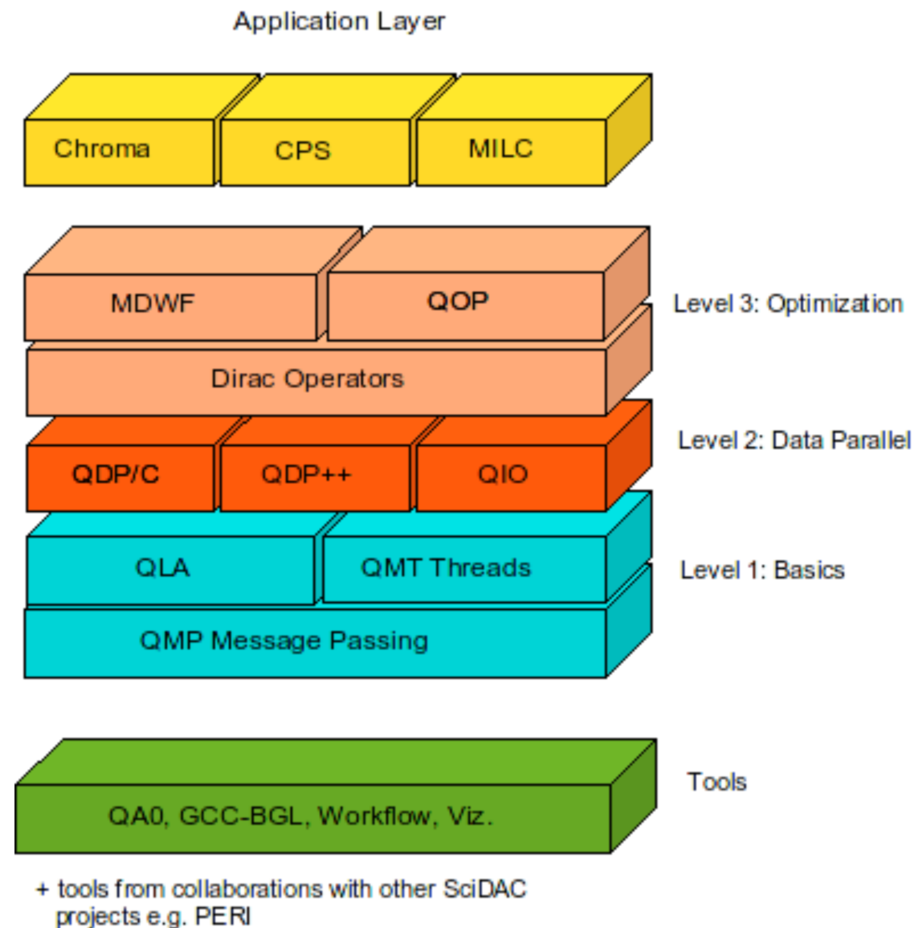
Part III: GPUs

- The problem of programming
  is making a Parallel problem Serial!

Library and Tool Dependencies

- QCD SciDAC API for Chroma/ CPS/MILC applications
- Level 3: Highly Optimized Dirac inverter, other critical kernels
- Level 2: Data Parallel Interface & IO library
- Level 1: Single core linear albebra, message passing, and threading libraries.
- Specialized code generators, workflow et al

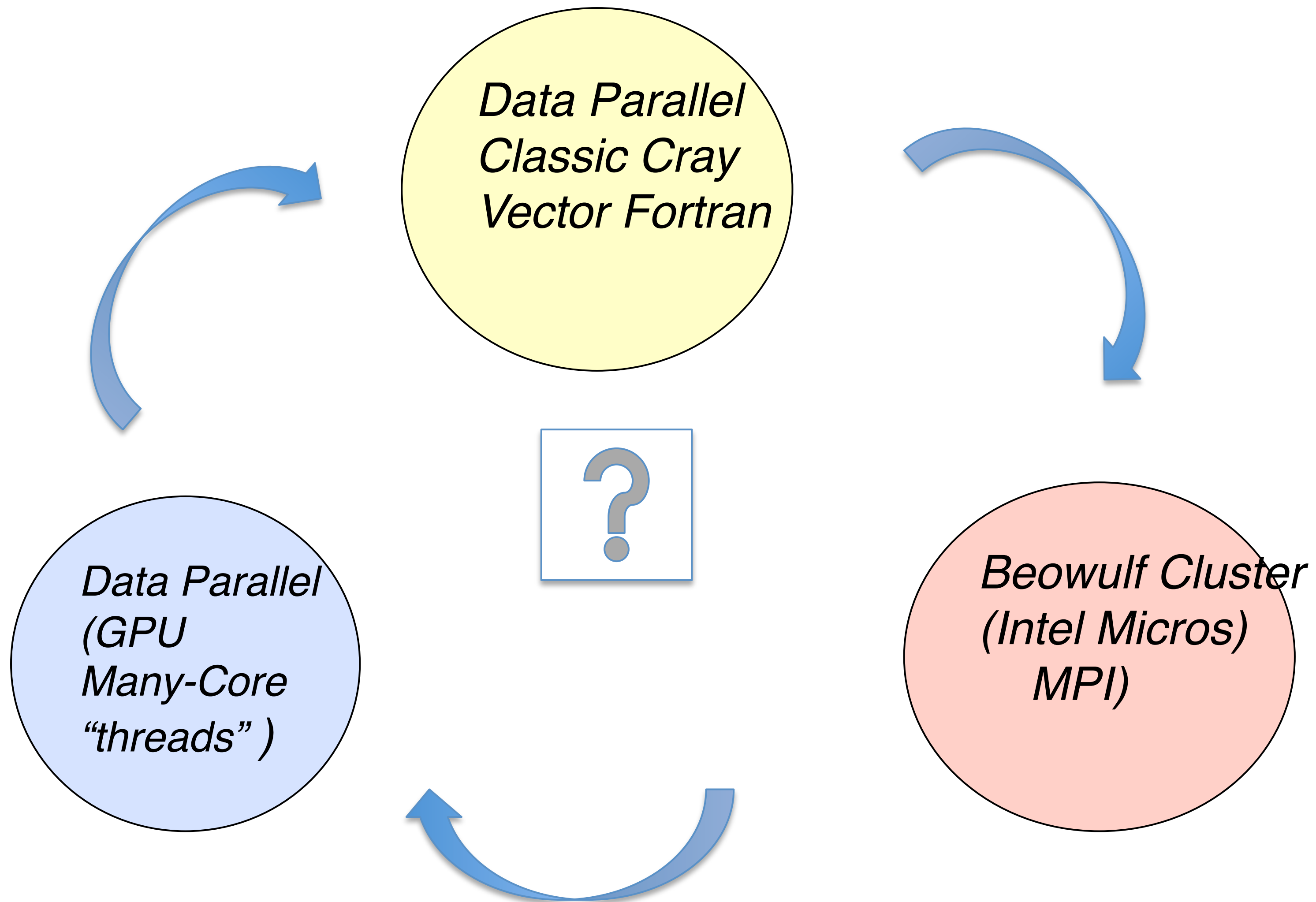


Rich Brower SciDAC Software co-ordinator.

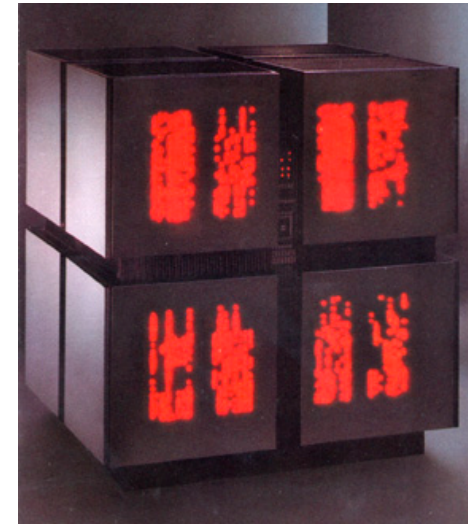
QUDA: QCD cuda Software

- Collaborators and QUDA developers:
    - Ron Babich
    - Kipton Barros (Northwestern)
    - Rich Brower (BU)
    - Mike Clark (Harvard)
    - Steve Gottlieb (Indiana)
    - Bálint Joó (JLab)
    - Claudio Rebbi (BU)
    - Guochun Shi (NCSA)
- SciDAC Lattice QCD Module

History: Architecture Revolution
Exascale Back to the Future!

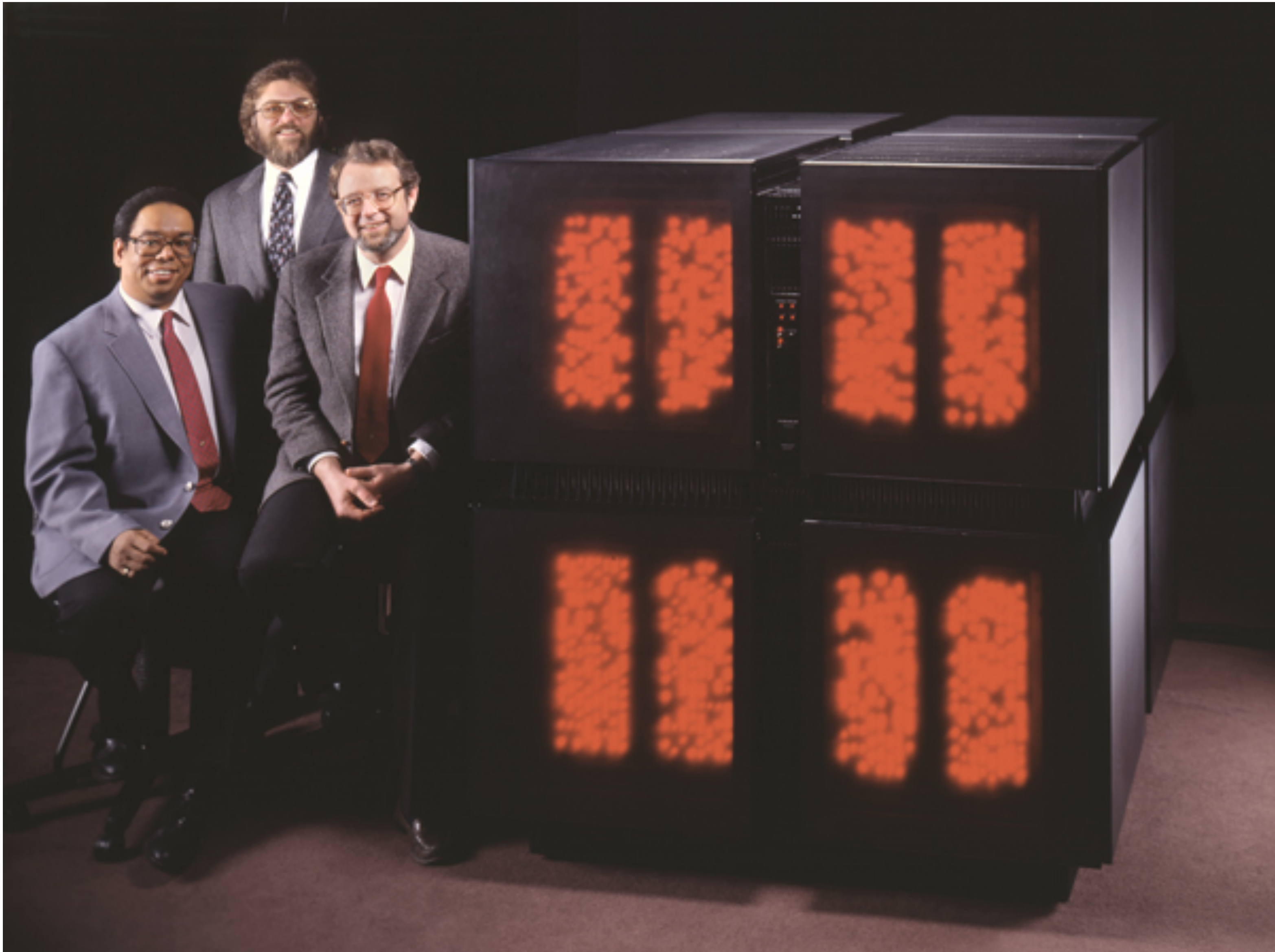


Historical Perspective: First "commercial" QCD machine

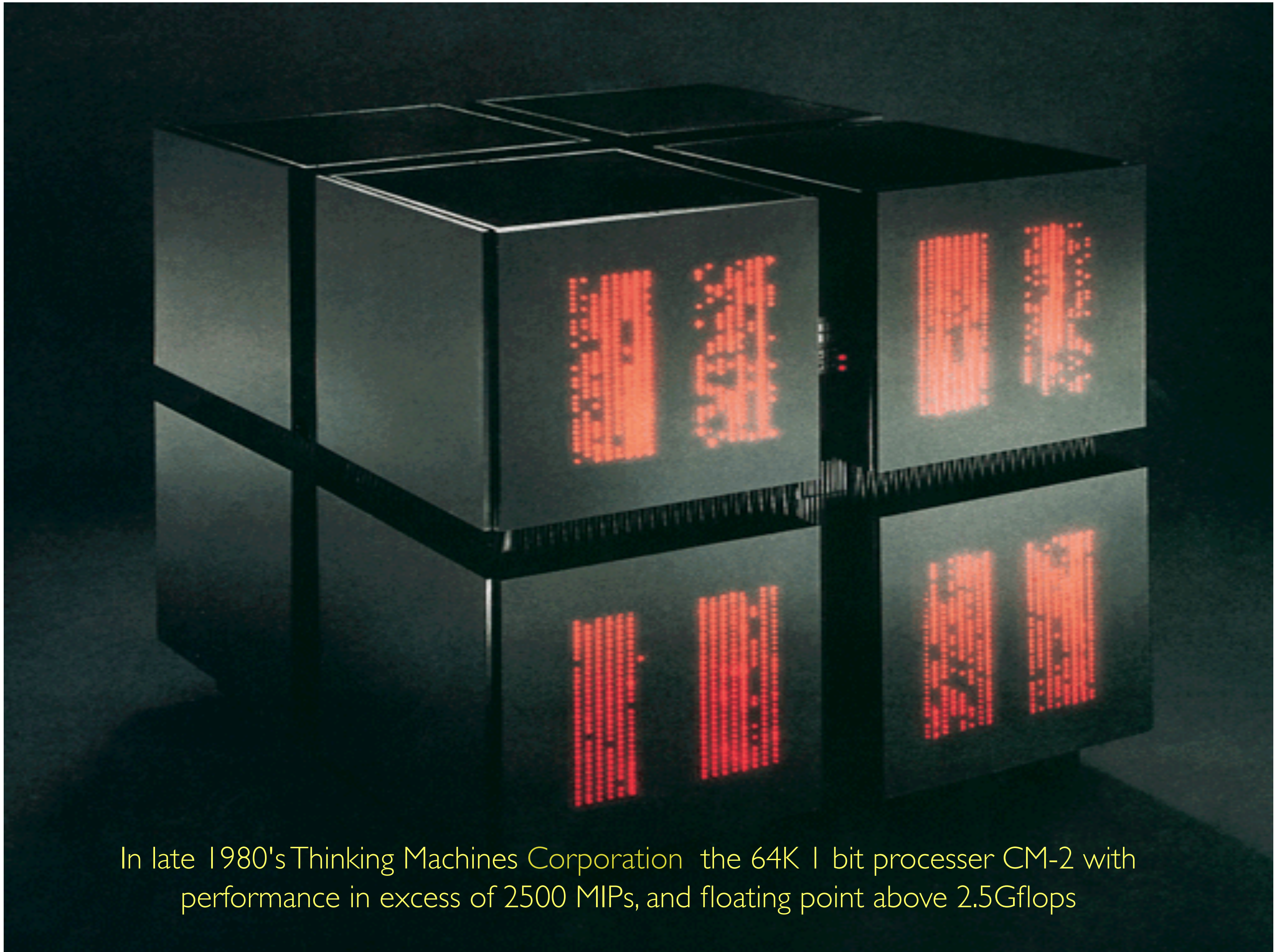


http://www.mission-base.com/tamiko/cm/cm-tshirt.html

First University installation in 1989*



*(from left) Roscoe Giles, Glenn Bresnahan and Claudio Rebbi with CM-2


In late 1980's Thinking Machines Corporation the 64K 1 bit processer CM-2 with performance in excess of 2500 MIPs, and floating point above 2.5Gflops

