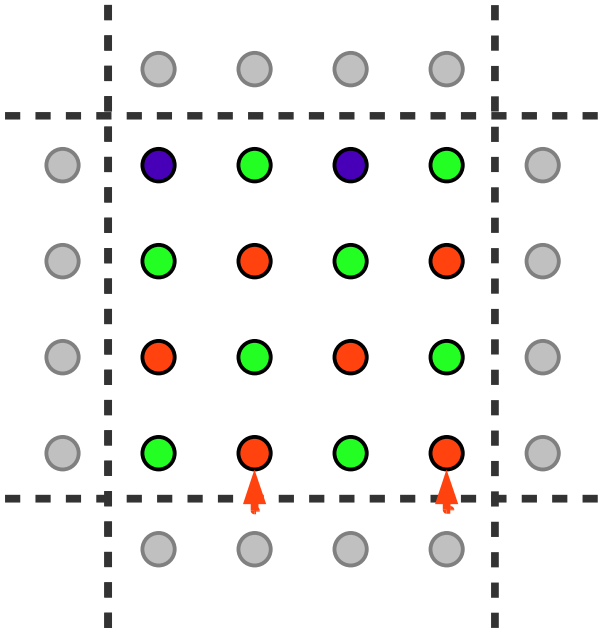
Parallelizing the Dslash



Step 3:

- Boundary sites are updated by a series of kernels (one per direction).
- Note that corner sites (and edges/faces in higher dimensions) introduce a data dependency between kernels, which must therefore execute sequentially.
- A given boundary kernel must also wait for it's “ghost zone” to arrive.

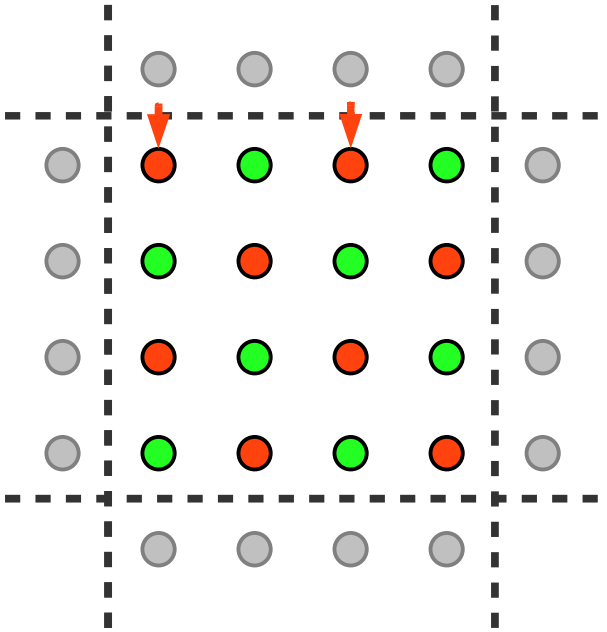
Parallelizing the Dslash



Step 3:

- Boundary sites are updated by a series of kernels (one per direction).
- Note that corner sites (and edges/faces in higher dimensions) introduce a data dependency between kernels, which must therefore execute sequentially.
- A given boundary kernel must also wait for it's “ghost zone” to arrive.

