Attaining multi-teraflops performance for QCD on GPUs

Ronald Babich
Boston University

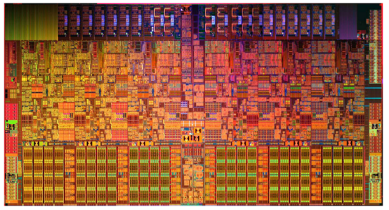
Lattice 2010 - Villasimius
June 18, 2010

with
Kip Barros, Rich Brower, Mike Clark,
Bálint Joó, and Claudio Rebbi

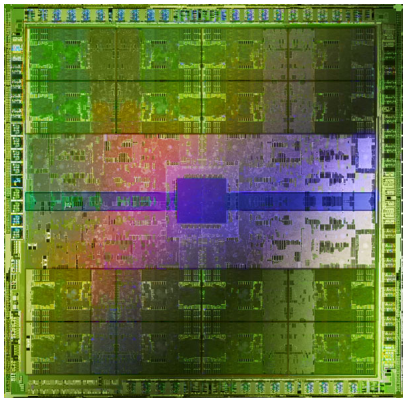
Outline

- A bit about GPUs
- General performance strategies
- Multi-GPU strategy
- Multi-GPU performance results
- Outlook

A tale of two processors



"Gulftown"



"Fermi"

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)

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)

Bandwidth constraints

- 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.

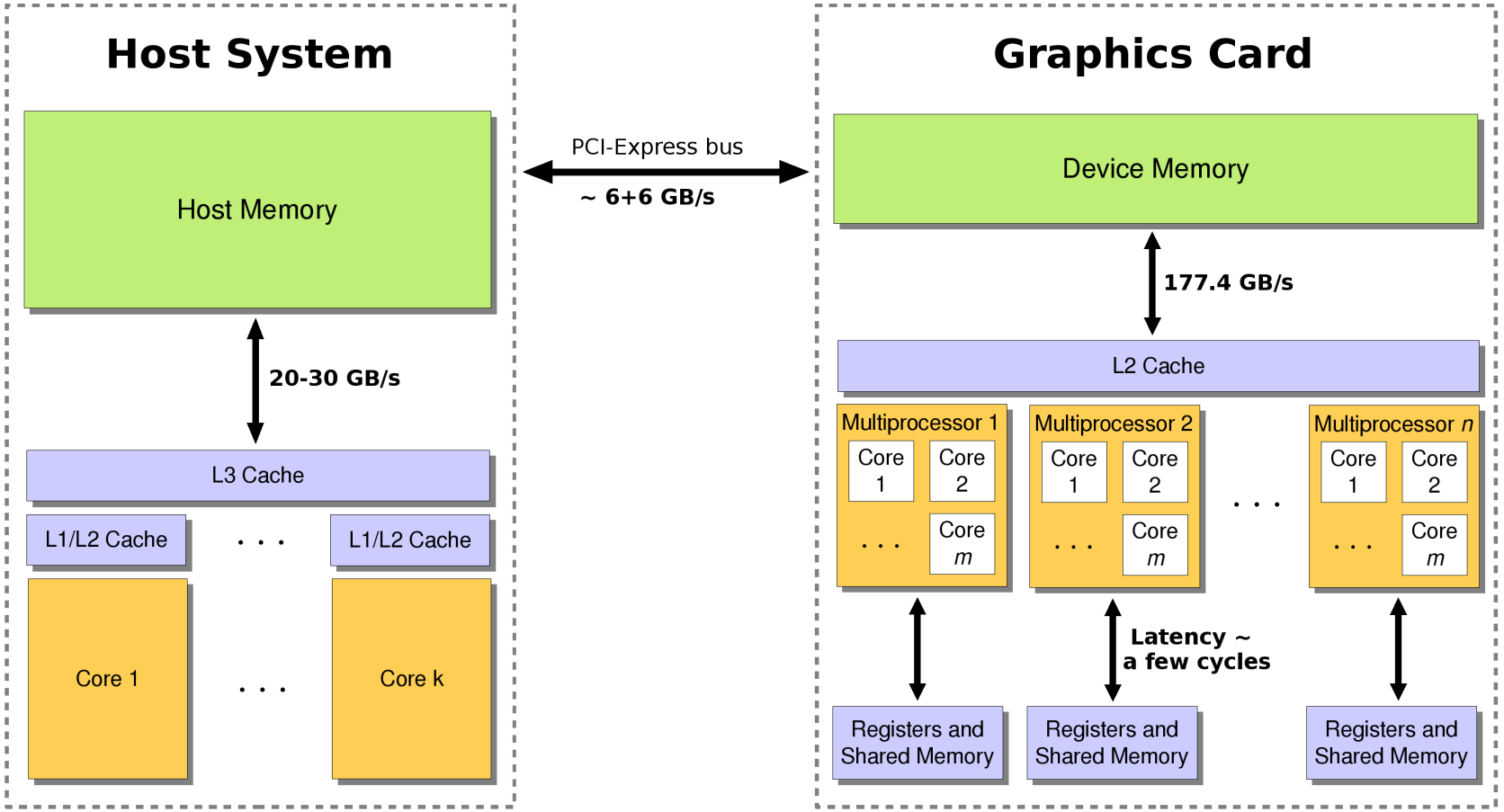
160 Gflops (SP)