Killed by Beowulf-clusters

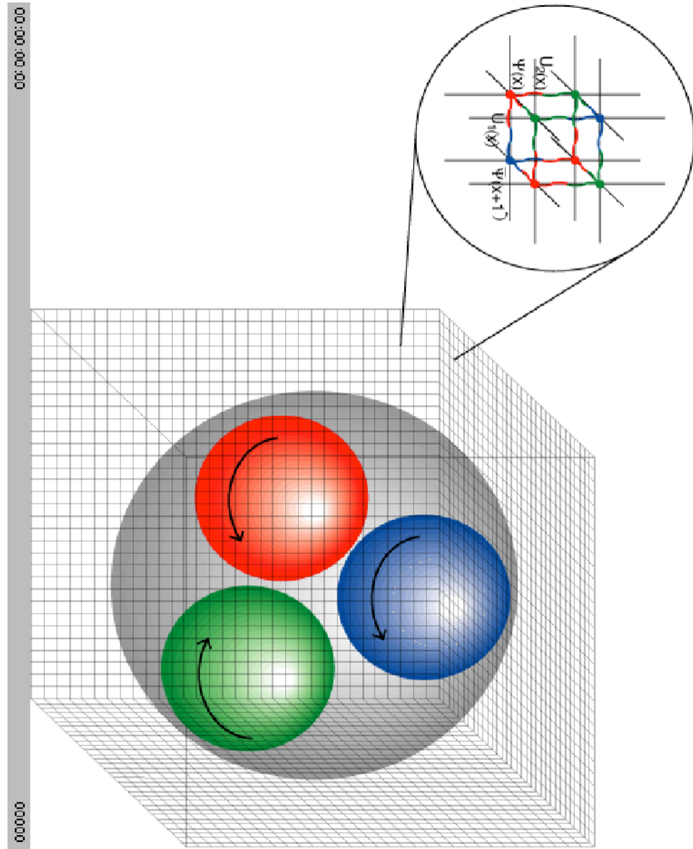
BNL:QCDOC

JLab

FNAL

BG/L

QCD Perfect scaling!



Wilson Inverter on the BlueGene/L

Benchmark Points

19% of Peak

Wilson Dirac Inverter

Teraflop/s: 0, 20, 40, 60, 80

Cores: 0.0×10^5, 0.5×10^5, 1.0×10^5, 1.5×10^5

† LLNL BG/L weak scaling up to 131,072 cores: 2006 Gordon Bell award

by Vranas, Bhanot, Blumrich, Chen, Gara, Giampapa, Heidelberger, Salapura and Sexton

BUT THIS IS NOT SUFFICIENT

K. Wilson (1989 Capri)

"lattice gauge theory could also require a 10^8 increase in computer power AND spectacular algorithmic advances before useful interactions with experiment ..."

- ab initio Chemistry
1. 1930+50 = 1980
2. 0.1 flops ➔ 10 Mflops
3. Gaussian Basis functions

vs

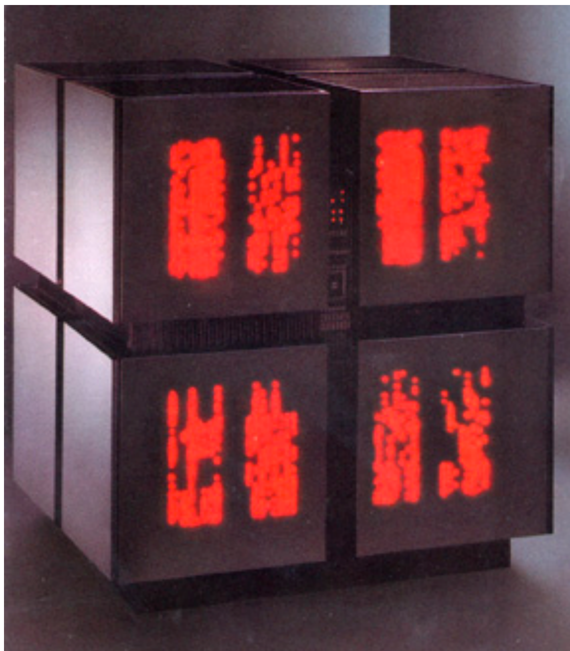
- ab initio QCD
1. 1980 + 50 = 2030?*
2. 10 Mflops ➔ 1000 Tflops
3. Clever Collective Variable?

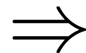
**Hopefully sooner but need $1/Mflops ➔ $1/Gflops!*

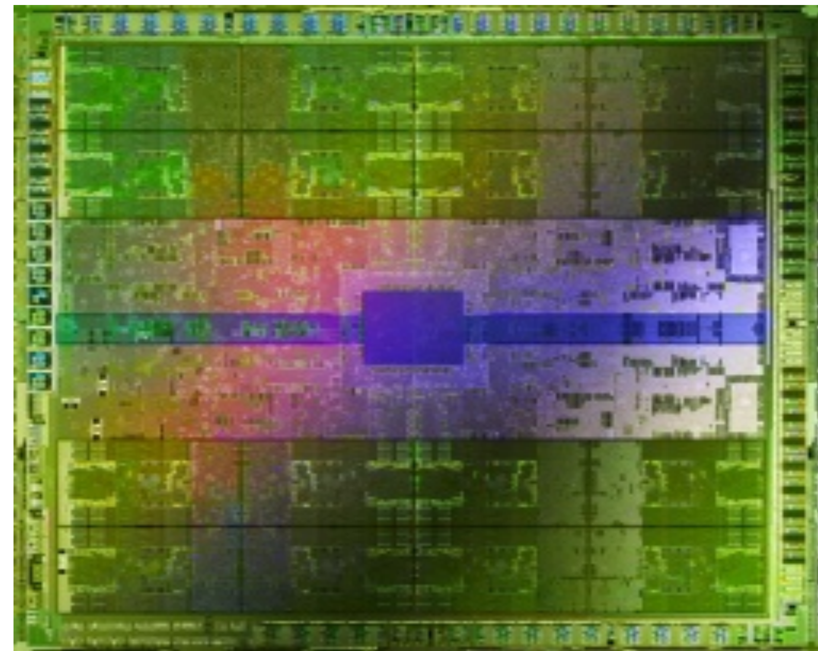
Disruptive many-core Architectures

- 1/4 CM-2

Nvidia FERMI chip

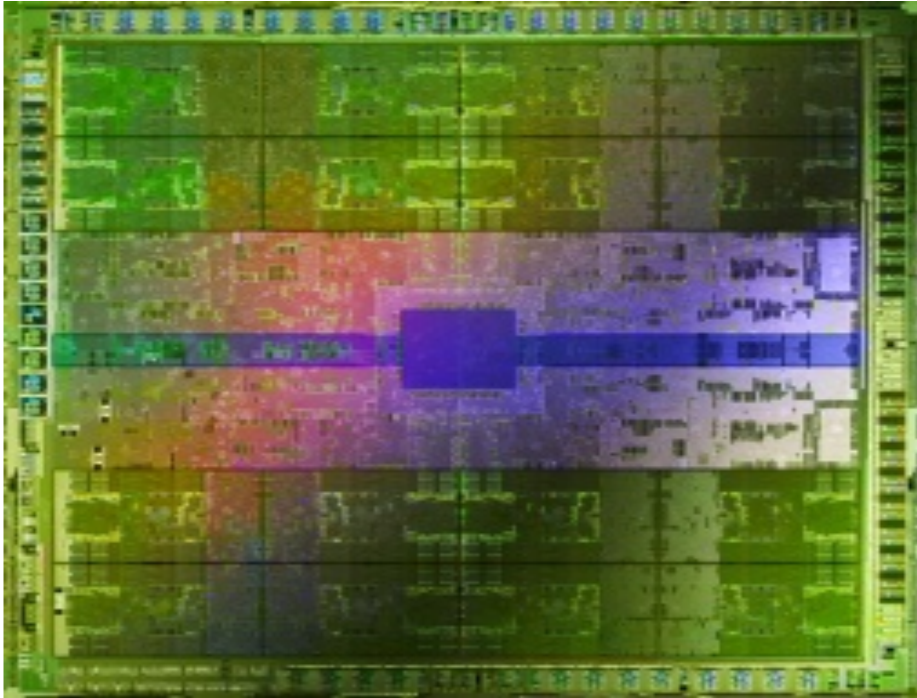






16 K bit serial PE. $\Rightarrow$ 512 x 32 bit PE = 16 K bits

- *Disruptive QCD Technology! Graphic Processor Units*



Nvidia's Fermi GPU:
512 cores x 32 bit FPU

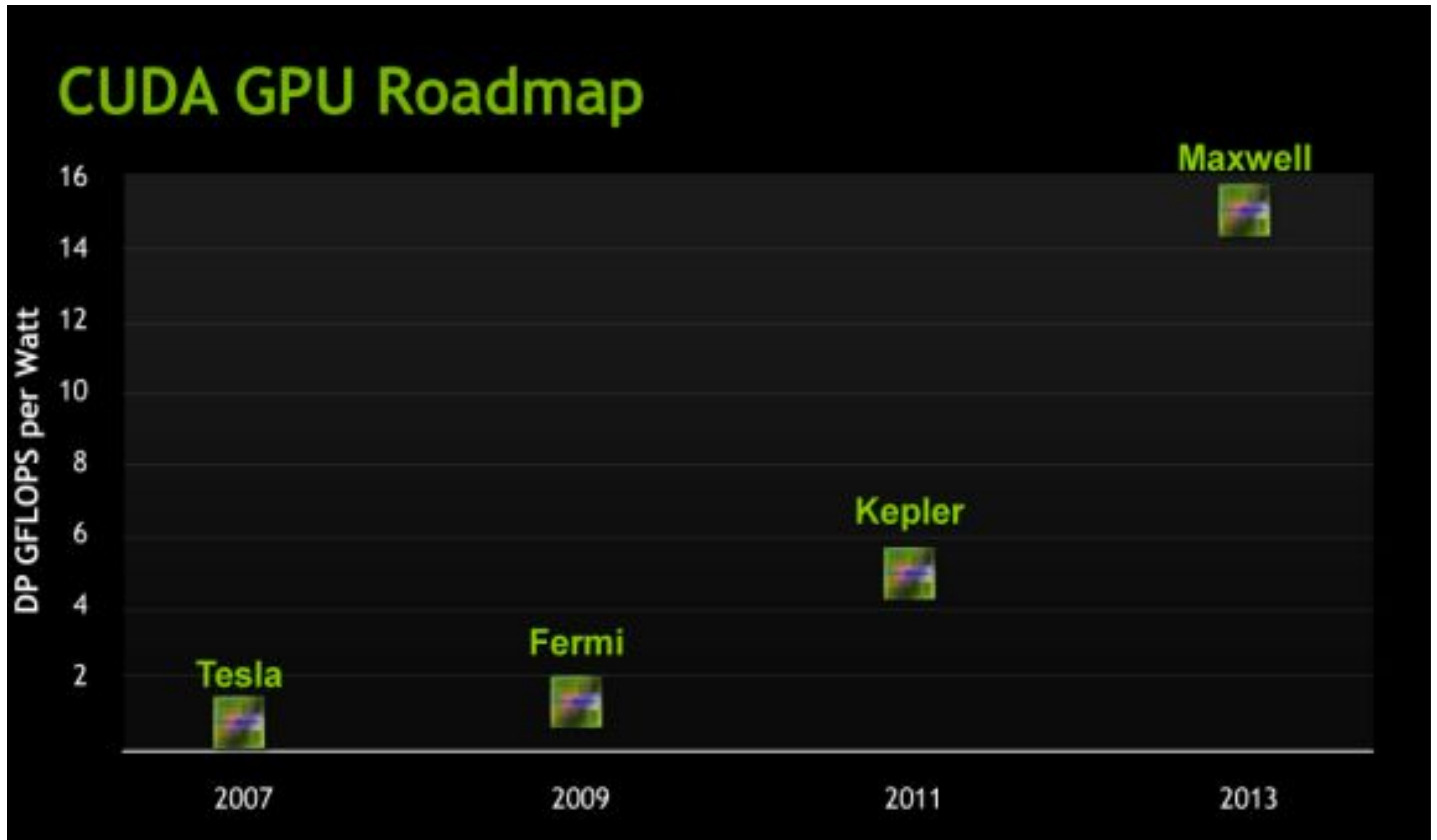
3-6 GigaByte Mem, 1+ Tflop peak

•GPU/BU code sent to Jefferson Lab for 3 million $ ARRA Cluster
⇒ 5 x performance @ 20% extra cost

GPUs on the Green 500

Green500 Rank	MFLOPS/ W	Site*	Computer*	Total Power (kW)
1	773.38	Forschungszentrum Juelich (FZJ)	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	57.54
1	773.38	Universitaet Regensburg	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	57.54
1	773.38	Universitaet Wuppertal	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	57.54
4	492.64	National Supercomputing Centre in Shenzhen (NSCS)	Dawning Nebulae, TC3600 blade CB60-G2 cluster, Intel Xeon 5650/ nVidia C2050, Infiniband	2580
5	458.33	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Infiniband	276
5	458.33	IBM Poughkeepsie Benchmarking Center	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Infiniband	138
7	444.25	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband	2345.5
8	431.88	Institute of Process Engineering, Chinese Academy of Sciences	Mole-8.5 Cluster Xeon L5520 2.26 Ghz, nVidia Tesla, Infiniband	480
9	418.47	Mississippi State University	iDataPlex, Xeon X56xx 6C 2.8 GHz, Infiniband	72
10	397.56	Banking (M)	iDataPlex, Xeon X56xx 6C 2.66 GHz, Infiniband	72

Exaflops Power/Communication Wall

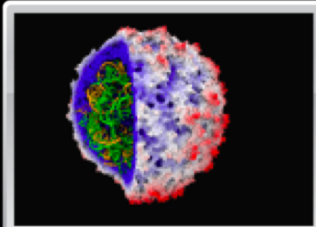


Hardware (e.g., NVIDIA)

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

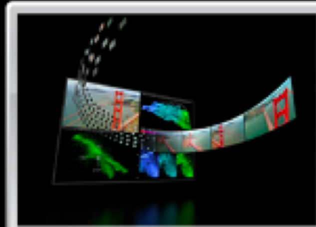
146X

Interactive visualization of volumetric white matter connectivity



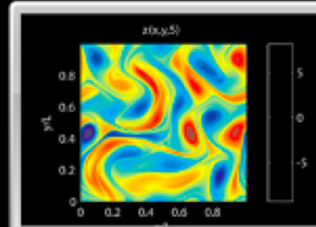
36X

Ionic placement for molecular dynamics simulation on GPU



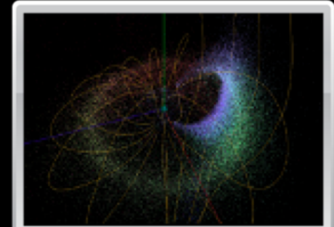
19X

Transcoding HD video stream to H.264



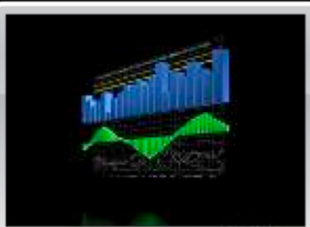
17X

Fluid mechanics in Matlab using .mex file CUDA function



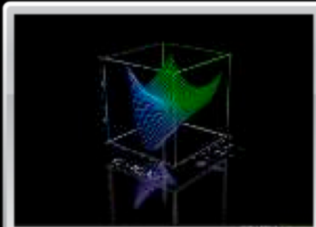
100X

Astrophysics N-body simulation



149X

Financial simulation of LIBOR model with swaptions



47X

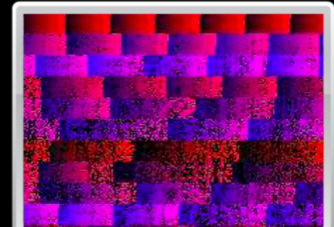
GLAME@lab: an M-script API for GPU linear algebra

20X

Ultrasound medical imaging for cancer diagnostics

24X

Highly optimized object oriented molecular dynamics



30X

Cmatch exact string matching to find similar proteins and gene sequences

Harvard/BU Cuda Center of Excellence

Tesla 1070 Nvidia Gift

NSF EAGER $300K grant for to build Experimental GPU Fermi cluster for QCD and CFD

NSF EAGER system at BU

- Measured on a cluster node at BU, similar to JLab nodes (but with Tesla C1060 cards, rather than GeForce GTX 285).