Parallelizing the Dslash

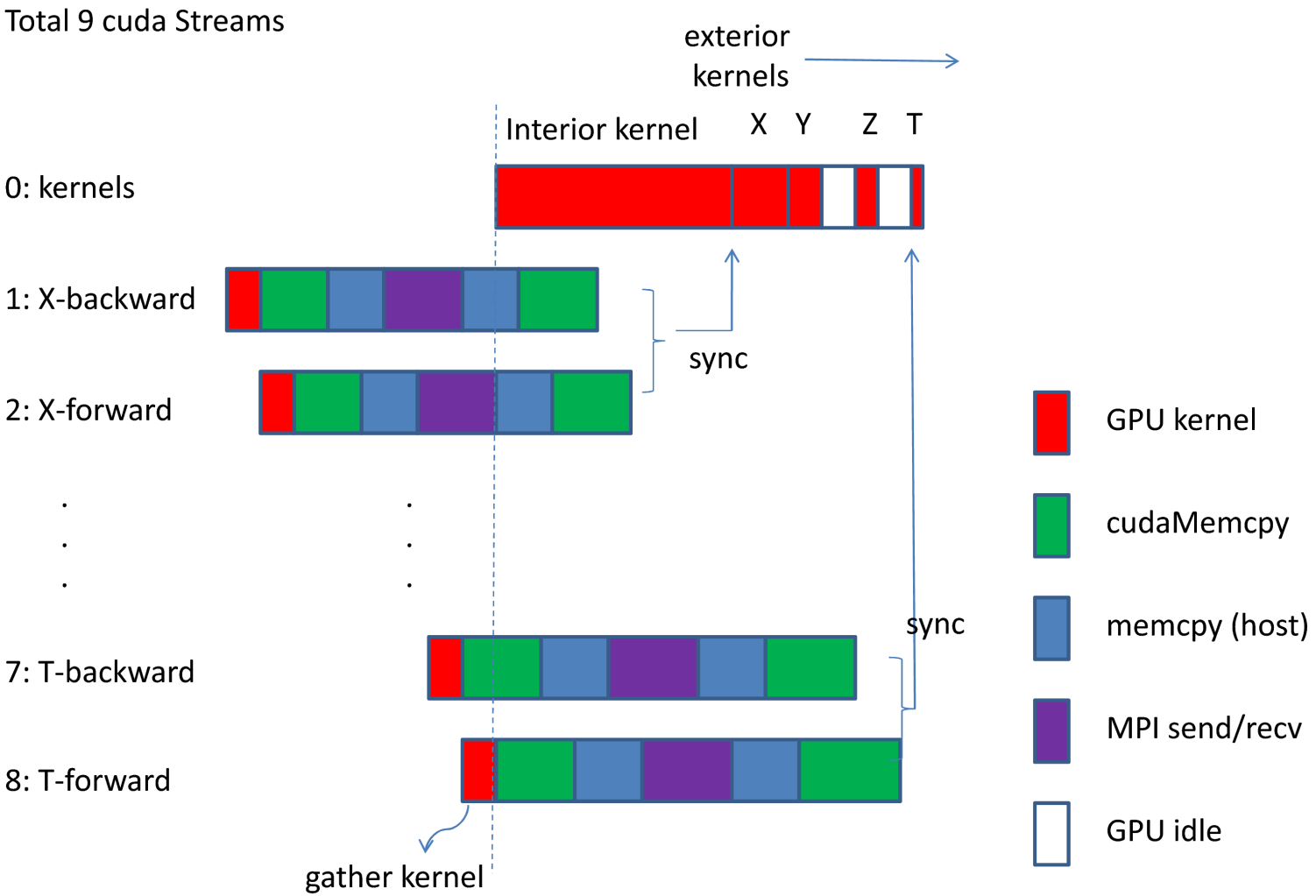


Done!

Step 3:

- Boundary sites are updated by a series of kernels (one per direction).
- Note that corner sites (and edges/faces in higher dimensions) introduce a data dependency between kernels, which must therefore execute sequentially.
- A given boundary kernel must also wait for it's “ghost zone” to arrive.

Overlapping communications

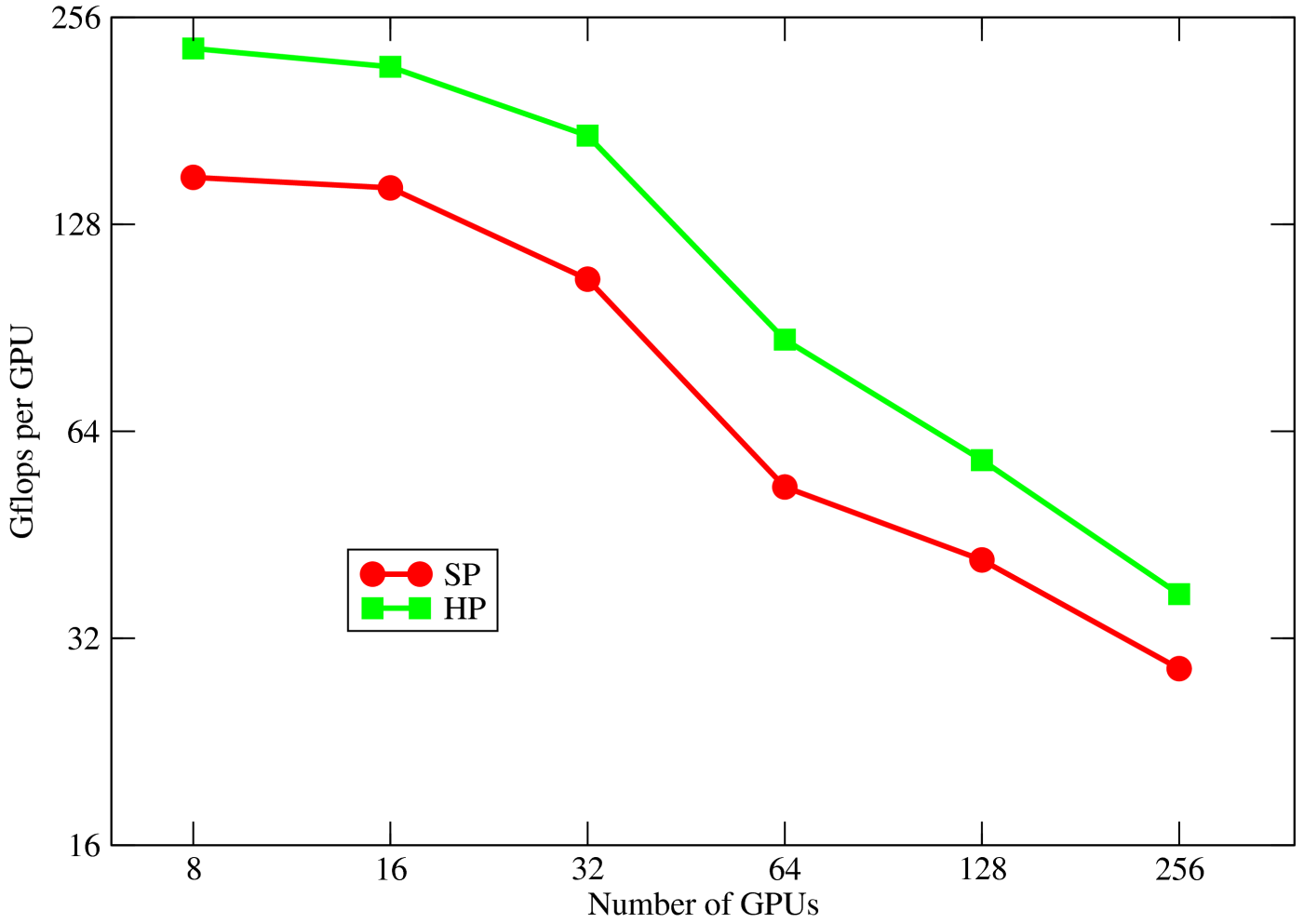


Hardware

- Out test bed is the “Edge” cluster at Lawrence Livermore National Lab:
  - 206 nodes available for batch jobs, interconnected by QDR infiniband
  - 2 Intel Xeon X5660 processors per node (6-core Westmere @ 2.8 GHz)
  - 2 Tesla M2050 cards per node, sharing 16 PCI-E lanes to the IOH via a switch
  - ECC enabled on the Teslas
  - CUDA 4.0 RC1 (but no GPU-Direct)
  - Driver version 270.27
  - Pre-release version of QUDA 0.4, interfaced to Chroma (an application suite for lattice QCD).