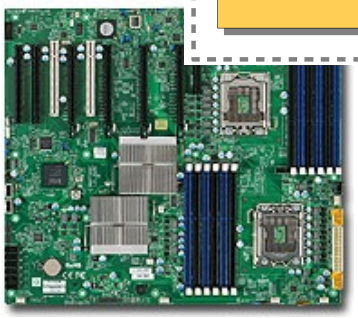
32 GB/s memory bandwidth

1345 Gflops (SP)

177 GB/s memory bandwidth

GPU memory hierarchy





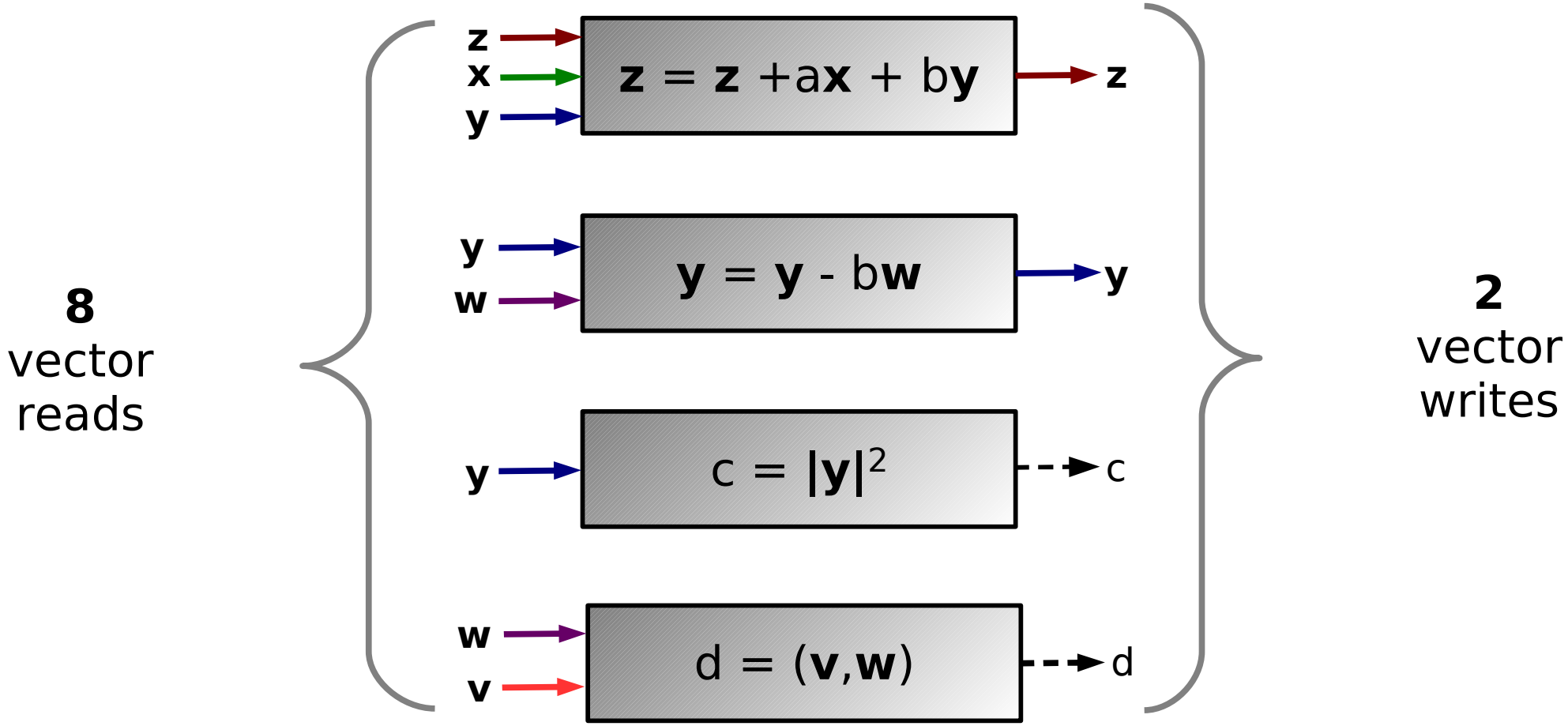
General strategies

- Follow the standard best practices (minimize PCI-E transfers, pay attention to thread occupancy, ensure memory coalescing, avoid bank conflicts, avoid partition camping, etc.).
- Reduce memory traffic:
  - Reconstruct SU(3) matrices from 8 or 12 real numbers on the fly.*
  - Choose a gamma basis with γ_4 diagonal.*
  - Fix to the temporal gauge.*
- Use multi-precision solvers, including half (16-bit) precision.*
- Take advantage of “kernel fusion.”
- Auto-tune kernel launch parameters (e.g., to optimize BLAS).