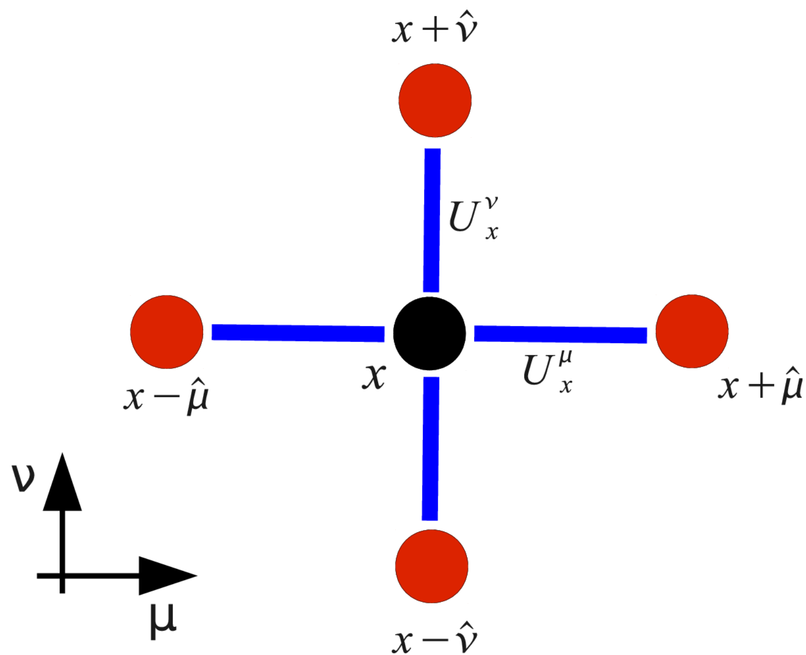
QUDA: QCD CUDA @ BU

- QUDA library ("QCD on CUDA") available here:
  - http://lattice.bu.edu/quda
- Provides optimized **CG** and **BiCGstab** solvers for Wilson and clover-improved Wilson, supporting **mixed precision** with **reliable updates**.
- Release 0.3 includes support for staggered fermions, contributed by Steve Gottlieb, Guochun Shi, and collaborators.
- Domain wall (contributed by Joel Giedt), twisted mass (contributed by Alexei Strelchenko), and **multi-GPU** support will be available soon.
- Conveniently interfaced to existing packages (Chroma/QDP++, QDP/C, CPS, etc.).

The Wilson Dirac operator

- The Wilson Dirac operator is given by

$$D_{x,x'} = -\frac{1}{2}\sum_{\mu=1}^{4}\left(P^{-\mu}\otimes U_x^{\mu}\,\delta_{x+\hat{\mu},x'} + P^{+\mu}\otimes U_{x-\hat{\mu}}^{\mu\dagger}\,\delta_{x-\hat{\mu},x'}\right) + (4+m)\delta_{x,x'}$$



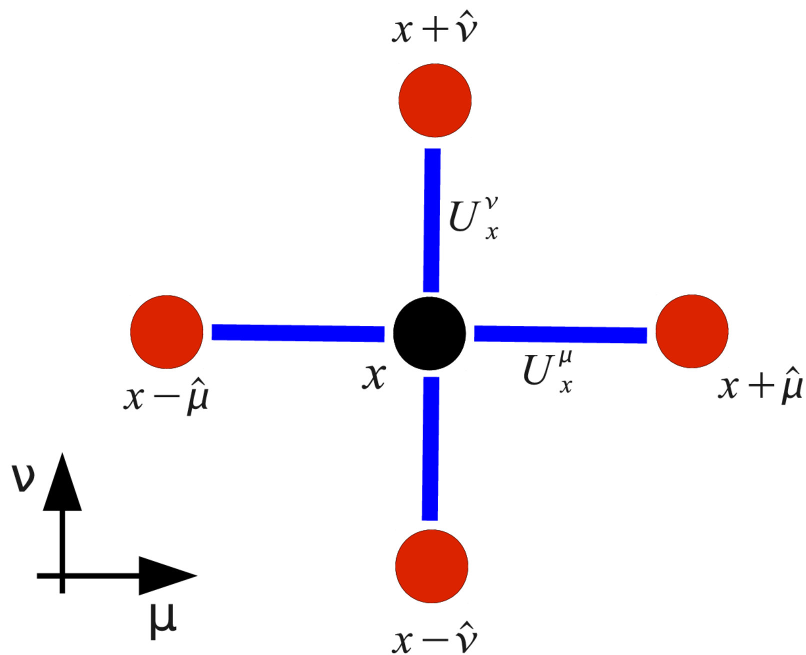
$$P^{\pm 1} = \begin{pmatrix} 1 & 0 & 0 & \pm i \\ 0 & 1 & \pm i & 0 \\ 0 & \mp i & 1 & 0 \\ \mp i & 0 & 0 & 1 \end{pmatrix},\ P^{\pm 2} = \begin{pmatrix} 1 & 0 & 0 & \mp 1 \\ 0 & 1 & \pm 1 & 0 \\ 0 & \pm 1 & 1 & 0 \\ \mp 1 & 0 & 0 & 1 \end{pmatrix}$$

$$P^{\pm 3} = \begin{pmatrix} 1 & 0 & \pm i & 0 \\ 0 & 1 & 0 & \mp i \\ \mp i & 0 & 1 & 0 \\ 0 & \pm i & 0 & 1 \end{pmatrix},\ P^{\pm 4} = \begin{pmatrix} 1 & 0 & \pm 1 & 0 \\ 0 & 1 & 0 & \pm 1 \\ \pm 1 & 0 & 1 & 0 \\ 0 & \pm 1 & 0 & 1 \end{pmatrix}$$

- *Cost to apply this matrix to a vector?*
- *Bytes moved to/from memory?*
- *Floating-point operations?*

Wilson matrix-vector: Flop count

$$D_{x,x'} = -\frac{1}{2}\sum_{\mu=1}^{4}\left(P^{-\mu}\otimes U_x^{\mu}\,\delta_{x+\hat{\mu},x'} + P^{+\mu}\otimes U_{x-\hat{\mu}}^{\mu\dagger}\,\delta_{x-\hat{\mu},x'}\right) + (4+m)\delta_{x,x'}$$



Spin-project (from 4 to 2 independent spin components), once per direction: 8 x 12 = 96 flops

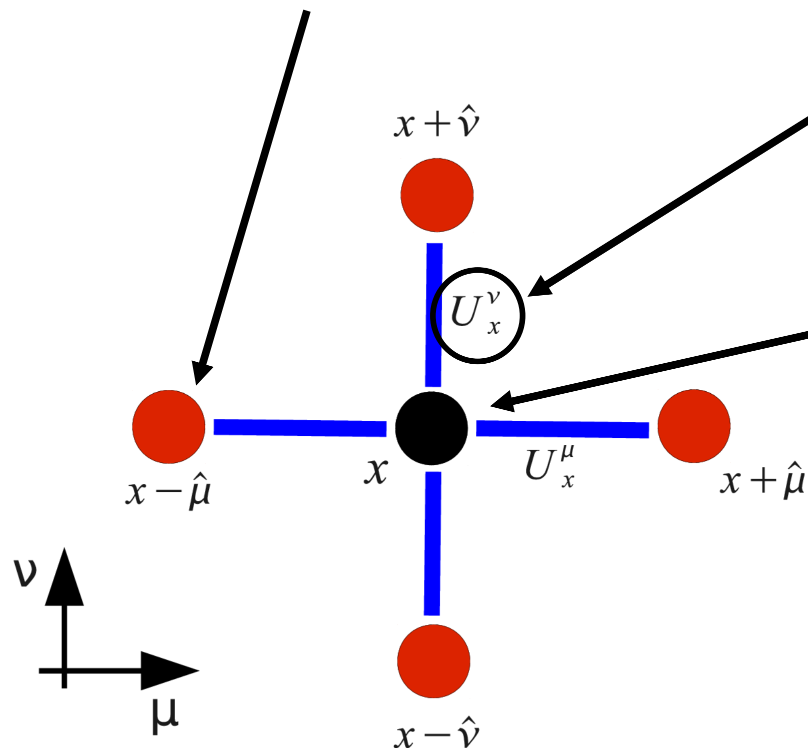
SU(3) multiply, once per direction: 8 x 132 = 1056 flops

Accumulate to the output spinor: 7 x 24 = 168 flops

Altogether, this gives ***1320 flops*** *per site.*

Wilson matrix-vector: Data movement

- To carry out the Wilson matrix-vector product, per site, we must:
- Read one spinor per direction, for a total of 8 x 24 = 192 floats



Read one gauge matrix per direction, for a total of 8 x 18 = 144 floats

Write one output spinor, consisting of 24 floats

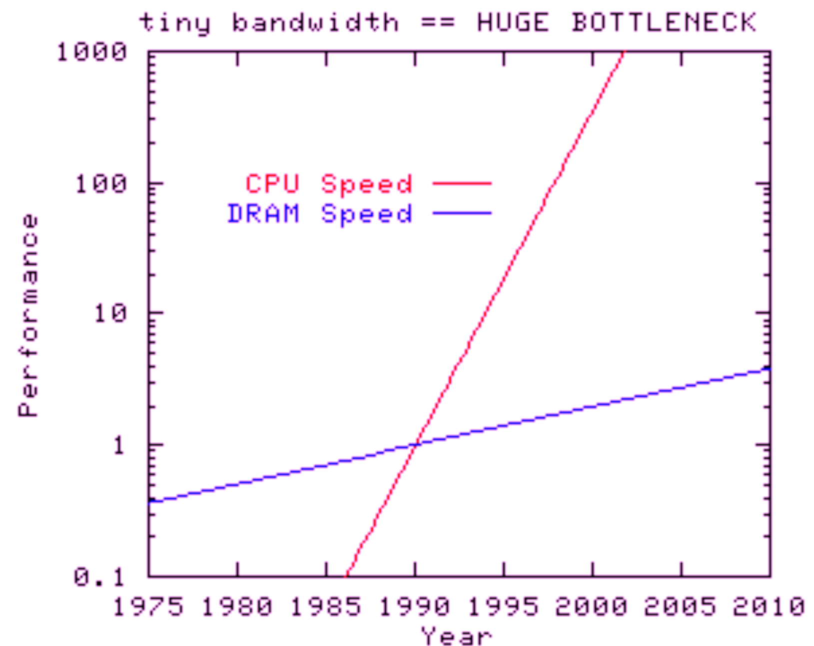
*Altogether, we must transfer 360 floats from memory, or **1440 bytes** in single precision.*

Byte/flop ratios

- In modern architectures, the main bottleneck is often bandwidth to memory, rather than raw floating point throughput.
- We've seen that applying the Wilson Dirac operator (in single precision) requires that we move **1440 bytes** for every **1320 flops**.
- Clover-improved Wilson (Sheikholeslami-Wohlert) is a bit "better" (less memory-bound), at **1728 bytes** and **1824 flops**.
- Asqtad and HISQ are a bit worse, at **1560 bytes** and **1146 flops**.
- Other operations (beyond the matrix-vector product) are often much worse. For example, adding two vectors of length *N* involves reading/writing a total of **12*N* bytes** but consists of only ***N* flops.**

Machine balance: Looking back

- In modern architectures, the main bottleneck is often bandwidth to memory, rather than raw floating point throughput.
- It wasn't always like this:

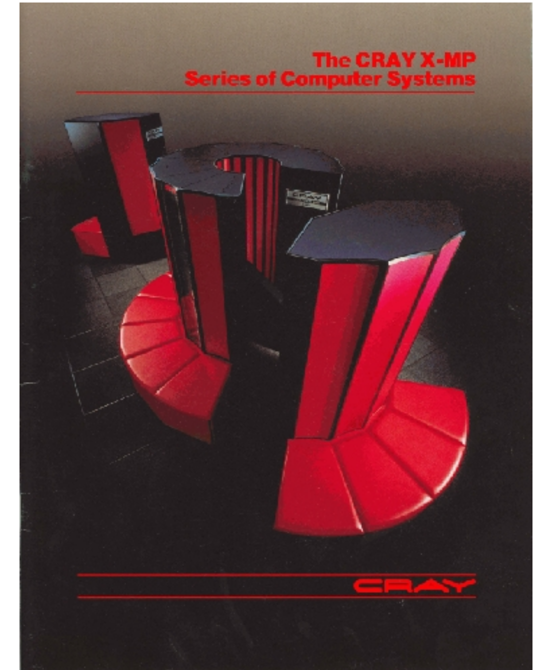


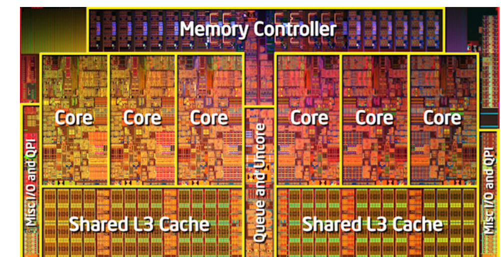
(Cartoon by John McCalpin, author of the STREAM benchmark)

- There was a time when flops were much more precious, relative to bandwidth.