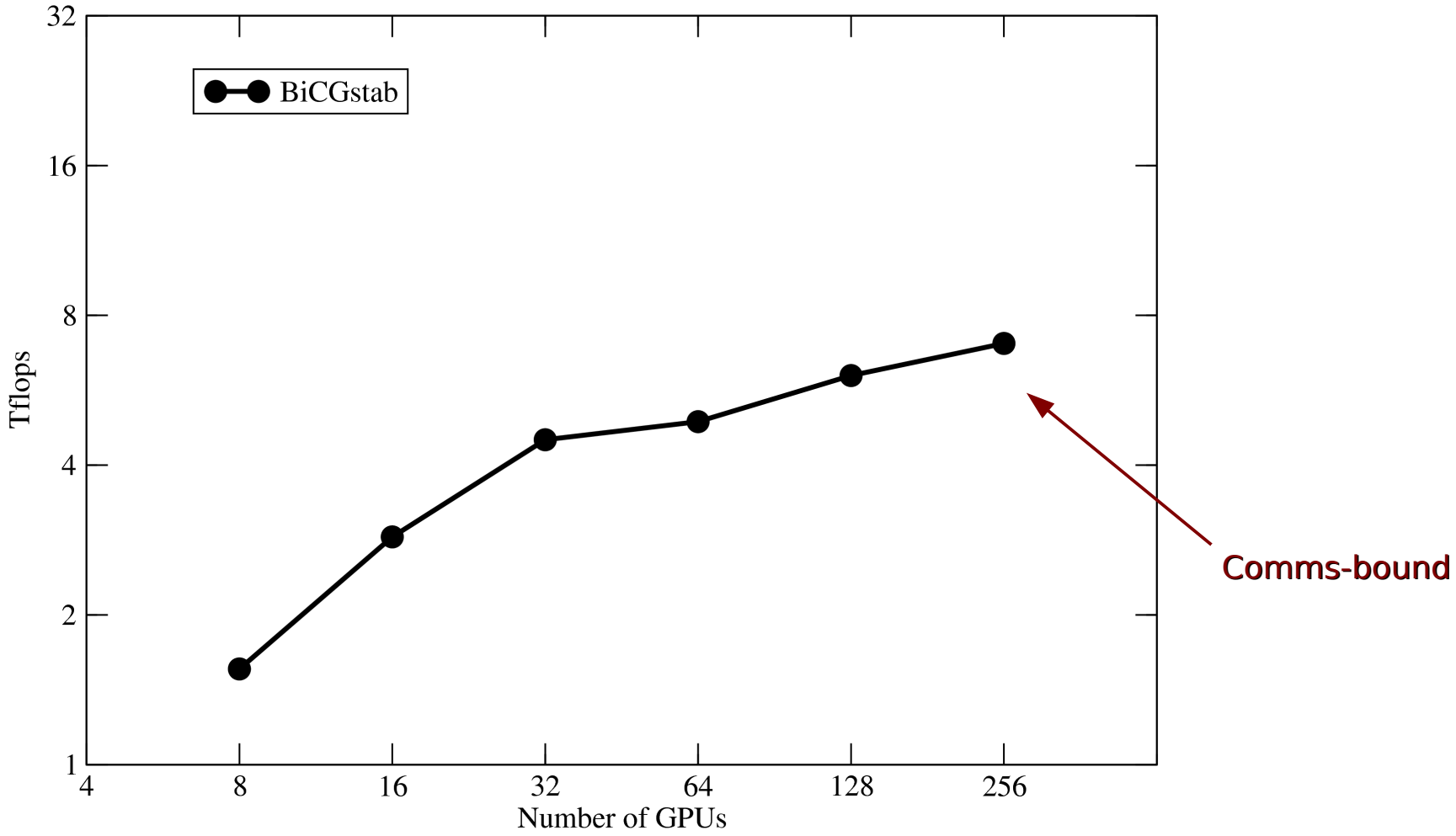
Matrix-vector performance results

$V = 32^3 \times 256$



Solver performance

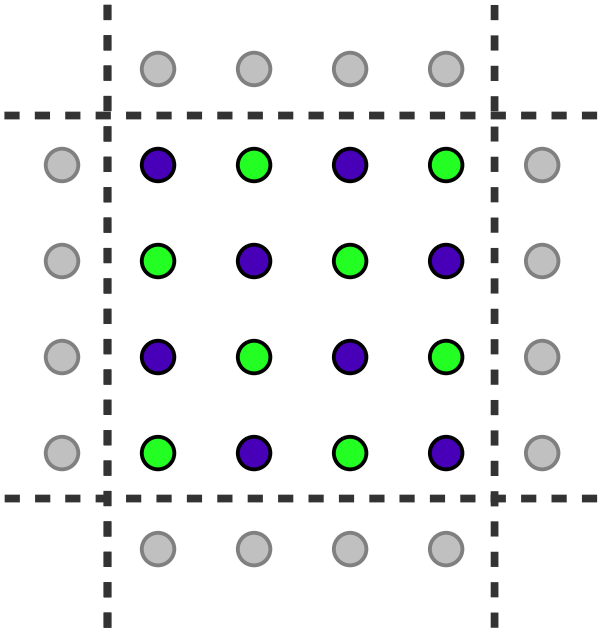
$V = 32^3 \times 256$



(BiCGstab, mixed single/half with reliable updates)

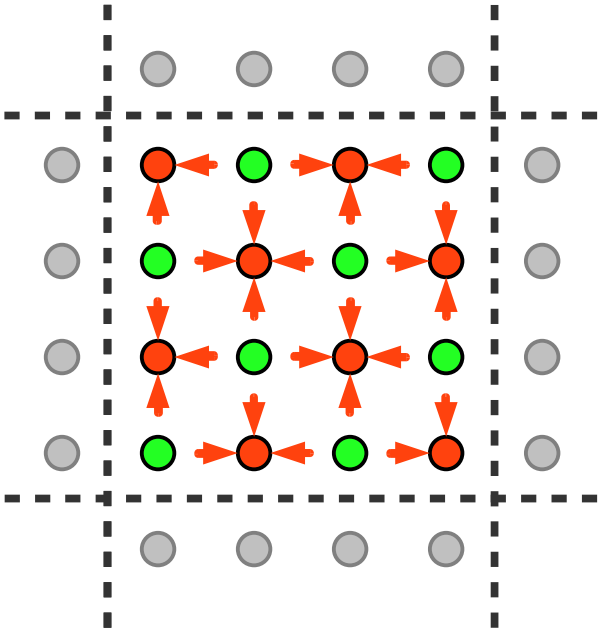
Building a scalable solver

- We need a smarter algorithm, one that takes advantage of the ample compute throughput available while minimizing communication.
- This led us to adopt a domain-decomposition approach by applying an additive Schwarz preconditioner to GCR.
- Most of the work is in the preconditioner, which solves a linear system (to low accuracy via MR) but with Dirichlet boundary conditions between GPUs. In other words, communication is simply turned off in the Dslash.
- Furthermore, this task is well-suited to reduced (e.g., half) precision.