* see Mike Clark's Lattice 2009 plenary

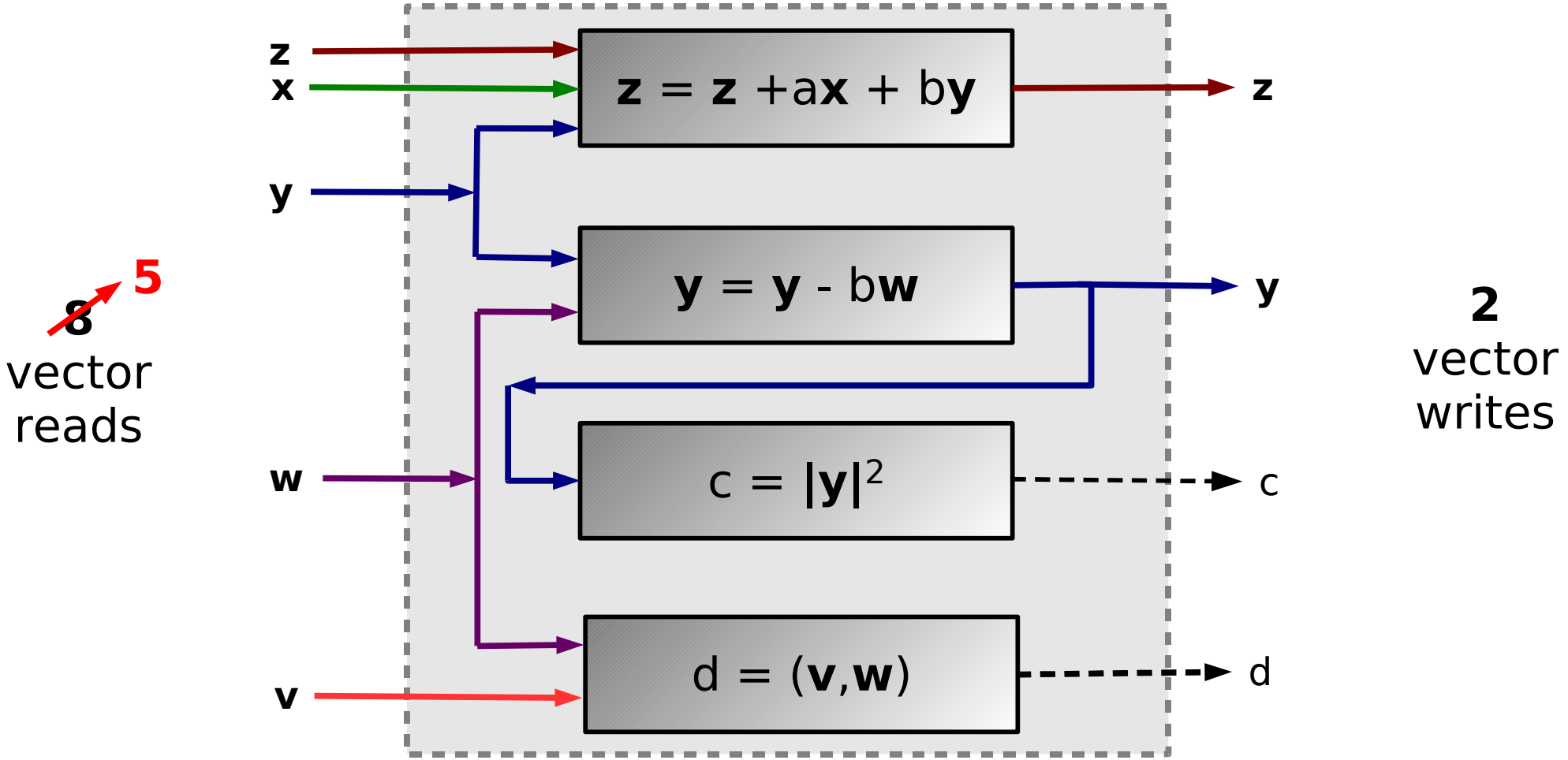
Kernel fusion

- Consider the following set of operations taken from our BiCGstab solver:



Kernel fusion

- We can avoid memory transfers by fusing these operations into a single compute kernel:

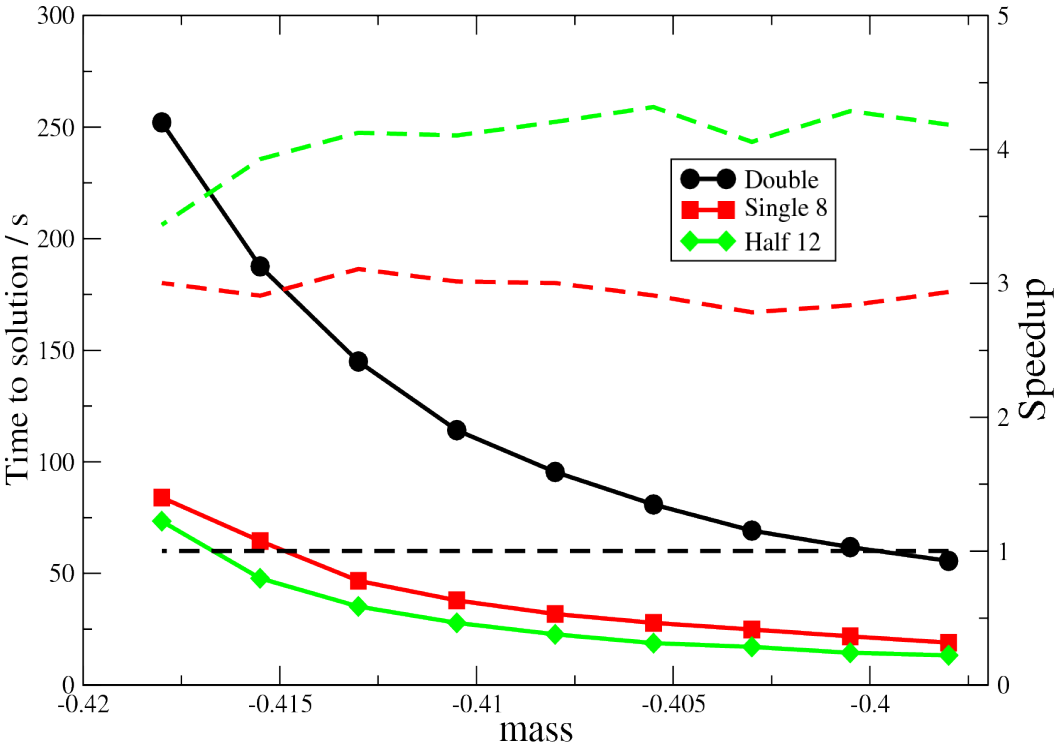


Auto-tuned linear algebra

```
$ make
...
$ make tune
...

Benchmarking 16 bit precision
copyCuda                          : 256 threads per block, 2048 blocks per grid, Gflops/s = 0.000000, GiB/s = 127.606472
axpbyCuda                         :  64 threads per block, 2048 blocks per grid, Gflops/s = 62.037775, GiB/s = 125.183891
xpyCuda                           : 256 threads per block,  512 blocks per grid, Gflops/s = 20.661412, GiB/s = 125.075855
axpyCuda                          :  64 threads per block, 2048 blocks per grid, Gflops/s = 41.360739, GiB/s = 125.190617
xpayCuda                          :  64 threads per block, 2048 blocks per grid, Gflops/s = 41.375916, GiB/s = 125.236556
mxpyCuda                          :  64 threads per block, 2048 blocks per grid, Gflops/s = 20.686066, GiB/s = 125.225099
axCuda                            :  64 threads per block, 2048 blocks per grid, Gflops/s = 31.444969, GiB/s = 126.903442
caxpyCuda                         :  64 threads per block, 2048 blocks per grid, Gflops/s = 82.751603, GiB/s = 125.236209
caxpbyCuda                        : 128 threads per block, 2048 blocks per grid, Gflops/s = 145.006273, GiB/s = 125.401357
cxpaypbzCuda                      : 256 threads per block, 1024 blocks per grid, Gflops/s = 125.884968, GiB/s = 127.009472
axpyZpbxCuda                      : 128 threads per block, 2048 blocks per grid, Gflops/s = 101.000918, GiB/s = 127.378922
caxpbypzYmbwCuda                  :  64 threads per block, 4096 blocks per grid, Gflops/s = 120.062473, GiB/s = 121.134966
sumCuda                           : 256 threads per block,  256 blocks per grid, Gflops/s = 51.459132, GiB/s = 103.837612
normCuda                          : 256 threads per block,  256 blocks per grid, Gflops/s = 103.021799, GiB/s = 95.946527
reDotProductCuda                  : 128 threads per block,  256 blocks per grid, Gflops/s = 59.747498, GiB/s = 120.562419
axpyNormCuda                      : 256 threads per block, 2048 blocks per grid, Gflops/s = 74.018444, GiB/s = 112.019453
xmyNormCuda                       : 256 threads per block, 4096 blocks per grid, Gflops/s = 55.737687, GiB/s = 112.471159
cDotProductCuda                   : 128 threads per block,  256 blocks per grid, Gflops/s = 119.348463, GiB/s = 120.414577
xpaycDotzyCuda                    : 256 threads per block, 2048 blocks per grid, Gflops/s = 85.237100, GiB/s = 114.664674
cDotProductNormACuda              : 128 threads per block,   64 blocks per grid, Gflops/s = 173.619070, GiB/s = 116.779982
cDotProductNormBCuda              : 128 threads per block,   64 blocks per grid, Gflops/s = 173.822401, GiB/s = 116.916746
caxpbypzYmbwcDotProductWYNormYQuda: 256 threads per block,  512 blocks per grid, Gflops/s = 145.992303, GiB/s = 114.563884

Benchmarking 32 bit precision
copyCuda                          :  64 threads per block, 4096 blocks per grid, Gflops/s = 0.000000, GiB/s = 126.151752
...

Benchmarking 64 bit precision
copyCuda                          : 256 threads per block, 4096 blocks per grid, Gflops/s = 0.000000, GiB/s = 125.865711
...

Writing optimal parameters to blas_param.h
make[1]: Leaving directory `/home/rbabich/quda/tests'
Autotuning completed successfully.  Please type 'make' to rebuild library.

$ make
...
```