Machine balance: Looking back

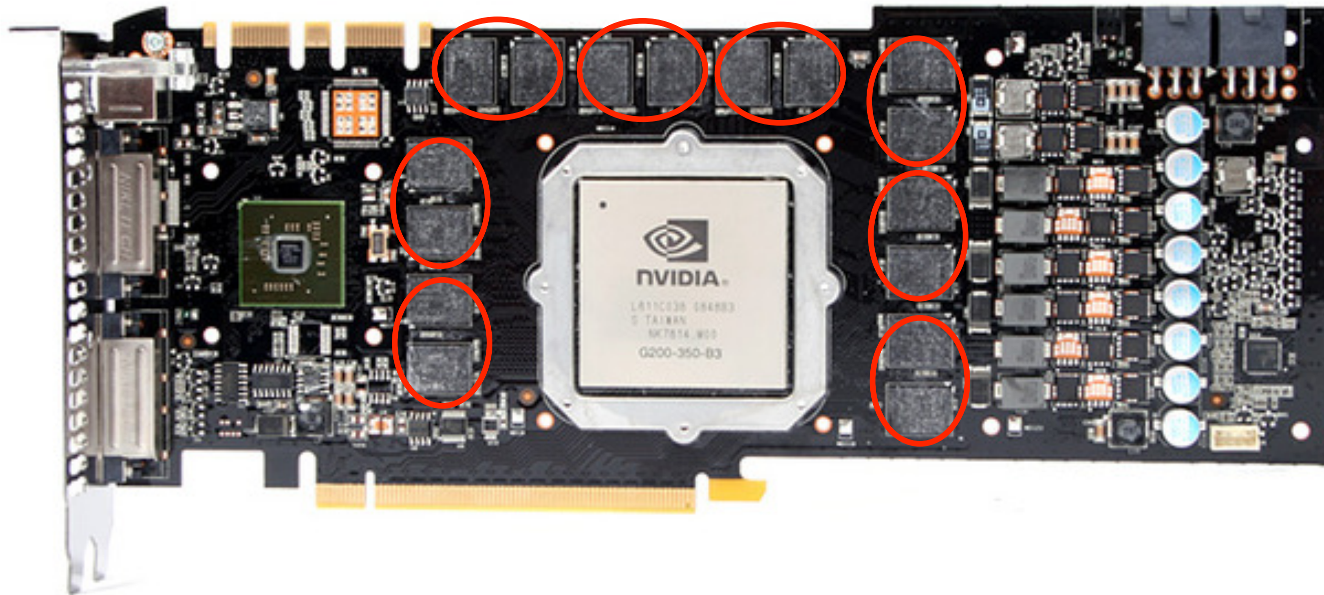
- Early vector machines delivered high bandwidth through a very wide memory bus.
- Memory of the Cray X-MP/4 (ca. 1985) was arranged in 32 (64-bit) banks, delivering **128 GB/s** – respectable even today!
- The MPPs, SMPs, and commodity clusters that followed relied increasingly on *data caches* (small amount of fast memory close to the processor) to contend with the balance issue.
- Modern commodity processors have three levels of cache, with L3 (and sometimes L2) shared among multiple cores.





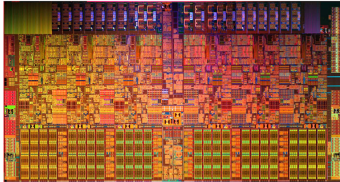
Machine balance: Looking back

- In this respect, GPUs are a throwback to the past. The NVIDIA GeForce GTX 285 has eight 64-bit banks, delivering **159 GB/s**:



(The more recent GTX 480 has a narrower 384-bit bus, but faster memory.)

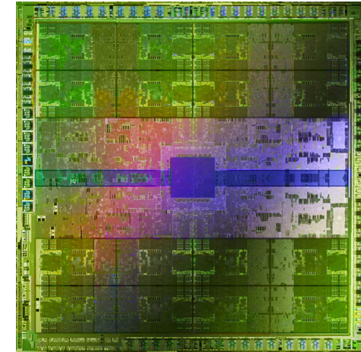
A tale of two processors



"Gulftown"

- **Intel Xeon X5680**
- 6 cores (each with 4-wide SSE unit)
- 1.17 billion transistors
- Shared L3 Cache: 12 MB
- L1+L2: 6 x (320 KB) = 1920 KB
- **160 Gflops (SP)**
- **32 GB/s memory bandwidth**
- up to 288 GB (96 GB is realistic)

5:1



"Fermi"

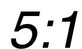
- **NVIDIA GeForce GTX 480**
- 480 cores
- 3.0 billion transistors
- Shared L2 Cache: 768 KB
- L1+SM+Reg: 15 x 192 KB = 2880 KB
- **1345 Gflops (SP)**
- **177 GB/s memory bandwidth**
- 1.5 GB (up to 6 GB in Tesla variant)

7.5:1

Bandwidth constraints

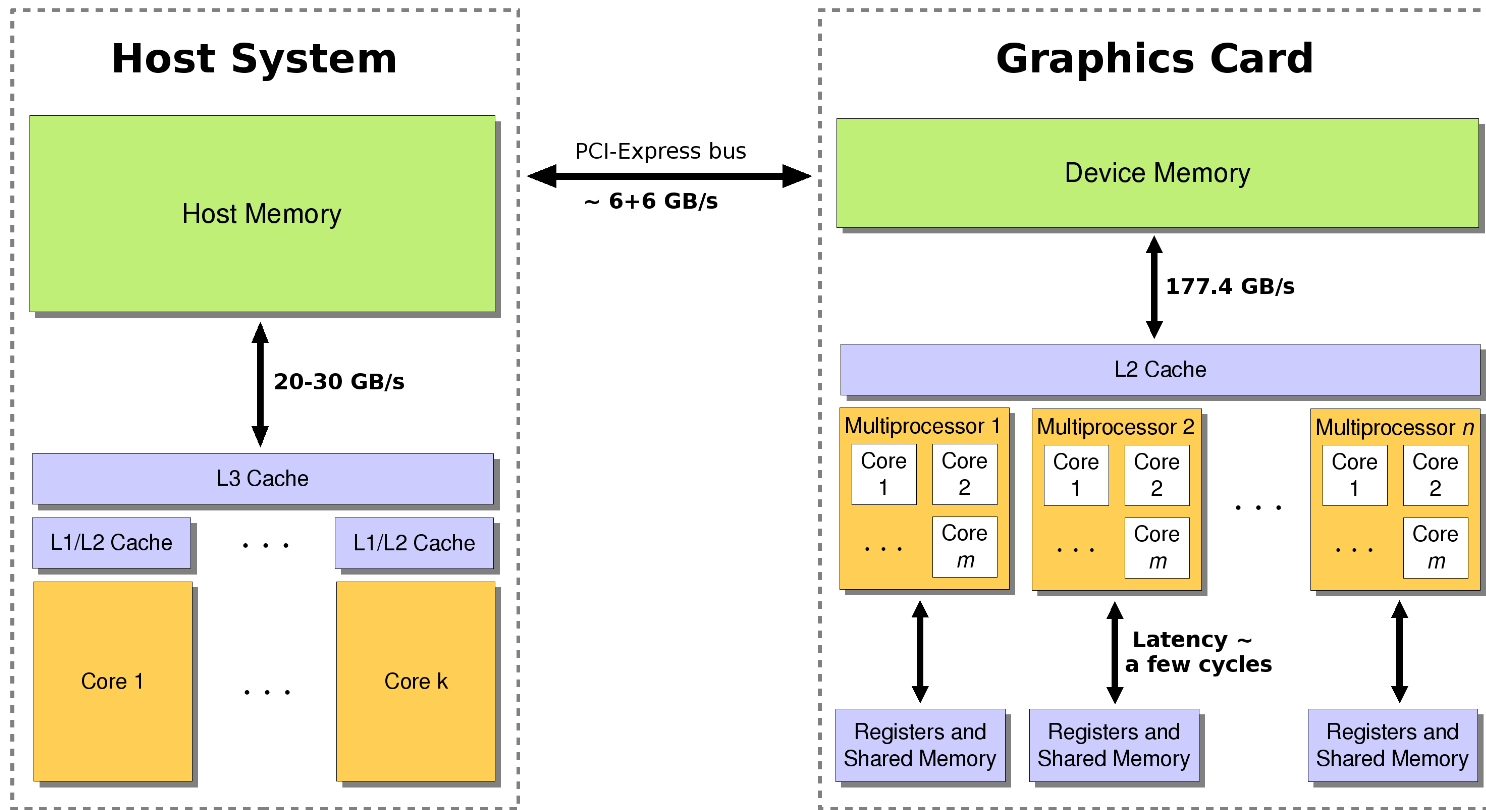
- *Recall that in single precision, the Wilson matrix-vector product has a byte/flop ratio of just over 1 (slightly lower for clover).*
- *We're entirely constrained by memory bandwidth. On the GPU, flops are virtually free.*

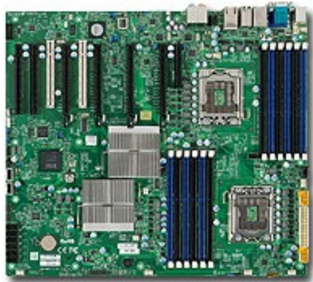
- **160 Gflops (SP)**
- **32 GB/s memory bandwidth**



- **1345 Gflops (SP)**
- **177 GB/s memory bandwidth**

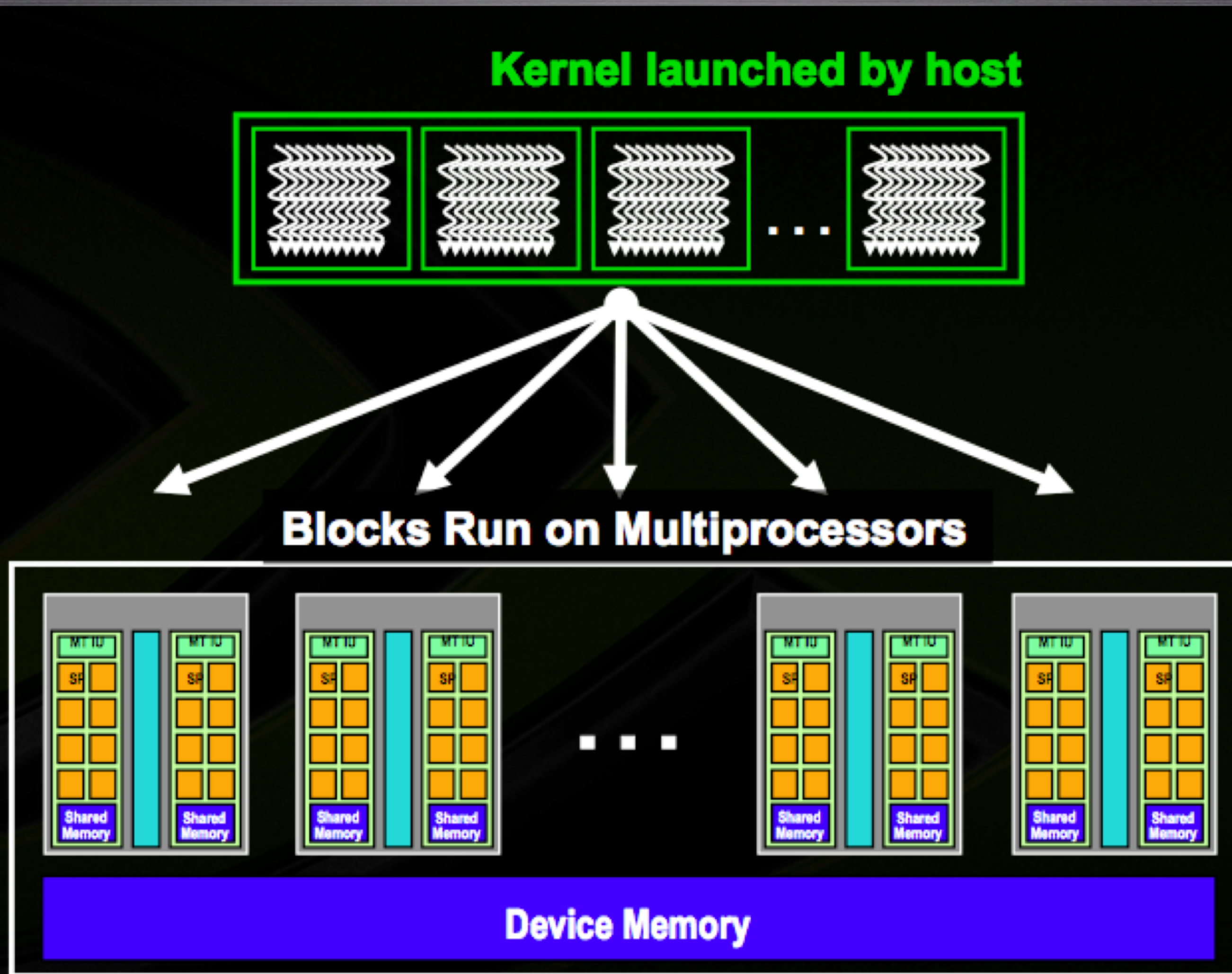
GPU memory hierarchy





(GeForce GTX 480)

CUDA programming model



Serial on CPU and Data Parallel on GPU

CUDA SDK

Libraries:FFT, BLAS,...
Example Source Code

Integrated CPU
and GPU C Source Code

NVIDIA C Compiler

NVIDIA Assembly
for Computing

CPU Host Code

CUDA
Driver

Debugger
Profiler

Standard C Compiler

GPU

CPU

© NVIDIA Corporation 2007

CUDA Programming

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Standard C Code

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

Tricks to reduce memory traffic

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

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

$$SU(2) : U = a_0\sigma_0 + \vec{a} \cdot \sigma$$

$$a_0^2 + \vec{a} \cdot \vec{a} = 1 \implies S_3 \text{ sphere}$$

Better still SU(3) has 8 parameters on S3 x S5

- Choose a gamma basis with γ_4 diagonal.