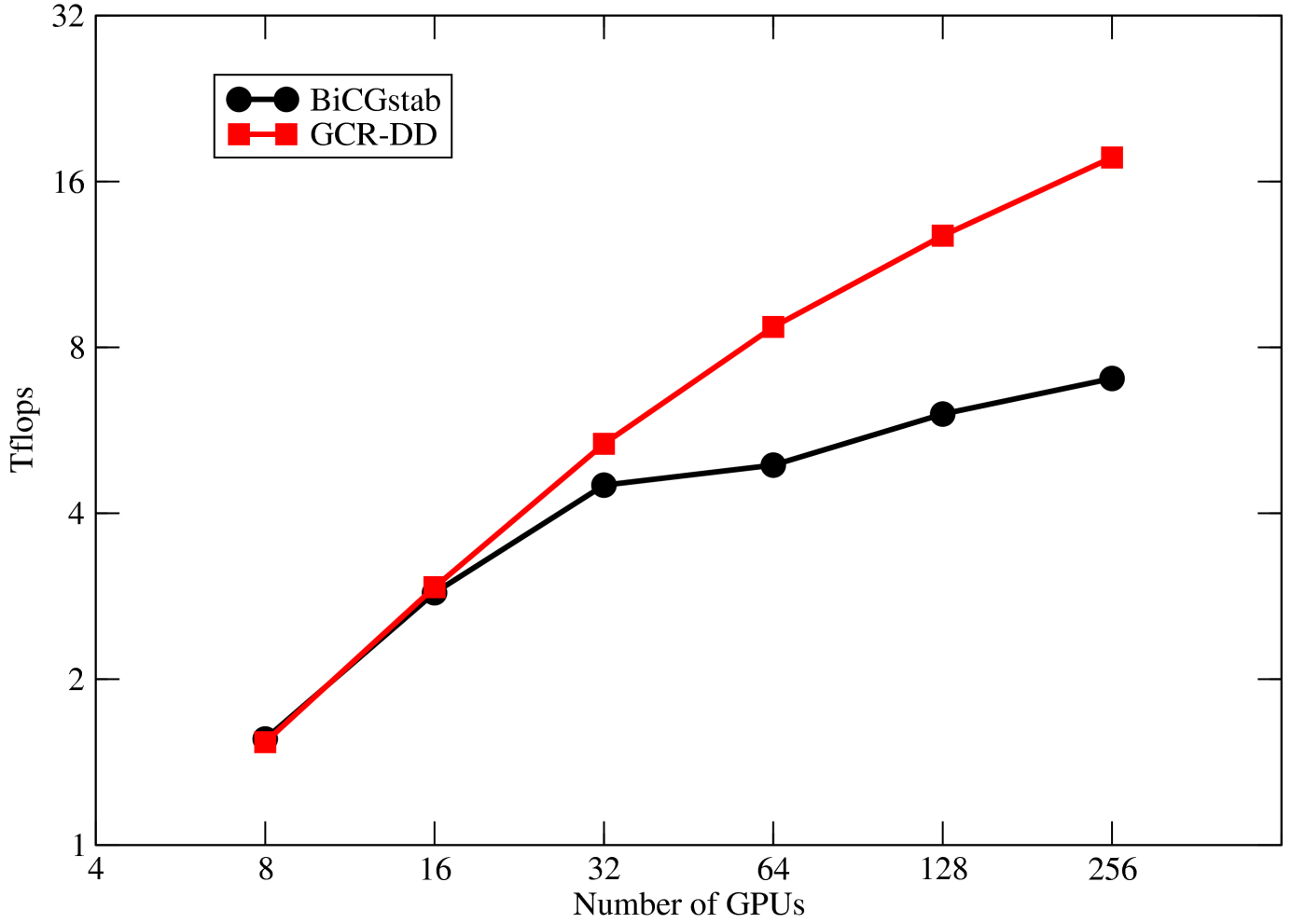
Building a scalable solver

- We need a smarter algorithm, one that takes advantage of the ample compute throughput available while minimizing communication.
- This led us to adopt a domain-decomposition approach by applying an additive Schwarz preconditioner to GCR.
- Most of the work is in the preconditioner, which solves a linear system (to low accuracy via MR) but with Dirichlet boundary conditions between GPUs. In other words, communication is simply turned off in the Dslash.
- Furthermore, this task is well-suited to reduced (e.g., half) precision.