Mixed precision with reliable updates

- Using a mixed-precision solver incorporating “reliable updates” (Clark et al., arXiv:0911.3191), single/half or double/half results in only a 10-20% increase in iteration count as compared to pure single or pure double, respectively.
- **Example single-GPU performance:** 190 Gflops sustained in BiCGstab for Wilson-clover with mixed single/half precision on a GeForce GTX 285. (last-gen., 240 cores/GPU)

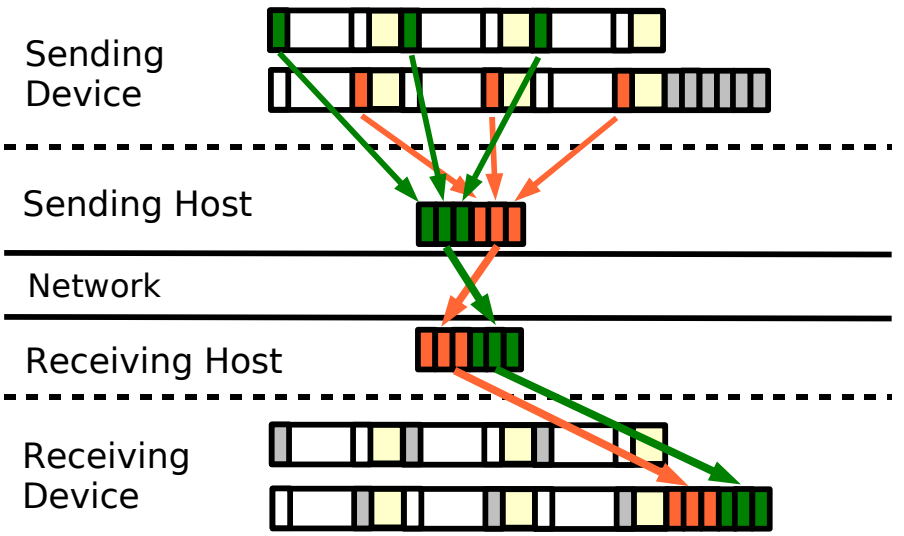


Multi-GPU motivation

- **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).

Multi-GPU strategy

- In this first pass, we divide up the temporal direction only.
- We must contend with the fact that the spinor field is stored in 6 separate arrays (necessary to ensure memory coalescing).
- With our choice of gamma basis, we need only transfer half the spin components (e.g., upper in the backward direction).
- The 3 sub-arrays containing these components on the boundary time-slice are copied into a contiguous buffer on the host.
- The buffer is then transferred across the network to the remote host, where it is copied onto the remote GPU.
- We use CUDA streams and *cudaMemcpyAsync()* to overlap boundary transfers with interior computation.