$$P^{\pm 4} = \begin{pmatrix} 1 & 0 & \pm 1 & 0 \\ 0 & 1 & 0 & \pm 1 \\ \pm 1 & 0 & 1 & 0 \\ 0 & \pm 1 & 0 & 1 \end{pmatrix} \Rightarrow \left\{ \begin{array}{l} P^{+4} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ P^{-4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \end{array} \right.$$

similarity transforms on D

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

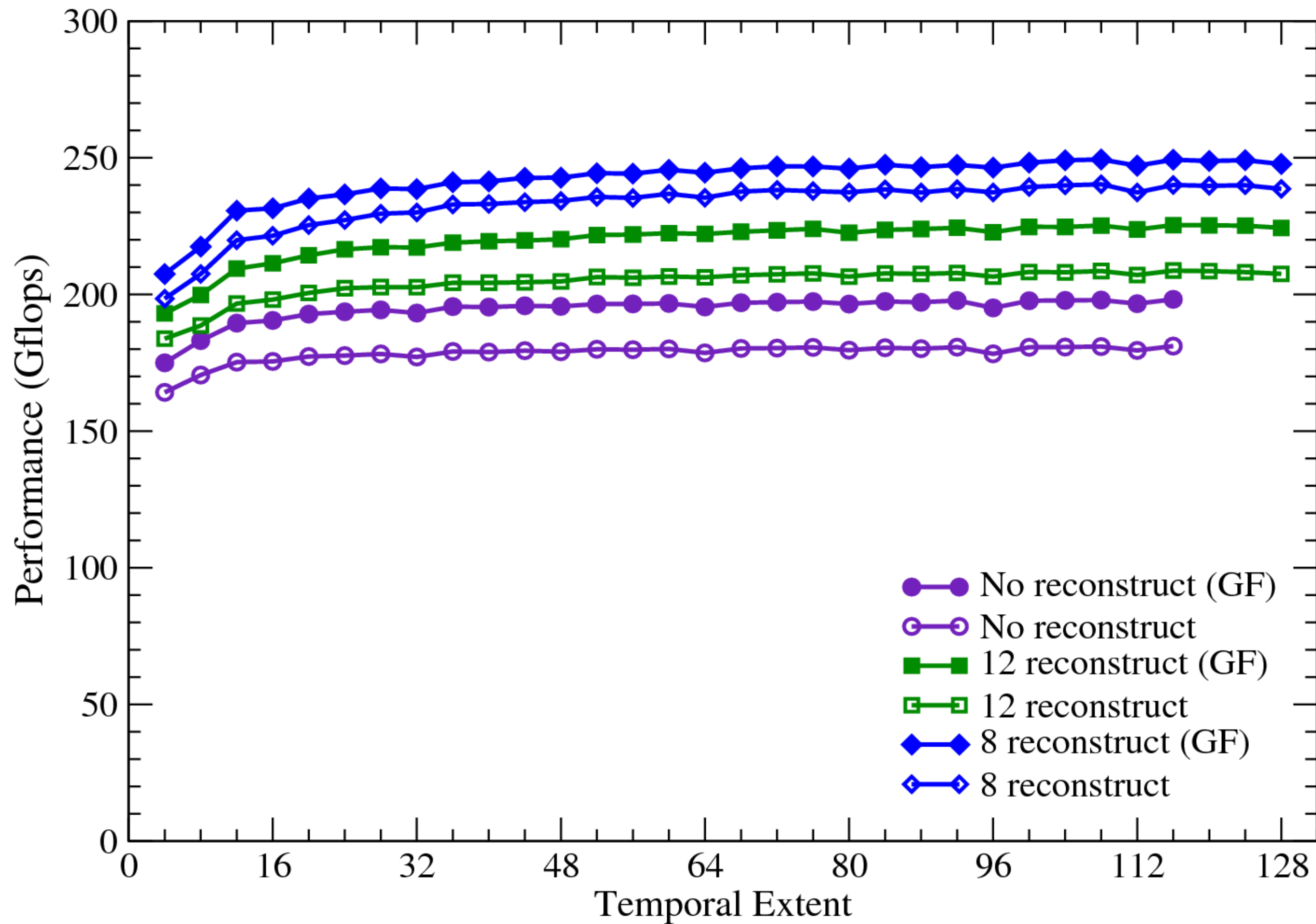
Performance results

- Results are for the even/odd preconditioned clover-improved Wilson matrix-vector product ("*Dslash*").

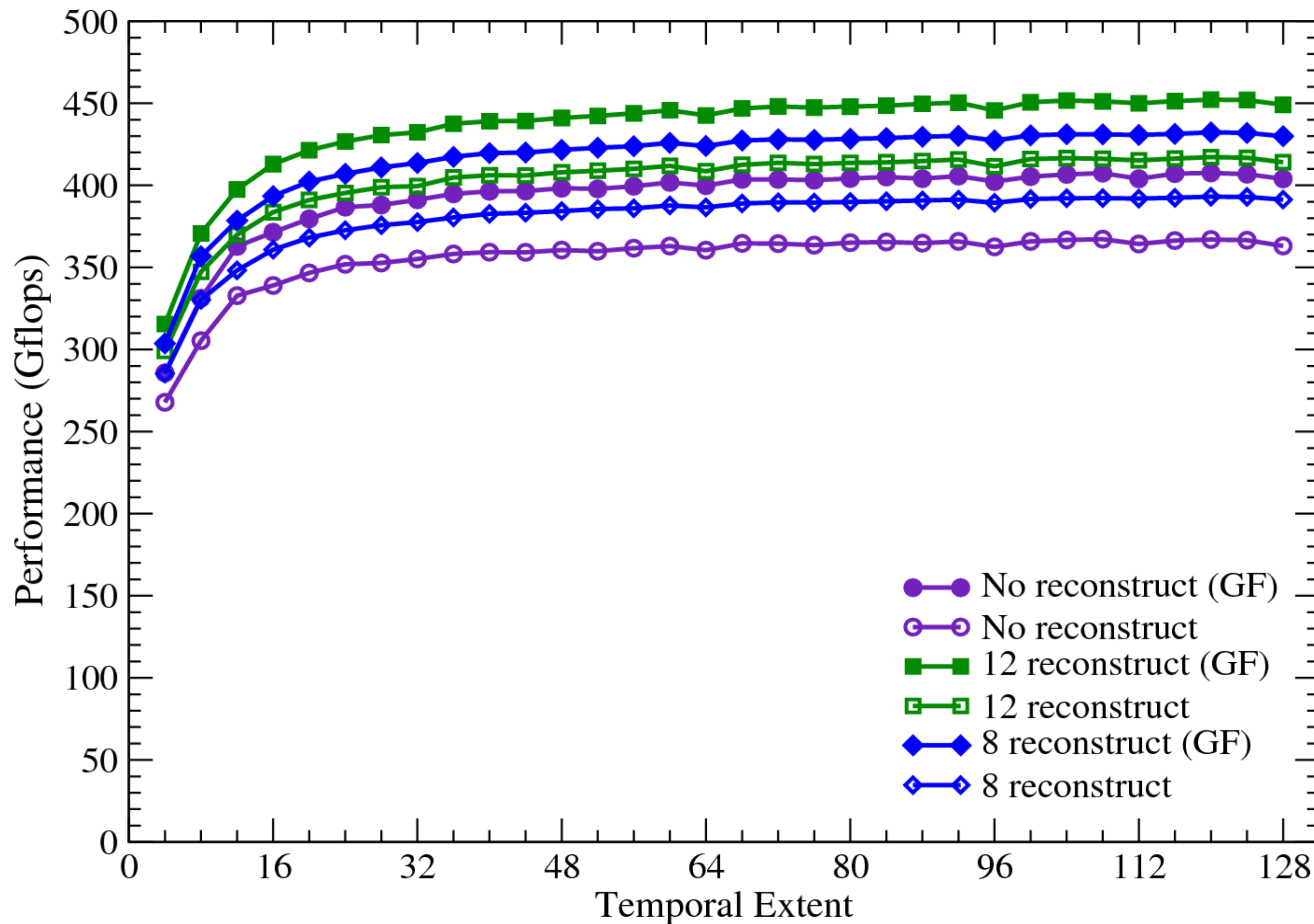
$$M = (1 - A_{ee}^{-1} D_{eo} A_{oo}^{-1} D_{oe})$$

- Runs were done on a GeForce GTX 480 (consumer-level "Fermi" card).
- For reference, a standard dual-socket node with recent (Westmere) quad-core Xeons would sustain around **20 Gflops** in single precision for a well-optimized Wilson-clover Dslash.
- We'll compare results for double, single, and half precision. In this case, half is a 16-bit quasi-fixed-point implementation, but GPUs support true FP16 as well.
- The spatial volume is held fixed at 24^3.

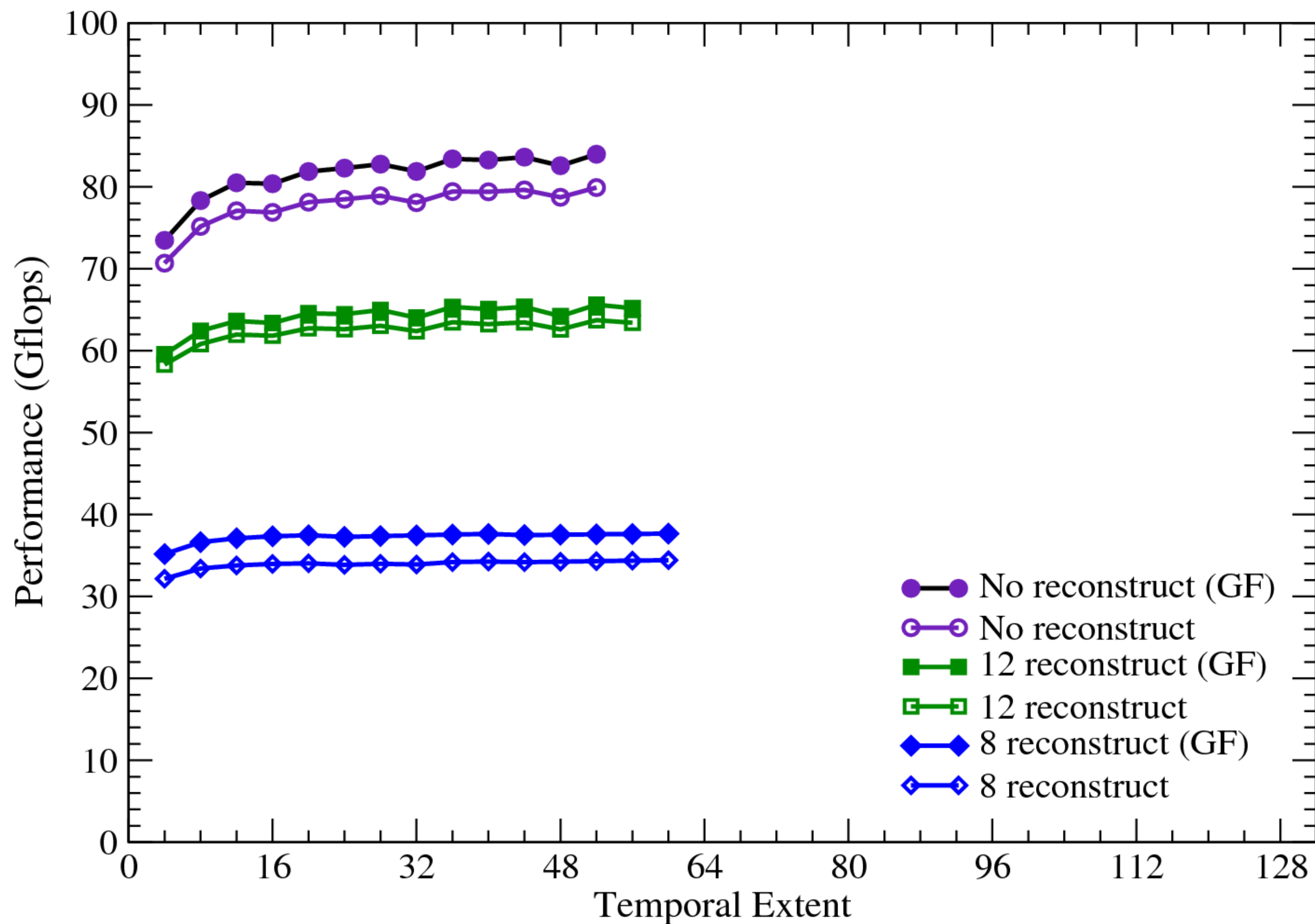
Clover performance (single precision)



Dslash performance (half precision)

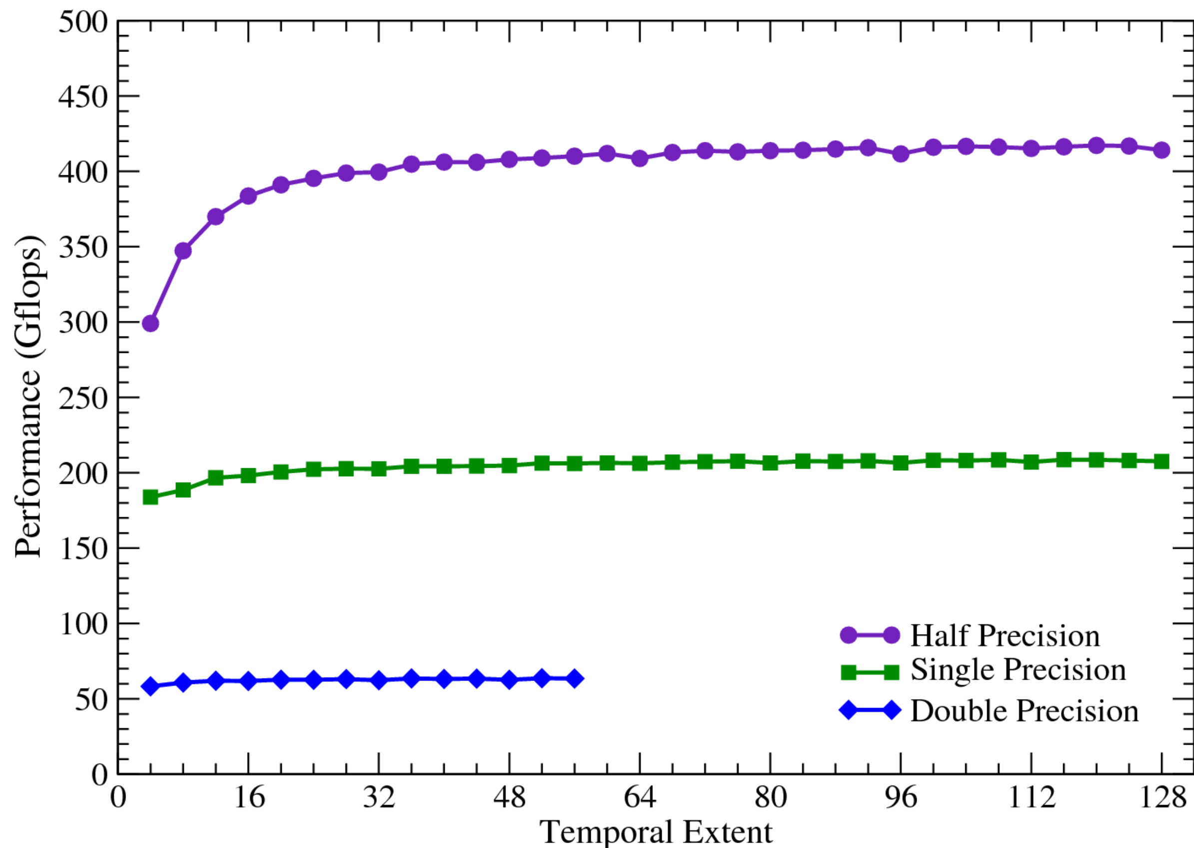


Dslash performance (double precision)



Dslash performance summary

- Summarized results are for a conservative case (12-reconstruct with no temporal gauge-fixing).



- Single and half performance are about 3.3x and 6.5x higher than double.