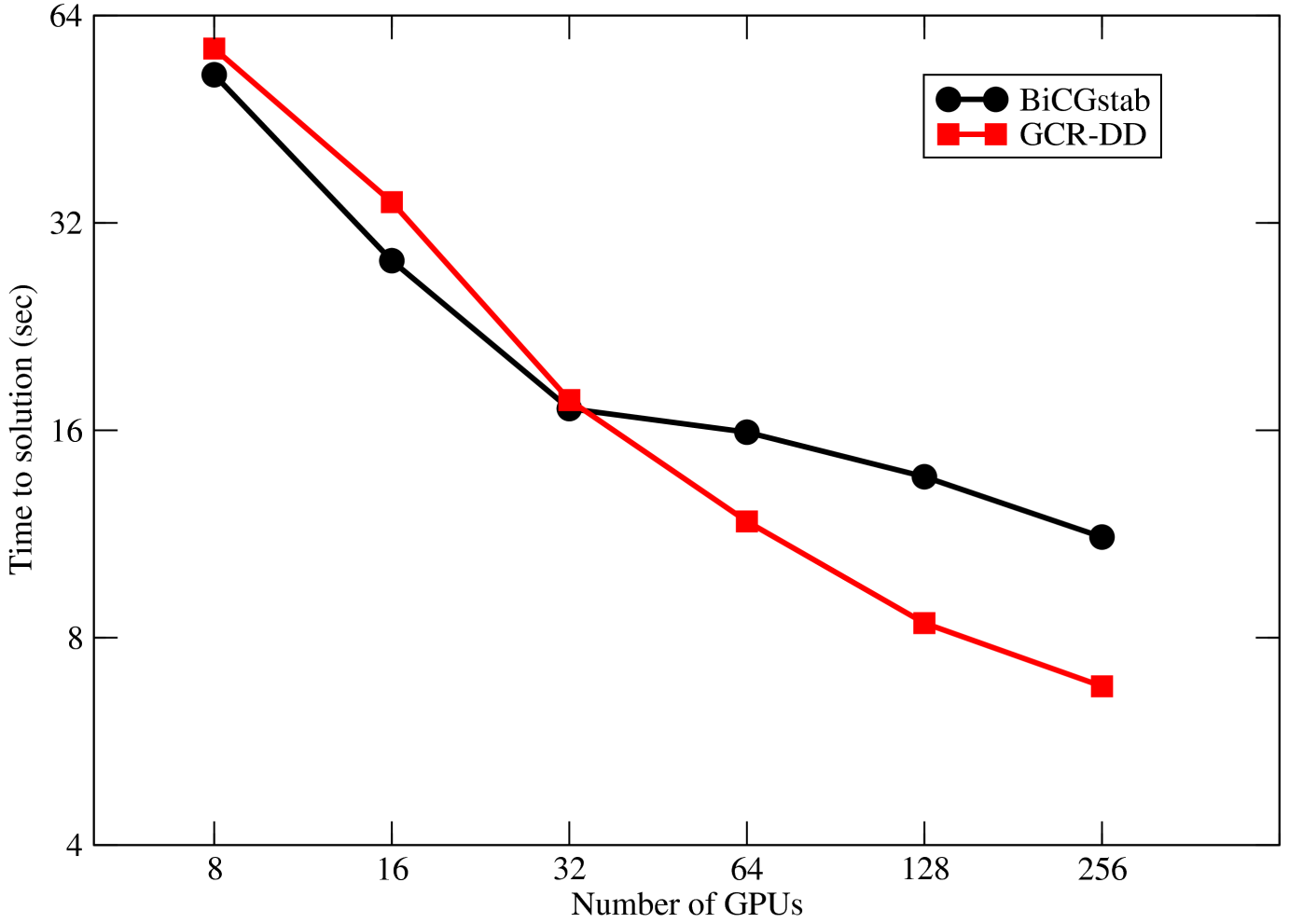
Solver performance

$V = 32^3 \times 256$



Solver time to solution

$V = 32^3 \times 256$

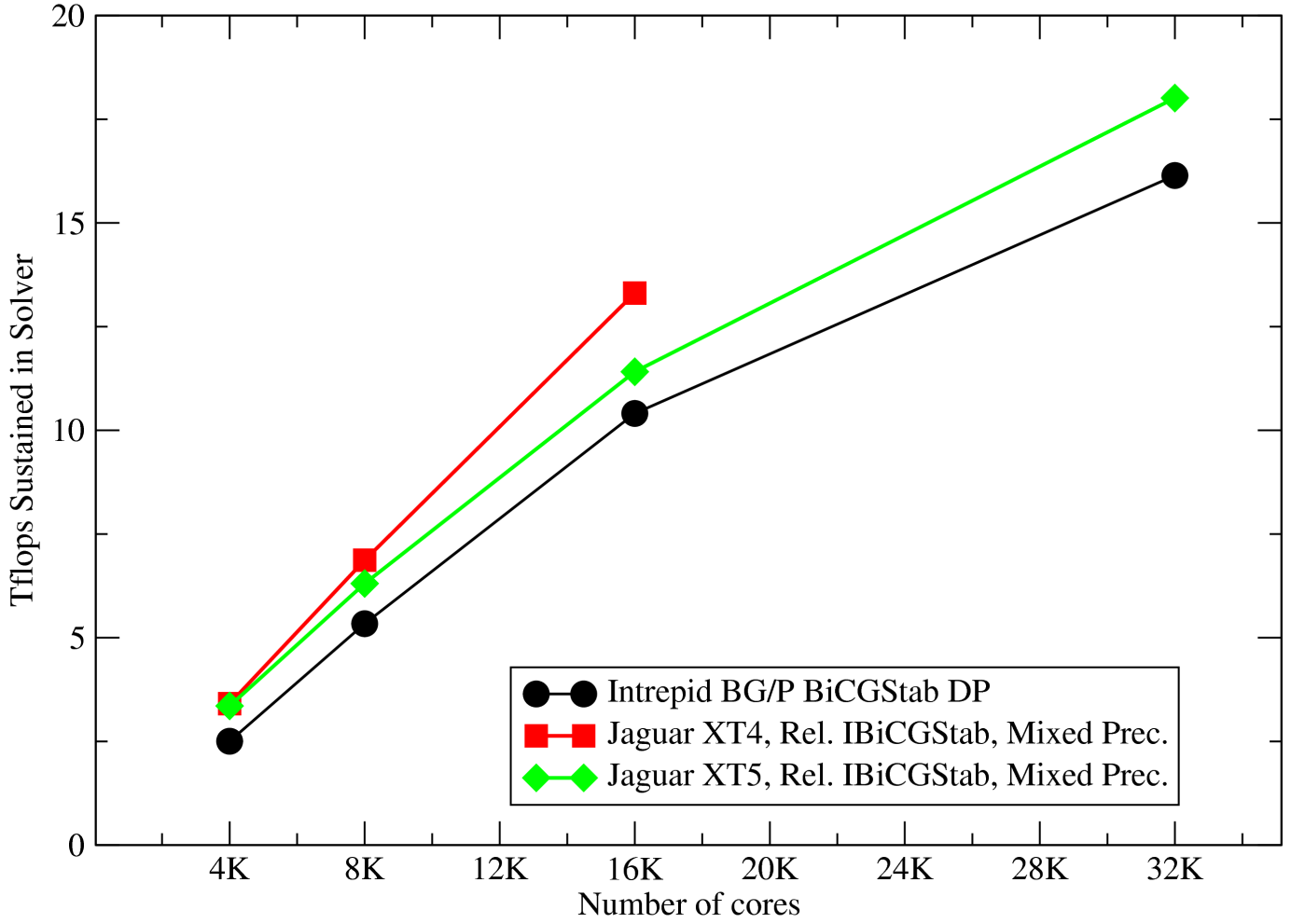


Comparisons

- For a fair comparison, time to solution is the relevant quantity, since algorithms differ in the total number of operations required to reach a given level of accuracy.
- We can define an “effective Tflops” number for GCR-DD, however, in terms of the level of performance that pure single-precision BiCGstab would have to achieve to obtain the same time to solution.
- This yields an effective **9.95 Tflops** for GCR-DD **on 128 GPUs** and **11.5 Tflops on 256 GPUs**.
- This allows us to make rough comparisons to comparable runs on various capability machines:
    - Cray XT4 (Jaguar at Oak Ridge LCF)
    - Cray XT5 (Jaguar PF at Oak Ridge LCF)
    - BlueGene/P (Intrepid at Argonne LCF)

Comparisons

$V = 32^3 \times 256$

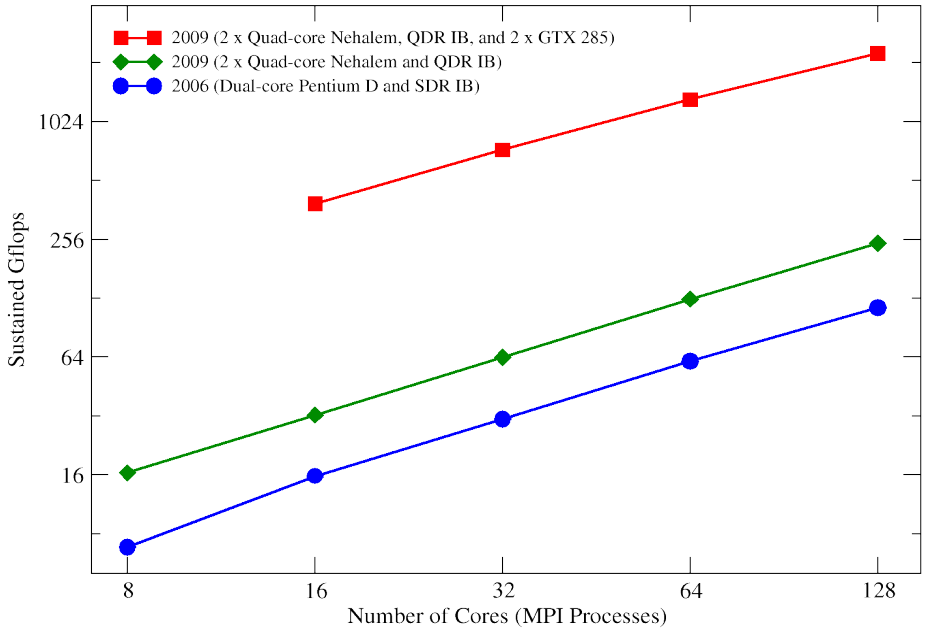