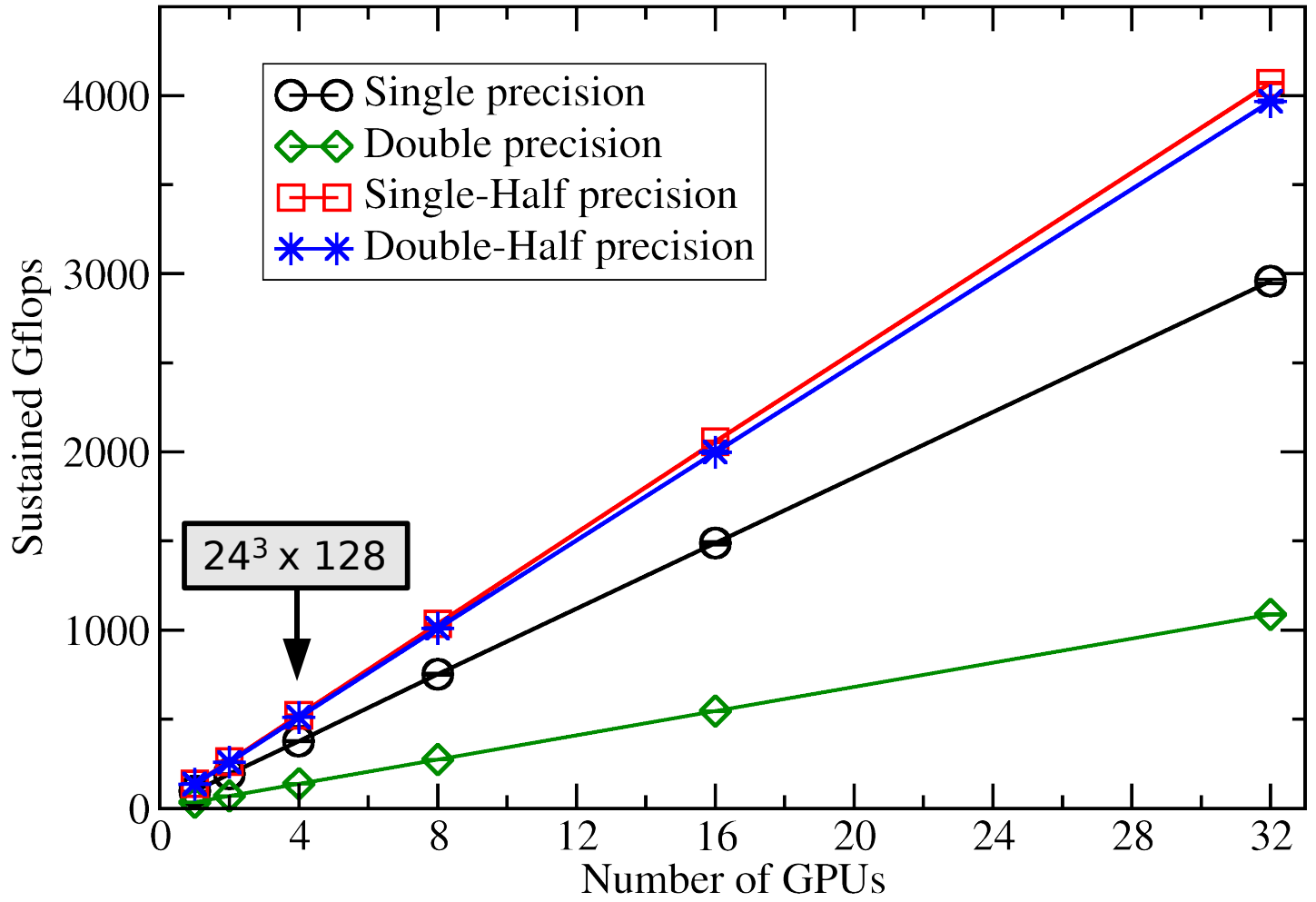


Multi-GPU results

- 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).
- Details are presented in submission to SC'10 (R.B., M. Clark, B. Joó).

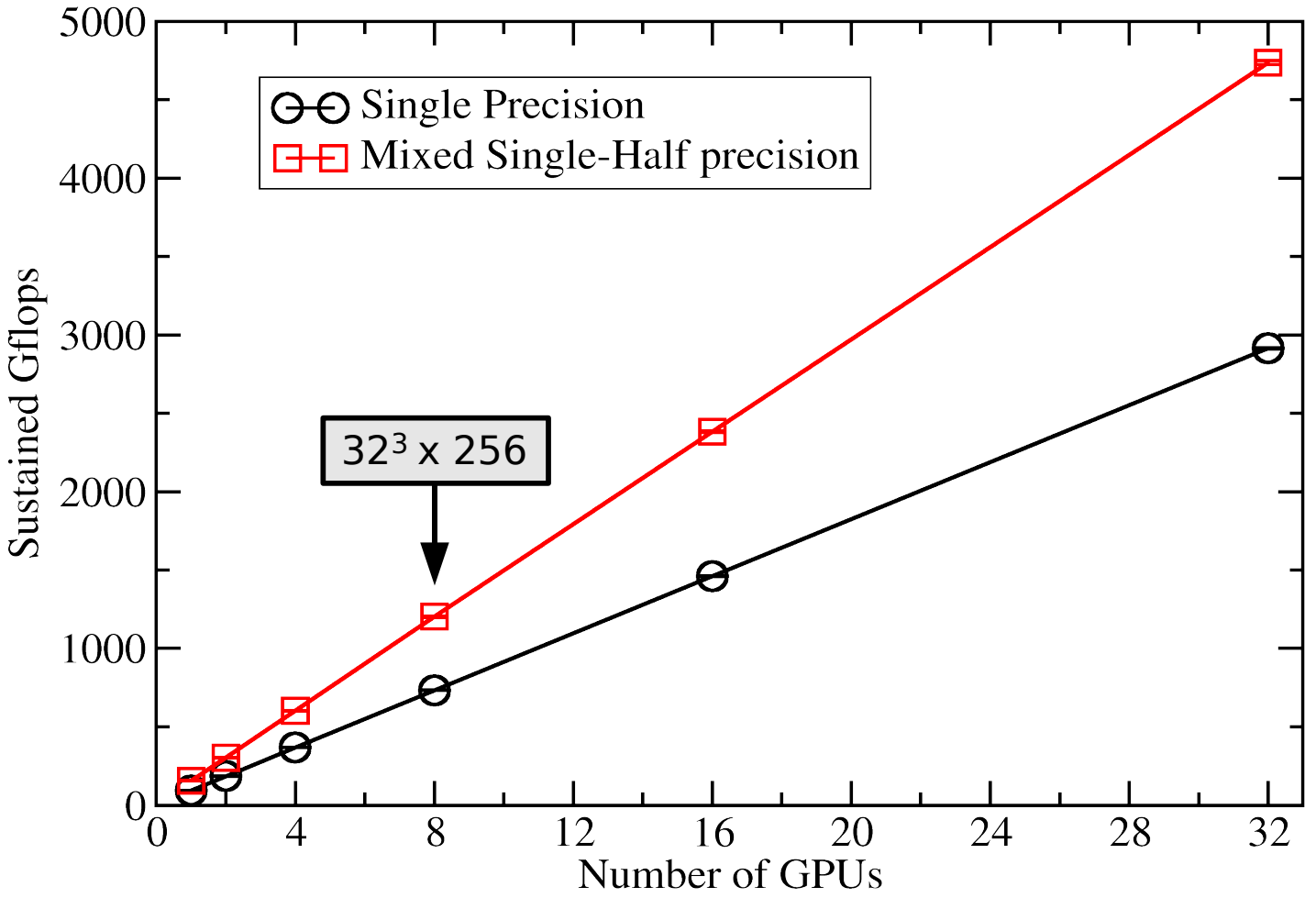
Weak scaling (24^3 x 32 local)