Mixed precision with reliable updates

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

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

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

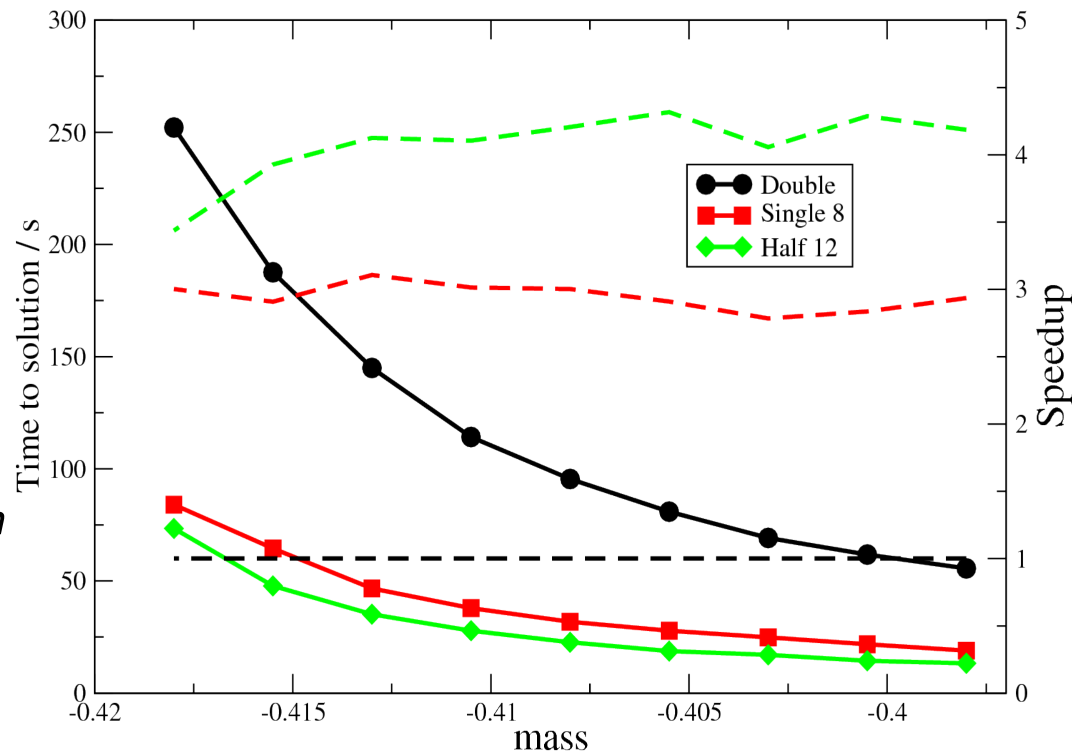
- *^ denotes reduced precision.*
- δ *is a parameter determining the frequency of updates.*
- $M(\hat{r})$ *denotes the maximum iterated residual since the last update.*

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

Mixed precision with reliable updates

- With this approach, even half (16-bit) precision is worthwhile. Mixed single/half or double/half results in only a 10-20% increase in iteration count as compared to pure single or pure double, respectively.

(Time to solution on a real lattice; updates are in double precision)



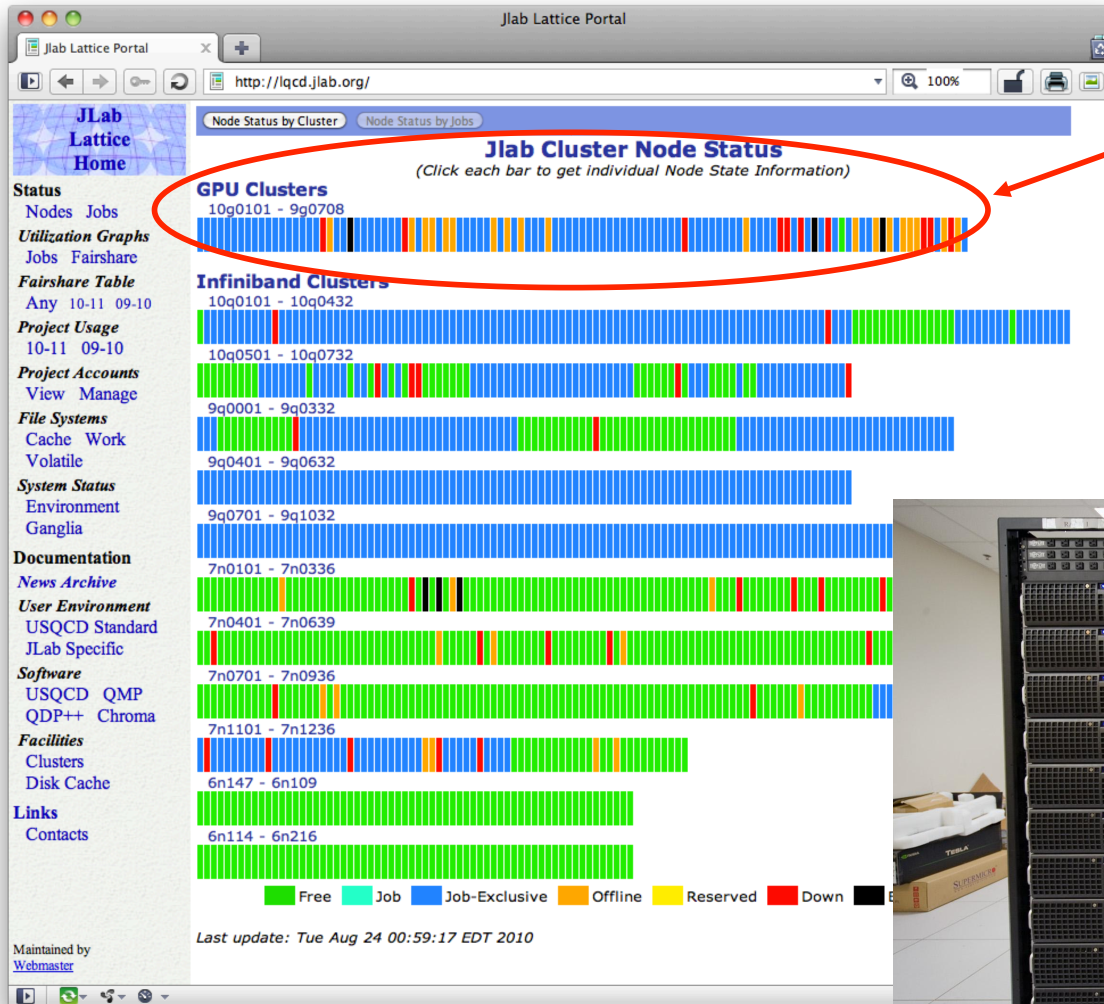
Analogy: Data compression

- In a sense, these various tricks correspond to forms of "data compression." We take advantage of both
  - **"lossless compression":** Eliminate genuine redundancy (e.g., in gauge or spin degrees of freedom), sometimes at the cost of extra computation (as in the SU(3) reconstruction).
  - **"lossy compression":** Throw away some information, but do it strategically (e.g., reduced precision).
- So far, we've applied these ideas only to reduce memory traffic within a node (on a single graphics card), but they're at least as important when parallelizing across nodes/GPUs.

Multi-GPU QUDA

"Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice Quantum Chromodynamics"
(by Ronald Babich, Michael A. Clark, Bálint Joó)
Proceedings of Supercomputing 2010

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

Multi-GPU Motivation

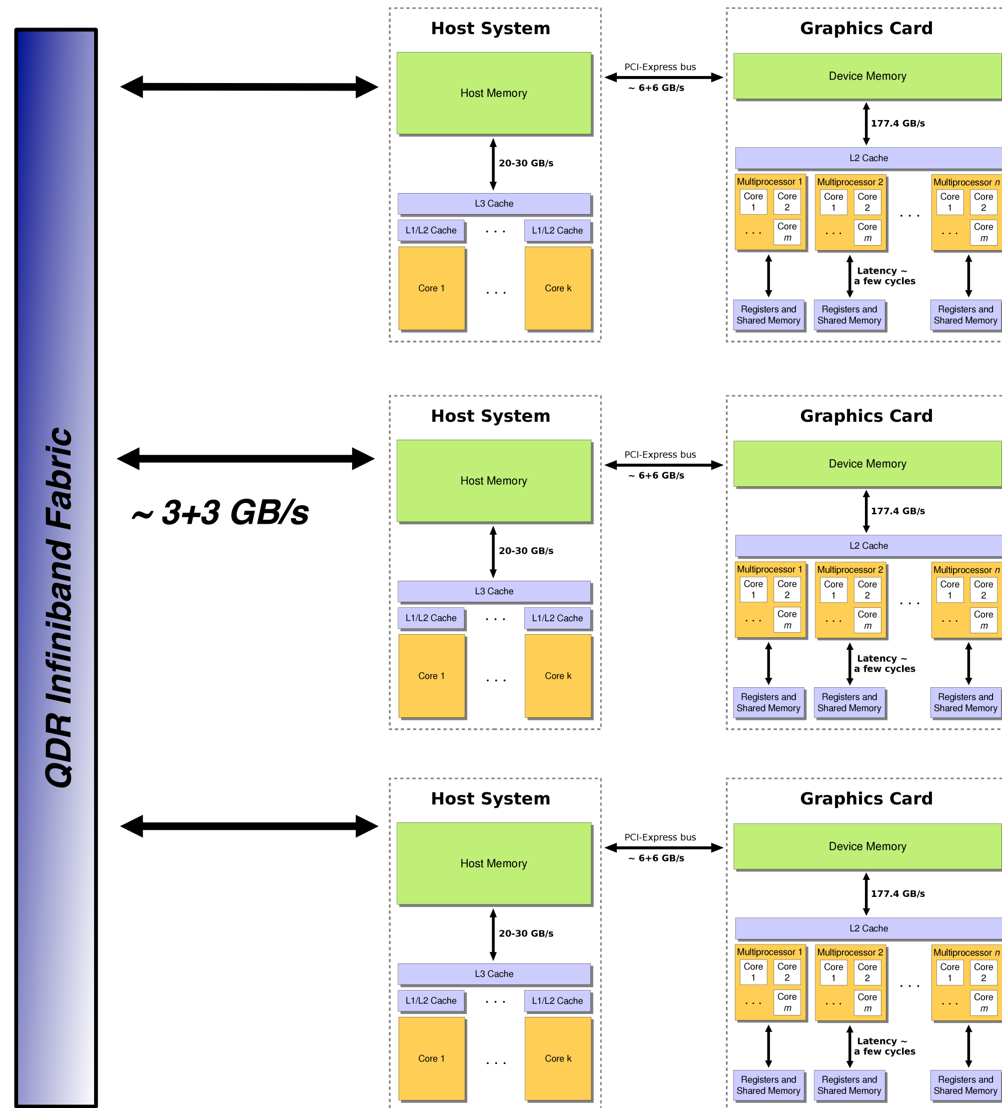
- **GPU memory:** For throughput jobs (e.g., computing propagators), it suffices to use the smallest number of GPUs that will fit the job, but often one GPU isn't enough.

- **Host memory:** It's generally most cost-effective to put more than one GPU in a node. These can be used in an embarrassingly parallel fashion (by running multiple separate jobs), but then host memory becomes a constraint.

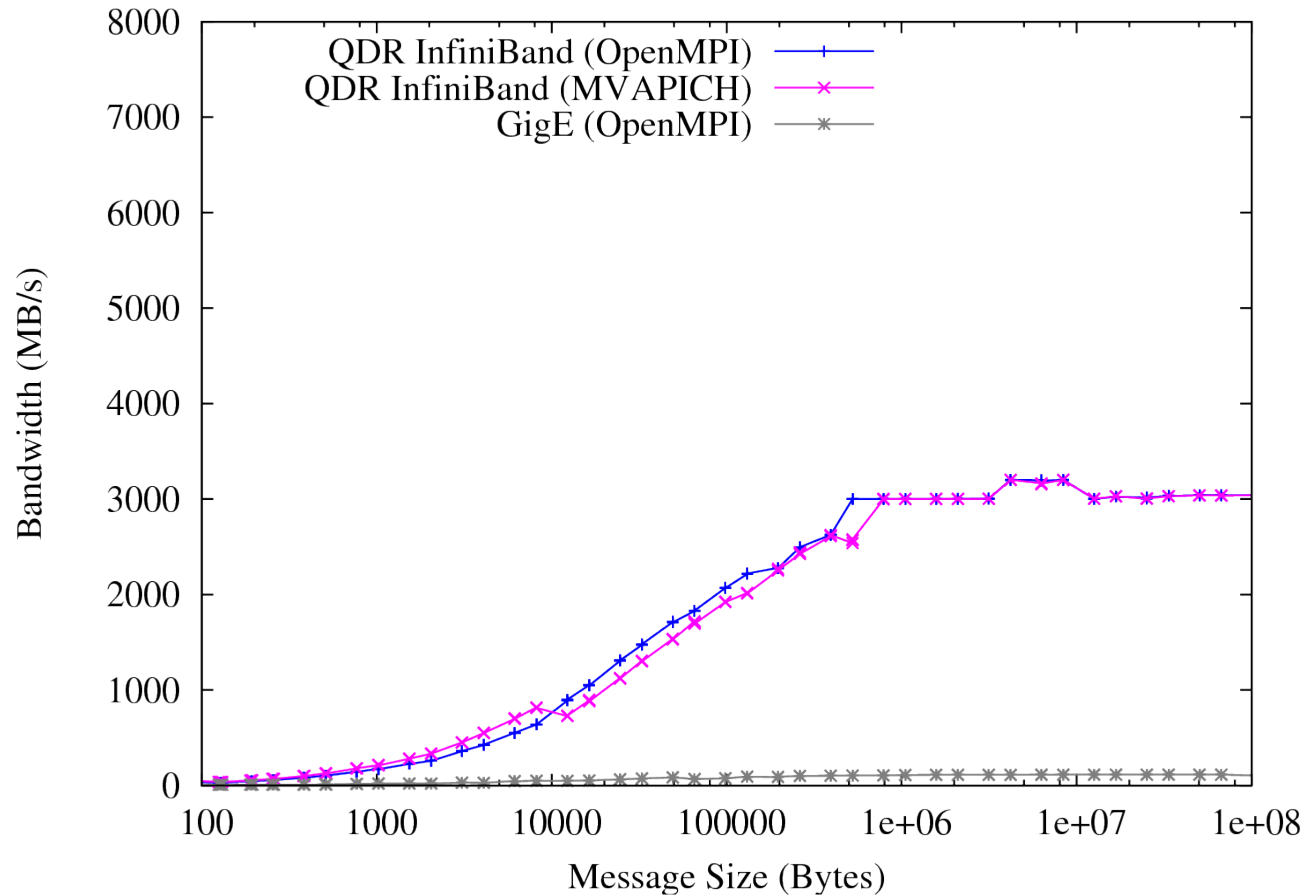
- **Capability:** We'd like to broaden the range of problems to which GPUs are applicable (e.g., gauge generation).