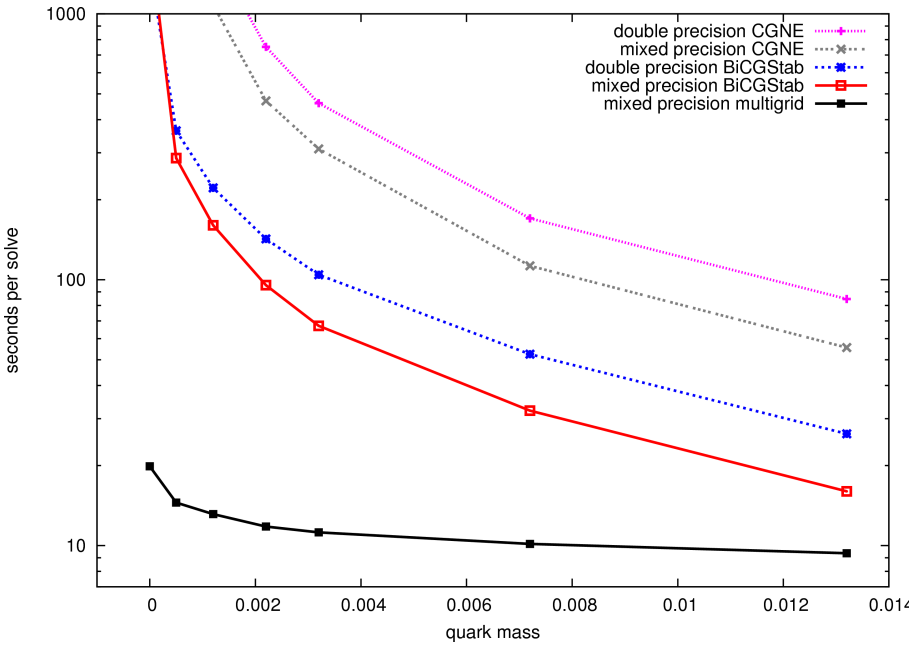
Much to do (no particular order)

- Efficient multi-GPU solvers for all actions (especially DWF)
- Gauge generation with all critical kernels offloaded
    - Work underway for improved staggered (MILC+QUDA, eventually FUEL)
- Strong-scaling HMC to O(1000) GPUs or more
    - Will require serious algorithmic research (low-hanging fruit: overlapping blocks, multiplicative Schwartz)
- BSM: Additional gauge groups & representations
- On-going optimization for new architectures as they emerge
- Going beyond hand-coding (see Bálint's talk)
- Multi-GPU multigrid (Wilson/clover initially)

Outlook: Multigrid on GPUs

- GPUs clearly win for many workloads in lattice QCD (5-10x improvement in price/performance)
- . . . but multigrid on traditional clusters is competitive (up to 20x over standard solvers at light masses).
- Next step: **MG²:** Multi-GPU multigrid (up to 100x **?**).