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



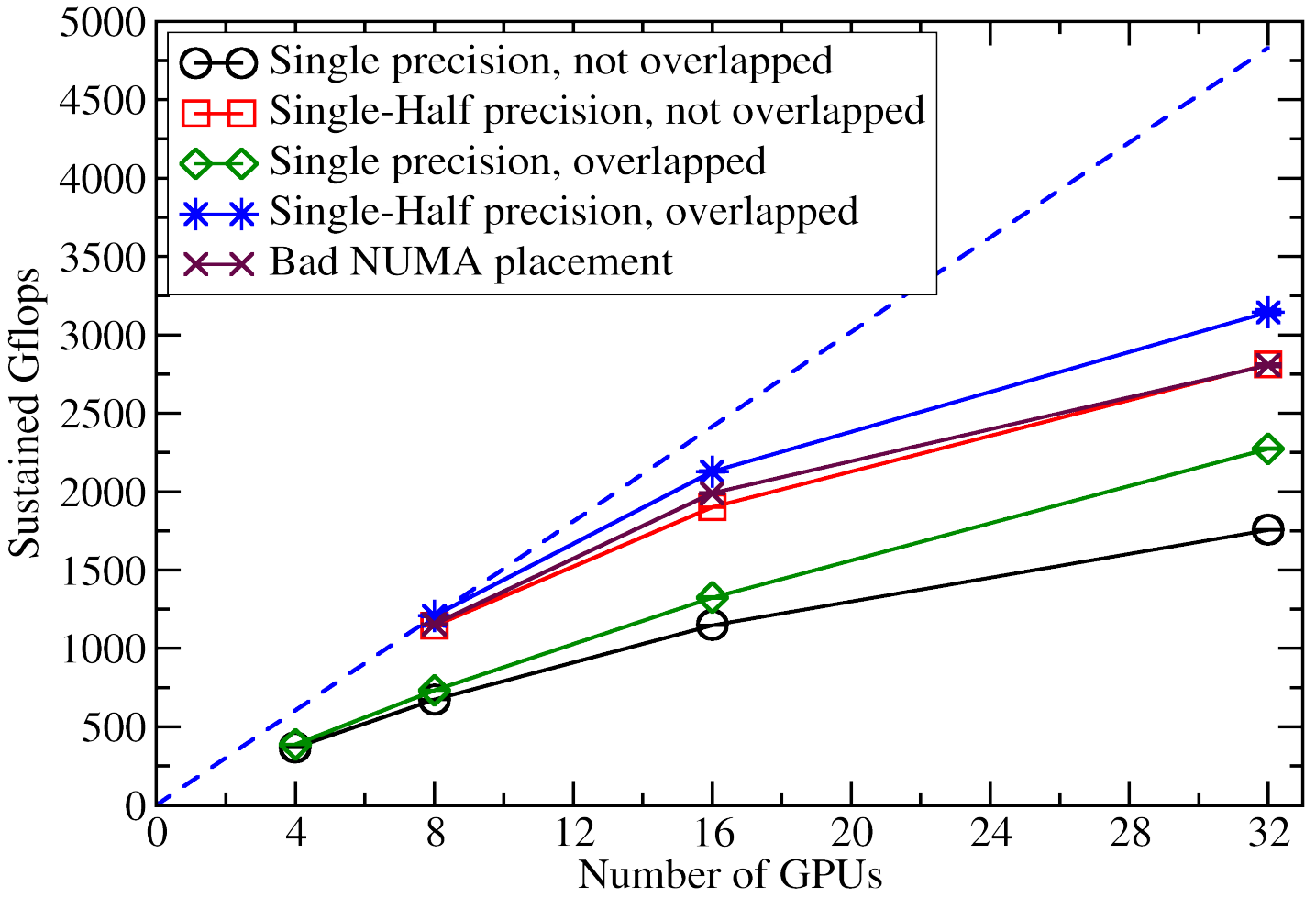
Weak scaling (32^4 local)