Challenges to scaling up

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

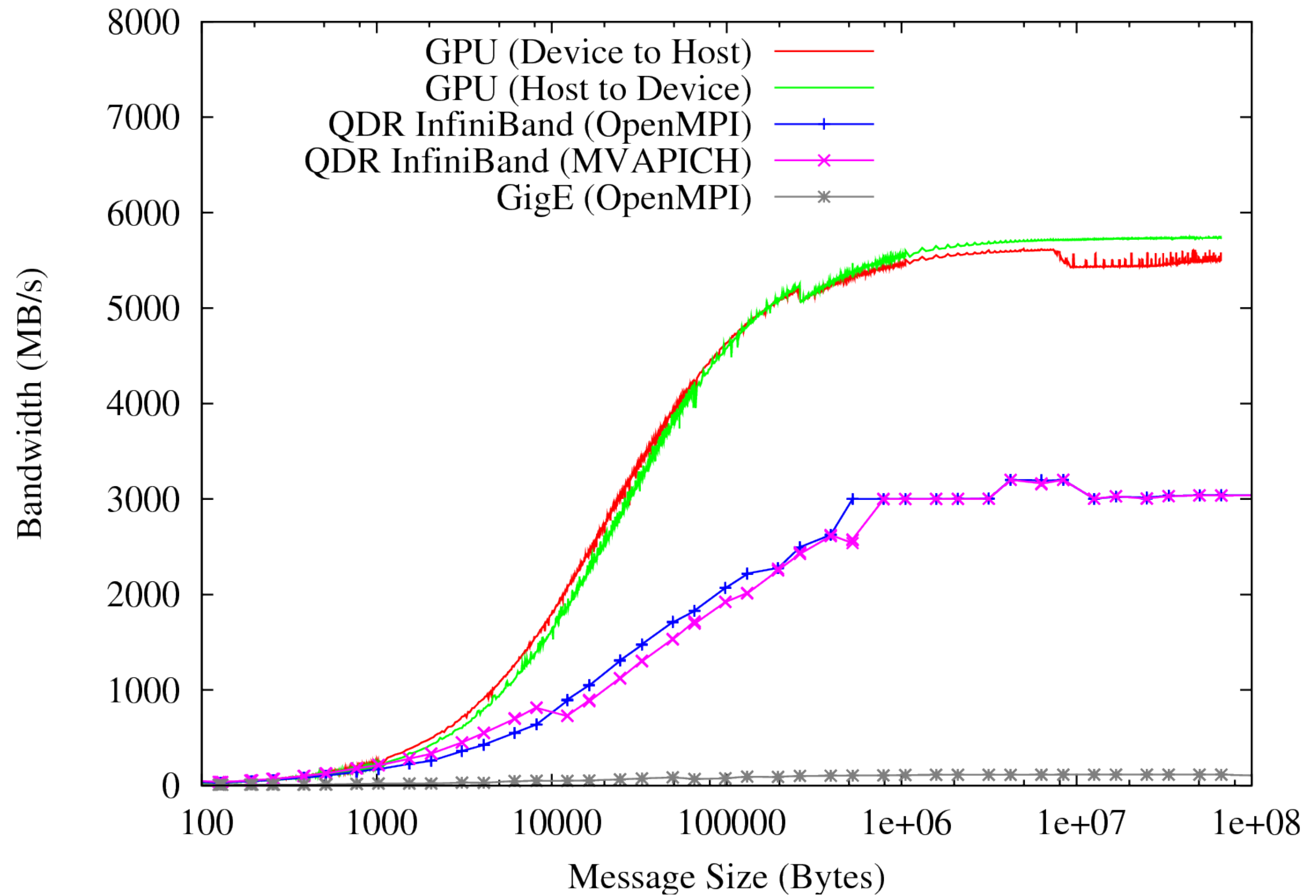


Interconnect bandwidth



(MPI bandwidth as measured by NetPIPE v3.7.1)

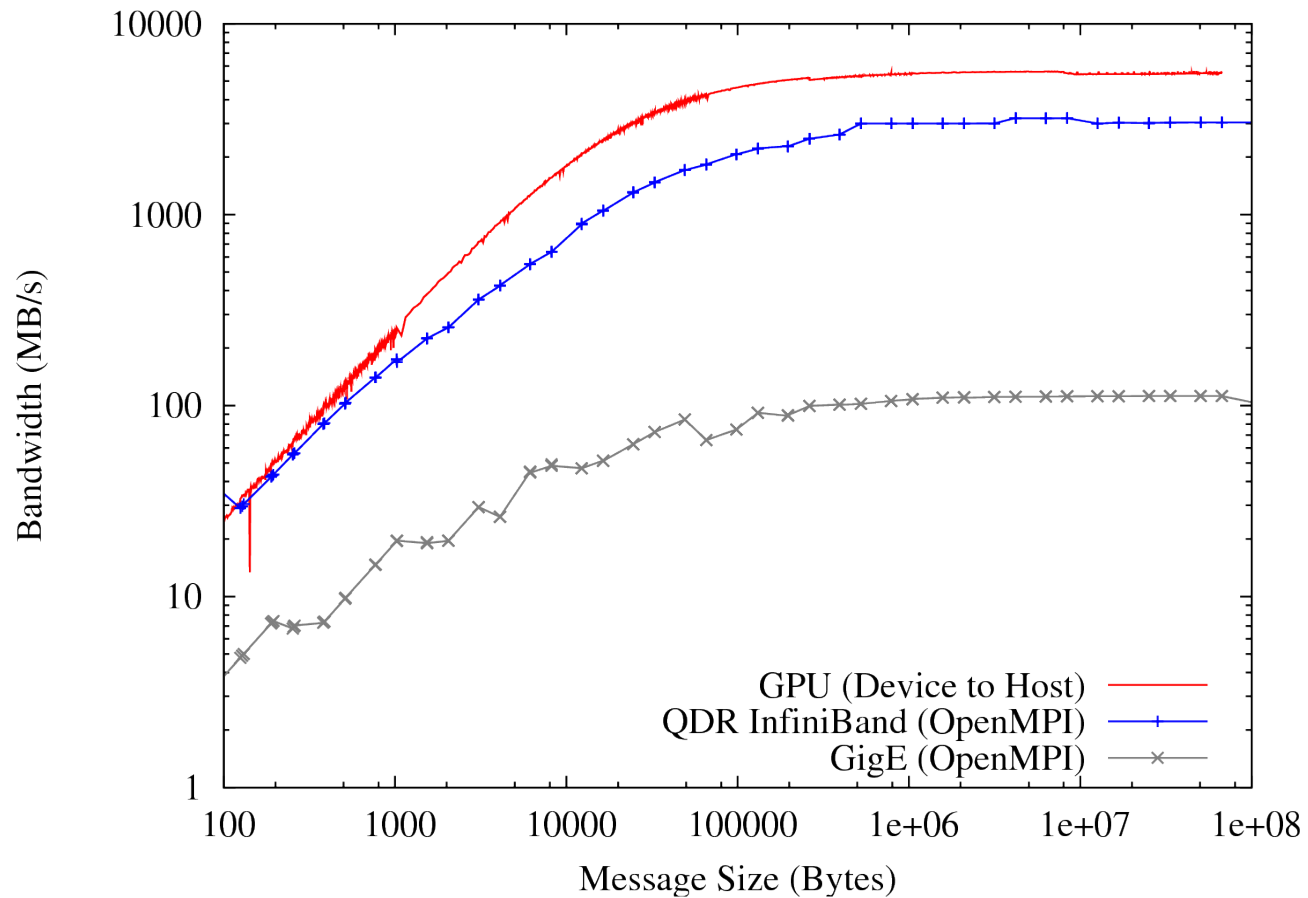
GPU bandwidth



(GPU bandwidth measured with CUDA SDK v2.3)

"bandwidthTest --memory=pinned"

Bandwidth (log scale)



Performance model

- *For the Wilson matrix-vector product, we have:*

$$(1320 \text{ flops/site}) \times (L^4/2 \text{ flops}) = 660L^4 \text{ flops}$$

$$(24/2 \times 4 \text{ bytes/boundary site}) \times (8L^3/2 \text{ sites}) = 192L^3 \text{ bytes}$$

$$\frac{660L^4}{\text{Perf}} = \frac{192L^3}{\text{Bandwidth}}$$

$$\text{Bandwidth [MB/s]} = \frac{0.29(\text{Perf [Mflop/s]})}{L}$$

$$\text{MessageSize [Bytes]} = 24L^3$$

- Inspired by Gottlieb (2000), http://physics.indiana.edu/~sg/pcnets/
- via Homgren (2004), arXiv:hep-lat/0410049

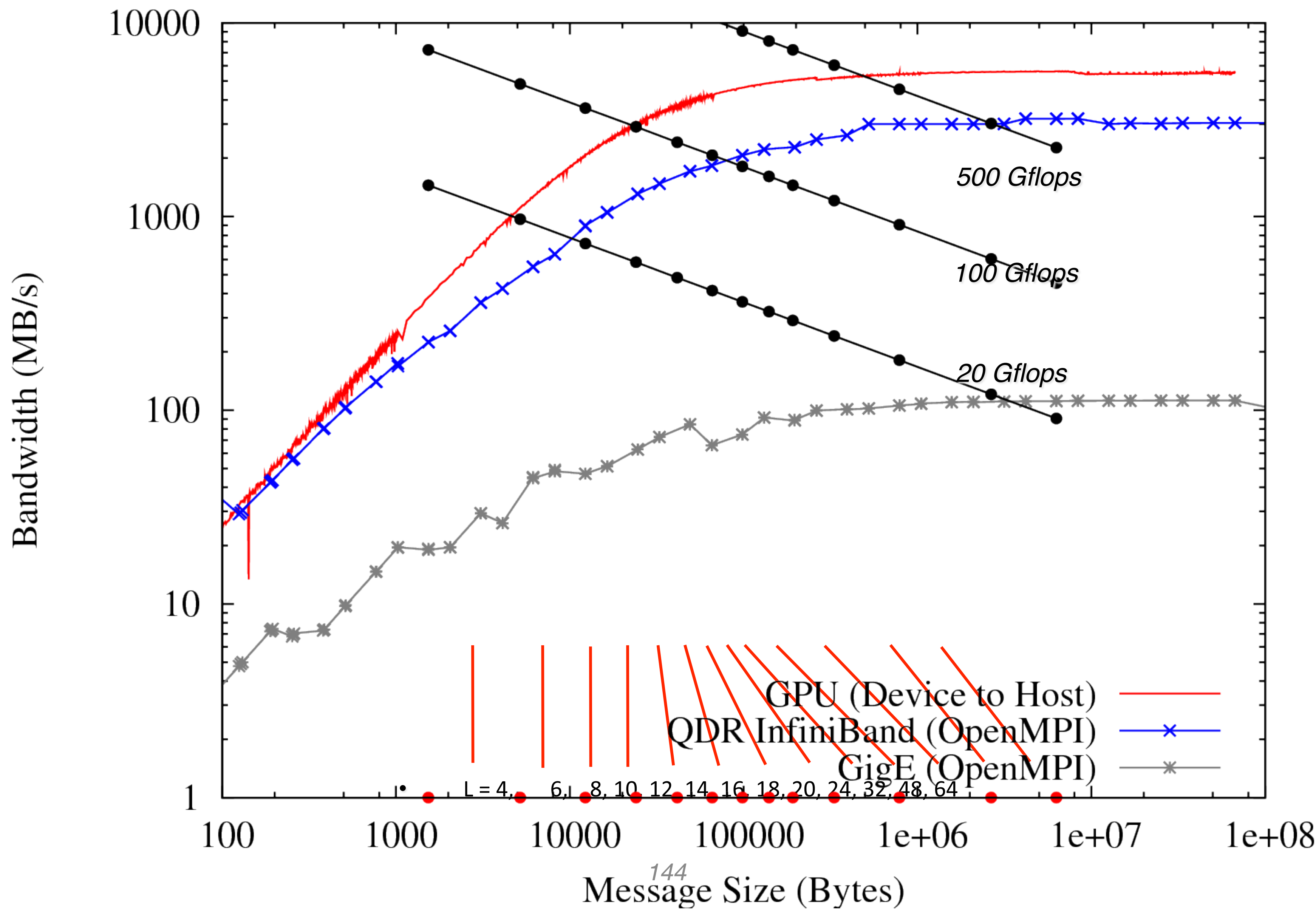
Performance model

- This model is pessimistic in the sense that it assumes we are going to parallelize in all 4 dimensions. For small numbers of nodes, this is never optimal.

- It is optimistic in all other respects (assuming perfect overlapping of communication and computation, for example), telling us the best we can possibly do.

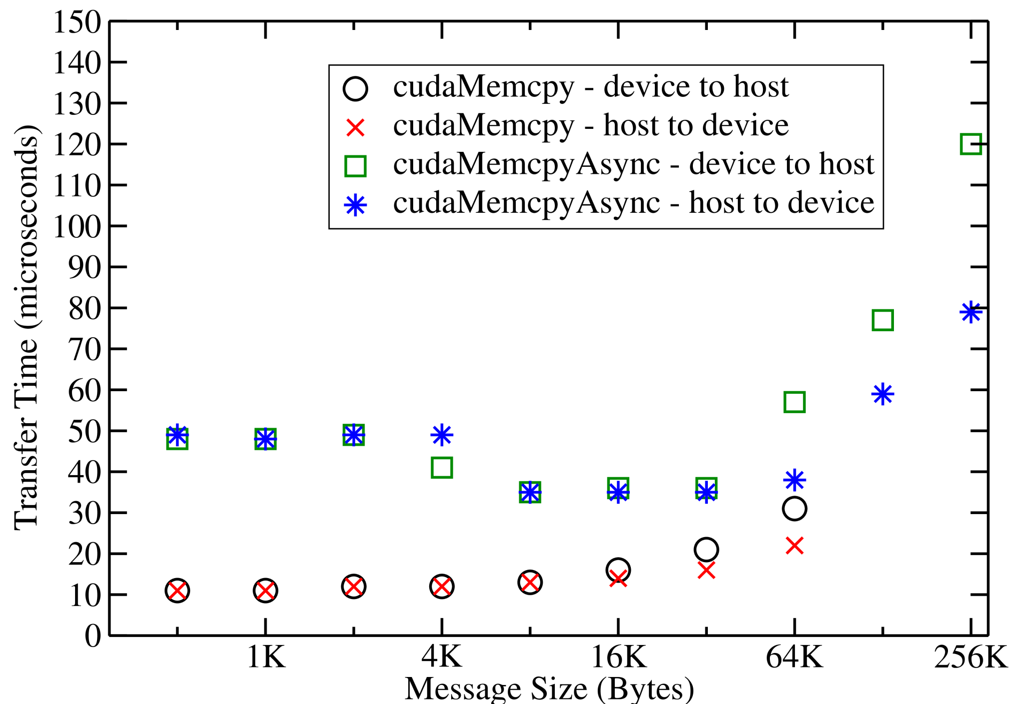
- For this exercise, we're interested in the *strong scaling* regime (smallest possible sub-volumes). How small can we go before the surface/volume ratio limits us?

Required Bandwidth



GPU-host transfer latency

- For reference, end-to-end MPI latency for QDR Infiniband is around 1-2 microseconds.
- QPACE inter-node latency is around 3 microseconds and (anecdotally) necessitated the use of DD-HMC.
- If accurate, the numbers below suggest > 20 microseconds to transfer a byte from one GPU to another.