Multi-GPU x Multigrid = ?

NVIDIA points of contact

- Mike Clark – currently >50% on QCD
- “NVAMG” – Effort within Jonathan Cohen's “Emerging Applications” group, partly inspired by HYPRE
- Ron – Evaluating/influencing future architectures, some continued work on QUDA
- Many interested in issues of programmability and performance characterization.

Bonus slides

References

- K. Barros, R. Babich, R. Brower, M. A. Clark, and C. Rebbi, “Blasting through lattice calculations using CUDA,” PoS(LATTICE2008) 045 (2008) [arXiv:0810.5365 [hep-lat]].
- M. A. Clark, R. Babich, K. Barros, R. Brower, and C. Rebbi, "Solving Lattice QCD systems of equations using mixed precision solvers on GPUs," Comput. Phys. Commun. 181, 1517 (2010) [arXiv:0911.3191 [hep-lat]].
- R. Babich, M. A. Clark, and B. Joó, “Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice Quantum Chromodynamics," SC'10.
- G. Shi, S. Gottlieb, A. Torok, and V. Kindratenko, "Accelerating Quantum Chromodynamics Calculations with GPUs," proceedings of SAAHPC'10. [*generalization of the above to staggered fermions*]
- R. Babich, M. A. Clark, B. Joó, G. Shi, R. Brower, and S. Gottlieb, “Scaling lattice QCD beyond 100 GPUs,” SC'11.
- **Prior and contemporary work:** Wuppertal/Budapest group (G.I. Egri et al., 2007), National Taiwan U. (T.W. Chiu et al.), George Washington U. (A. Alexandru et al.), Seoul National U. (W. Lee et al.), Pisa U. (G. Cossu et al.), Hiroshima/Nagoya/KEK, others.

Tricks to reduce memory traffic

- Reconstruct SU(3) matrices from 8 or 12 real numbers on the fly, e.g.,

$$\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \qquad \mathbf{c} = (\mathbf{a} \times \mathbf{b})^*$$

P. De Forcrand, D. Lellouch and C. Roiesnel, "Optimizing a lattice QCD simulation program," J. Comput. Phys. 59, 324 (1985).

- Choose a gamma basis with γ_4 diagonal.

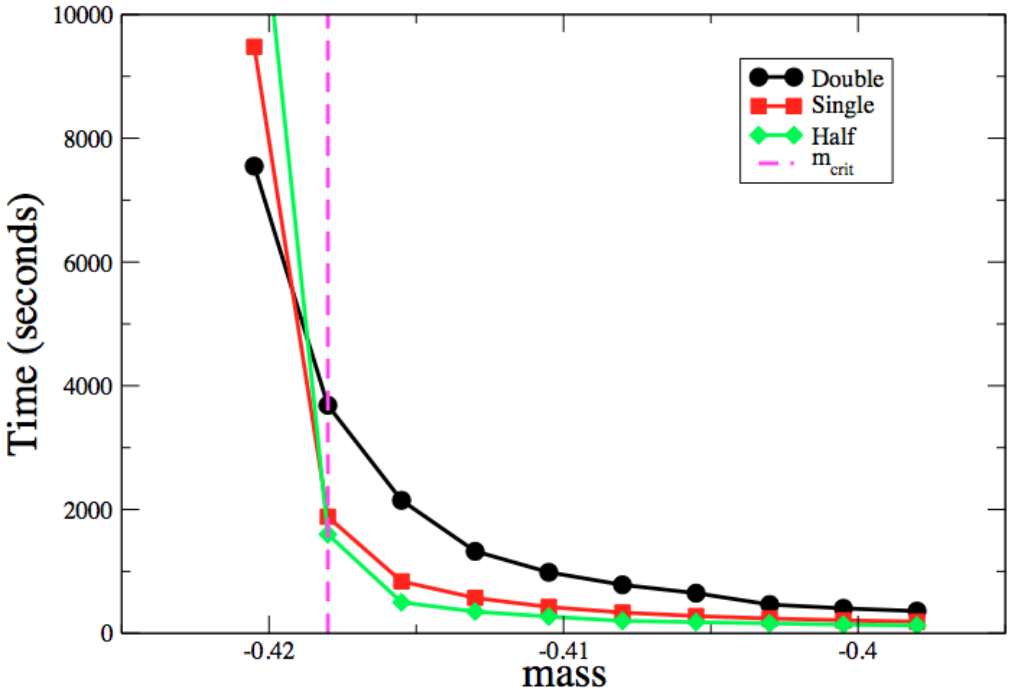
$$P^{\pm 4} = \begin{pmatrix} 1 & 0 & \pm 1 & 0 \\ 0 & 1 & 0 & \pm 1 \\ \pm 1 & 0 & 1 & 0 \\ 0 & \pm 1 & 0 & 1 \end{pmatrix} \Rightarrow \begin{cases} P^{+4} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ P^{-4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \end{cases}$$