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



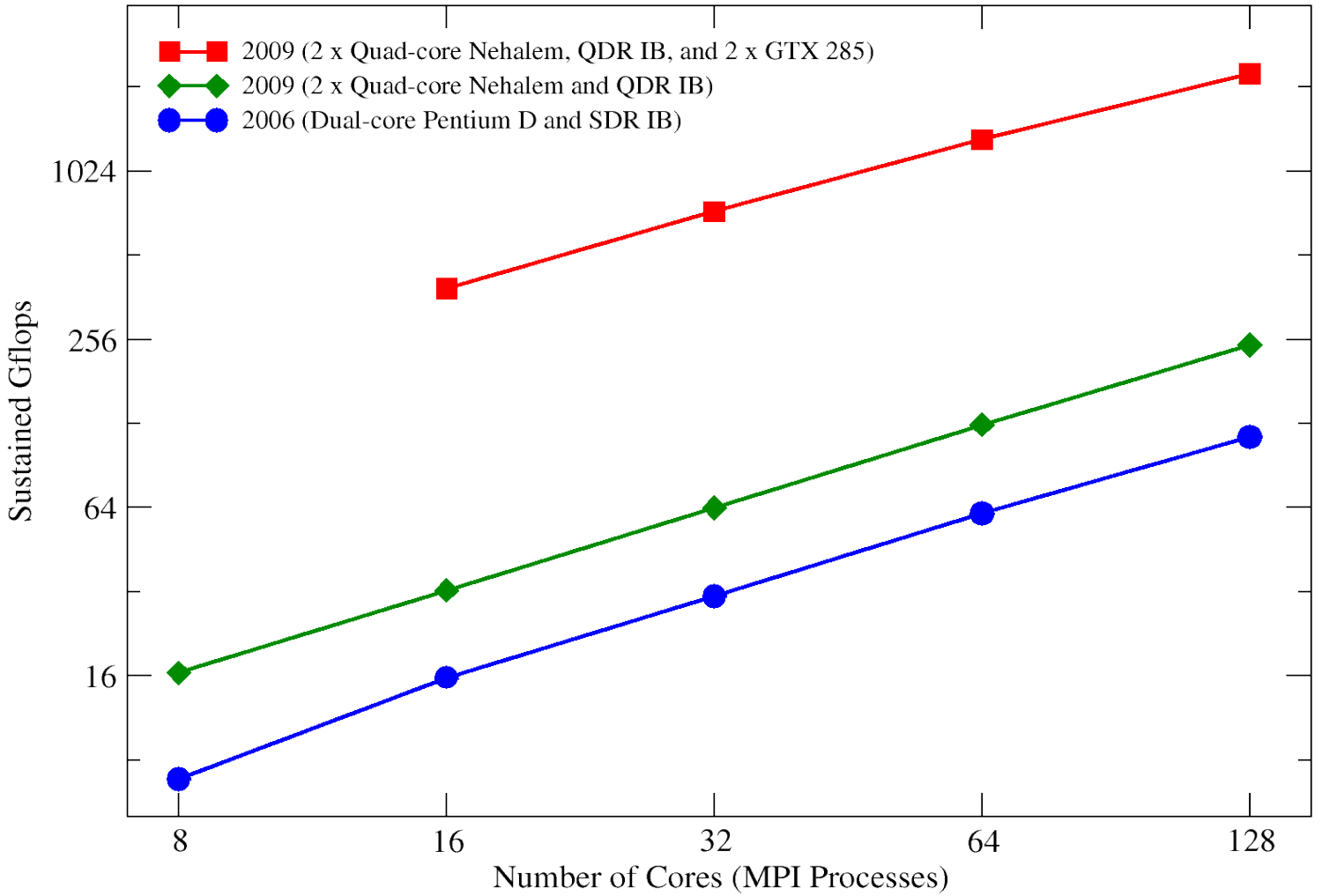
Strong scaling (32^4 x 256)

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



Cluster comparison (32^4 x 256)

- What happens if we take a modern cluster and add GPUs?



First results on Fermi

- 1 node (Dual-socket/dual-chipset)
- 4 NVIDIA GeForce GTX 480 cards
- Code has not been optimized for Fermi, aside from setting L1 cache to 48 KB and the usual auto-tuning.
- Sustained performance in the inverter (BiCGstab, clover-improved Wilson, mixed single/half):

1020 Gflops

Retrospective

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)

QUDA

- 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 (see next talk).
- Domain wall (contributed by Joel Giedt), twisted mass, and multi-GPU support will be available soon.
- For efficiency, QUDA natively supports various host-side data layouts (Chroma/QDP++, QDP/C, CPS). Others can be added upon request.