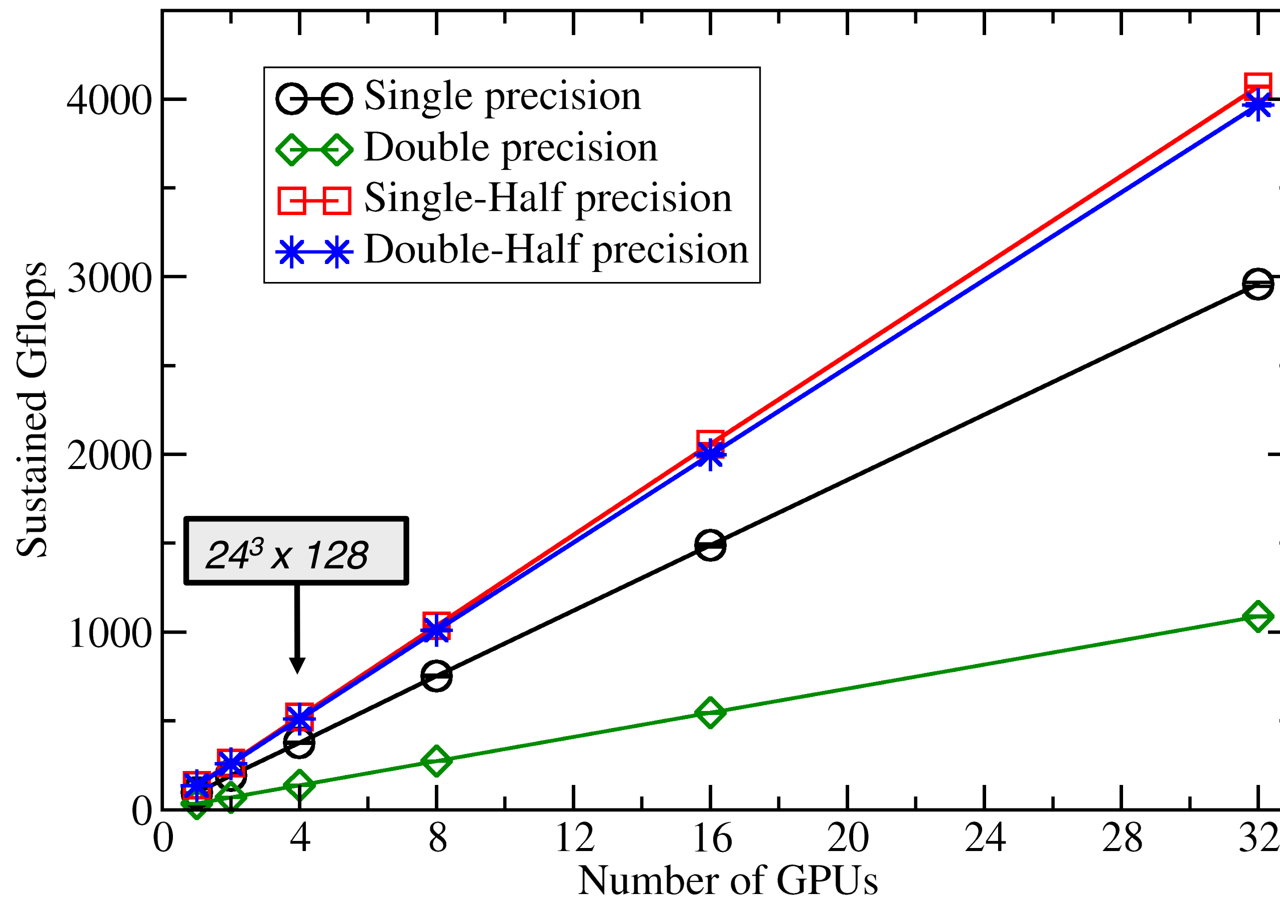
Multi-GPU results

- All performance numbers are for the full inverter (BiCGstab, anisotropic clover-improved Wilson with "symmetric" even/odd preconditioning).
- Tests were run on a 16-node cluster at Jefferson Laboratory, interconnected by QDR Infiniband.

- *Each node has 2 GeForce GTX 285 cards (previous generation; 240 cores/GPU).*

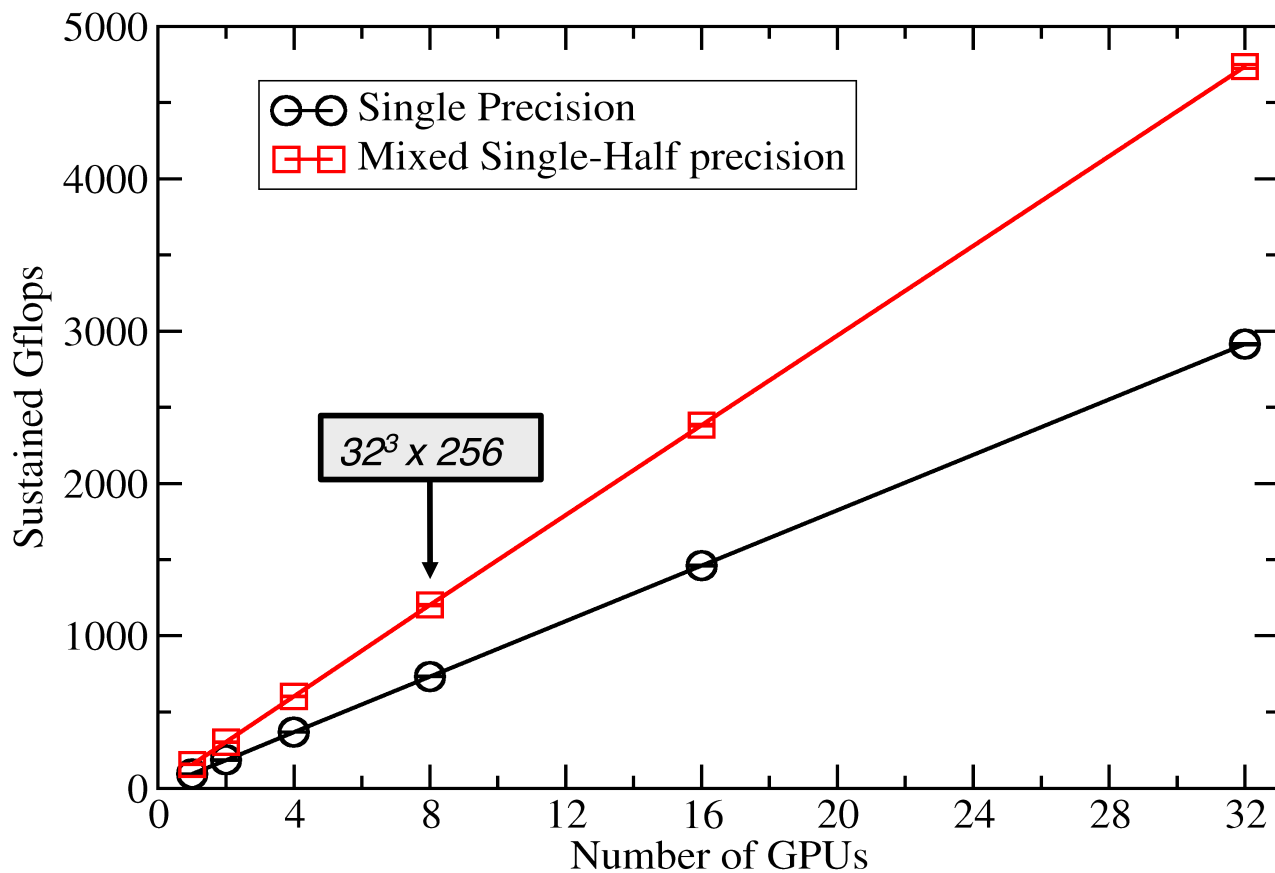
Weak scaling (24^3 x 32 local)

- Local volume (per GPU) is held fixed: 24^3 x 32



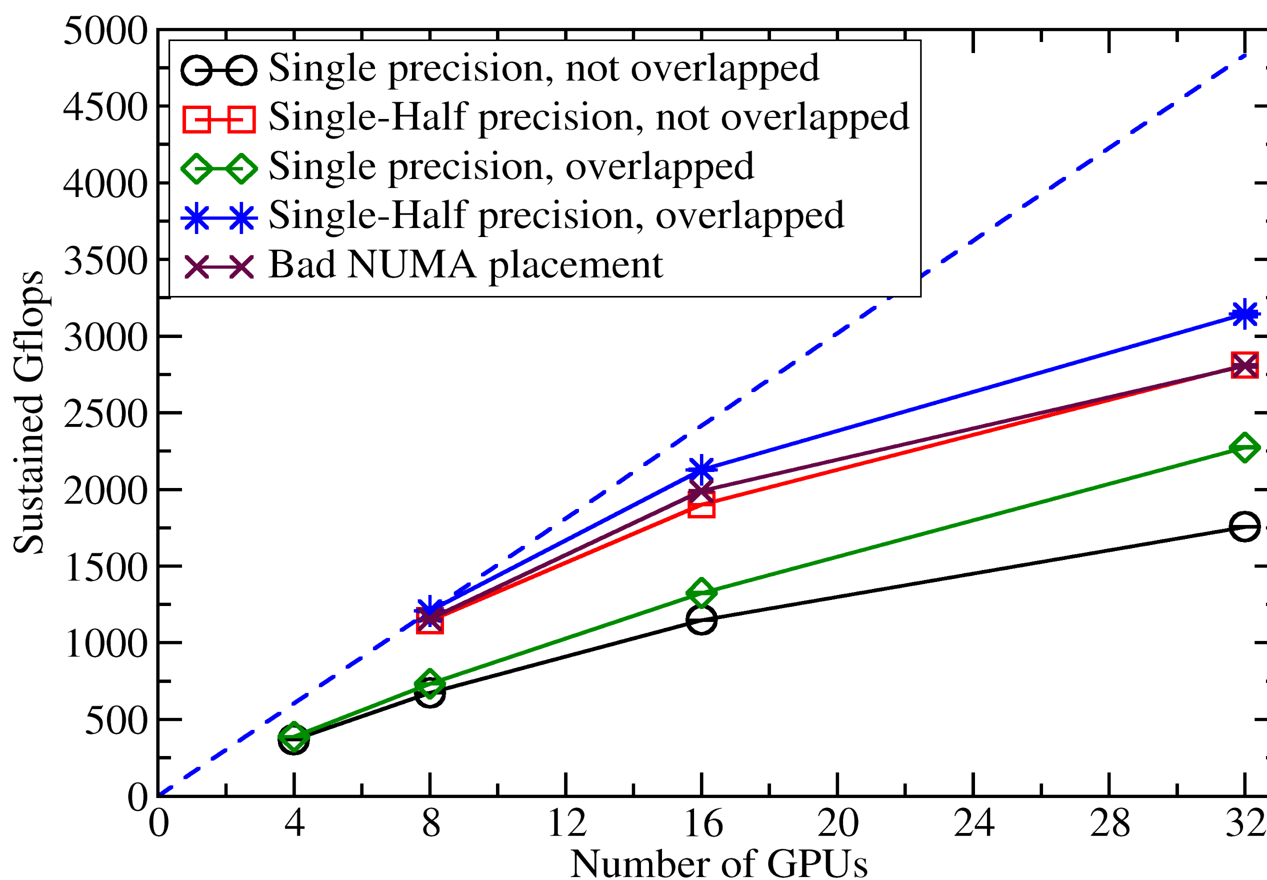
Weak scaling (32^4 local)

- Local volume (per GPU) is held fixed: 32^4



Strong scaling (32^3 x 256)

- Total volume is held fixed: 32^4 x 256



First multi-GPU results on Fermi

- 1 node (Dual-socket/dual-chipset)
- 4 NVIDIA GeForce GTX 480 cards
- Code has not been particularly optimized.
- Sustained performance in the inverter (BiCGstab, clover-improved Wilson, mixed single/half):

1020 Gflops

Retrospective

Each use should be accompanied by the credit line and notice, "Courtesy of International Business Machines Corporation. Unauthorized use not permitted." Copying of images for further distribution or commercial use is prohibited without the express written consent of IBM.

- **2004**: First 1 Tflops sustained for QCD (P. Vranas)
  - 1 rack Blue Gene/L
  - ~ $1M in 2005 or 2006

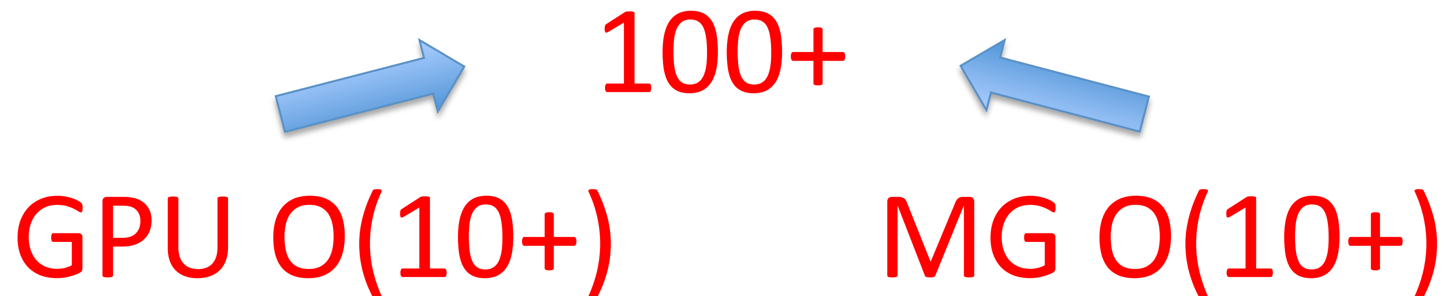
2010: 1 Tflops sustained, under your desk
  - Dual-socket node with 4 GPUs
  - ~ $5k (200x improvement in price/performance)

... for problems that fit.

(1 rack BG/L has 512 GB RAM vs. 6 GB for four GTX 480s)

New Project: MG on GPU

- Cost in $s reduced by a factor of at least



New apps:

1. Higgs-Nucleon coupling for Dark Matter detection
2. Beyond the Standard Model results...
3. Nuclear excitations and interactions etc....
4. GPU-MG for graphene and solid modeling

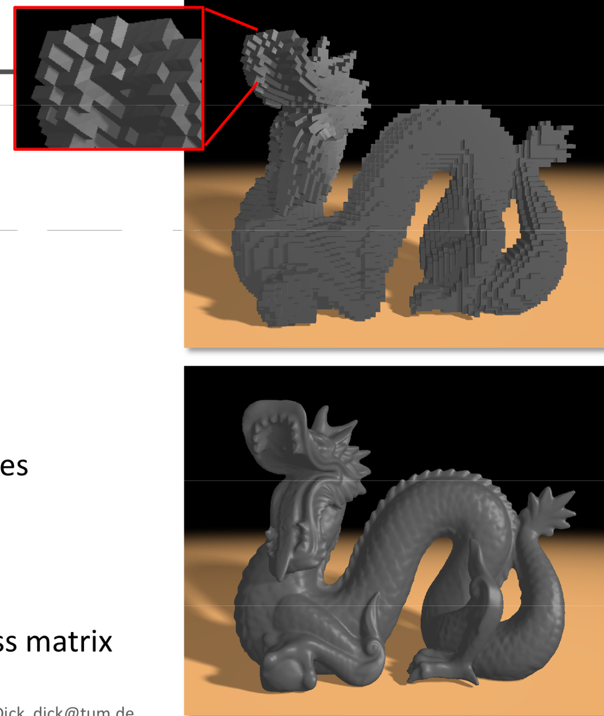
CUDA for Real-Time Multigrid Finite Element Simulation of Soft Tissue Deformations

Christian Dick

Computer Graphics and Visualization Group
Technische Universität München, Germany

Our Approach

- Hexahedral (tri-linear) finite elements on a uniform Cartesian grid
- Linear elasticity, co-rotated strain
- Geometric multigrid solver
- CUDA API to flexibly access all resources on the graphics card
- Advantages:
  - Numerical stencil of regular shape enables efficient GPU implementation
  - FE model and multigrid hierarchy generation is easy and fast
  - Only one pre-computed element stiffness matrix (greatly reduces memory requirements)



Conclusions

- **Status:**
  - GPUs clearly win for many analysis jobs (5-10x improvement in price/performance)
  - … but multigrid on traditional clusters is competitive (up to 20x over standard solvers at light masses).
  - Next step: **MG^2:** Multi-GPU multigrid (up to 100x?).
  - Large-scale gauge generation is a hard problem.
- **Lessons:**
  - The future is full of hard problems.
  - The trend is toward huge floating point performance, but relatively anemic memory bandwidth.
  - Inter-node communication is an even greater challenge.

Broader impact

- Initial target applications in Lattice Field Theory and Computational Fluid Dynamics.
- But also to be a catalyst for local researchers in many fields to explore GPU architectures and share experience and methods.
- Access to faculty, post docs, graduate and undergraduate students in nano technology, chemistry, biological modeling, medical imaging, etc.
- Educational context and impact is crucial advantage of university based experimental GPGPU cluster!