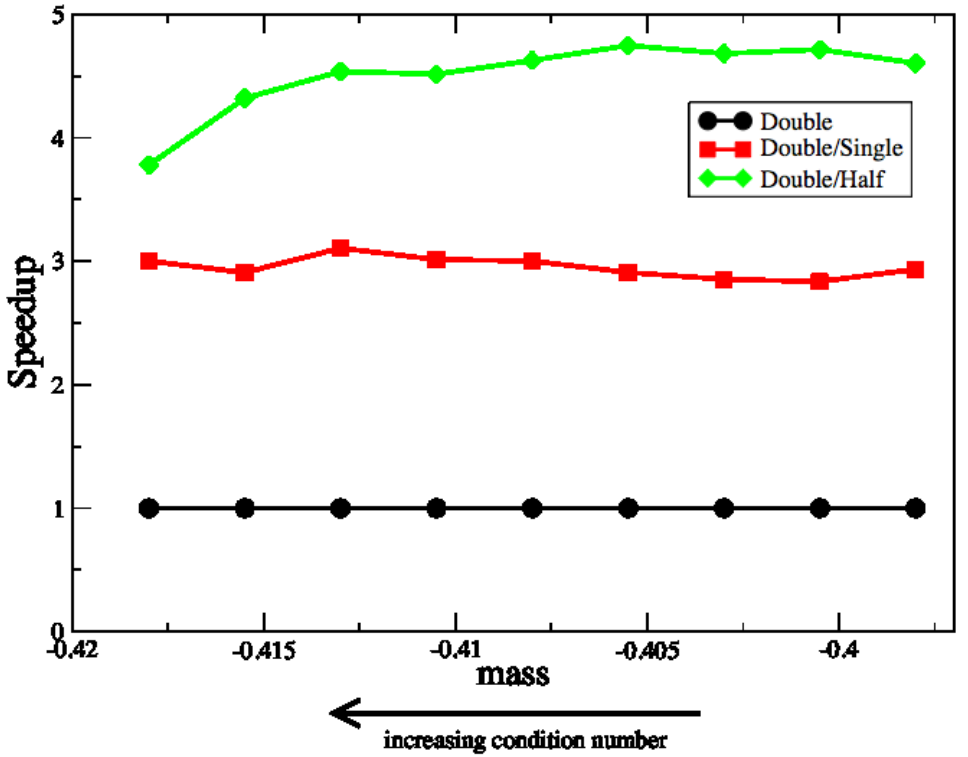
similarity transforms on *D*

- Fix to the temporal gauge (setting gauge links in the *t*-direction to the identity).

Mixed precision with reliable updates

- Using a mixed-precision solver incorporating “reliable updates” (Clark et al., arXiv:0911.3191) with half precision greatly reduces time-to-solution while maintaining double precision accuracy.





Mixed precision with reliable updates

- In the usual method of *iterative refinement* (or "*defect correction*"), the Krylov subspace is thrown away at every restart:

$$
\begin{array}{l}
r_0 = b - Ax_0; \\
k = 0; \\
\textbf{while } ||r_k|| > \epsilon \textbf{ do} \\
\quad \text{Solve } Ap_{k+1} = r_k \text{ to precision } \epsilon^{in}; \\
\quad x_{k+1} = x_k + p_{k+1}; \\
\quad r_{k+1} = b - Ax_{k+1} \ ; \\
\quad k = k + 1; \\
\textbf{end}
\end{array}
$$

- An alternative is "*reliable updates*," originally introduced to combat residual drift caused by the erratic convergence of BiCGstab: G. L. G. Sleijpen, and H. A. van der Vorst, "Reliable updated residuals in hybrid Bi-CG methods," Computing 56, 141-164 (1996).

Mixed precision with reliable updates

- New (?) idea is to apply this approach to mixed precision.
  (Clark et al., arXiv:0911.3191)

$r_0 = b - Ax_0;$
$\hat{r}_0 = r;$
$\hat{x}_0 = 0;$
$k = 0;$
while $||\hat{r}_k|| > \epsilon$ **do**
  Low precision solver iteration: $\hat{r}_k \rightarrow \hat{r}_{k+1}$, $\hat{x}_k \rightarrow \hat{x}_{k+1}$;
  if $||\hat{r}_{k+1}|| < \delta M(\hat{r})$ **then**
    $x_{l+1} = x_l + \hat{x}_{k+1};$
    $r_{l+1} = b - Ax_{l+1}$;
    $\hat{x}_{k+1} = 0;$
    $\hat{r}_{k+1} = r;$
    $l = l + 1;$
  end
  $k = k + 1$;
end

- ^ denotes reduced precision.
- δ is a parameter determining the frequency of updates.
- $M(\hat{r})$ denotes the maximum iterated residual since the last update.

- Reliable updates seems to win handily at light quark masses (and is no worse than iterative refinement at heavy masses).

GCR-DD

$k = 0$
$r_0 = b - Mx_0$
$\hat{r}_0 = r_0$
$\hat{x} = 0$
while $||r_0|| > tol$ **do**
  $\hat{p}_k = \hat{K}\hat{r}_k$
  $\hat{z}_k = \hat{M}\hat{p}_k$
  // Orthogonalization
  for $i \leftarrow 0$ **to** $k-1$ **do**
    $\beta_{i,k} = (\hat{z}_i, \hat{z}_k)$
    $\hat{z}_k = \hat{z}_k - \beta_{i,k}\hat{z}_i$
  end
  $\gamma_k = ||\hat{z}_k||$
  $z_k = z_k/\gamma_k$
  $\alpha_k = (\hat{z}_k, \hat{r}_k)$
  $\hat{r}_{k+1} = \hat{r}_k - \alpha_k\hat{z}_k$
  $k = k + 1$
  ⋮
  // High precision restart
  if $k = k_{max}$ **or** $||\hat{r}_k||/||r_0|| < \delta$ **or** $||\hat{r}_k|| < tol$ **then**
    for $l \leftarrow k-1$ **down to** 0 **do**
      solve $\gamma_l\chi_l + \sum_{i=l+1}^{k-1}\beta_{l,i}\chi_i = \alpha_l$ **for** χ_l
    end
    $\hat{x} = \sum_{i=0}^{k-1}\chi_i\hat{p}_i$
    $x = x + \hat{x}$
    $r_0 = b' - Ax$
    $\hat{r}_0 = r_0$
    $\hat{x} = 0$
    $k = 0$
  end
end