LQCD on GPUs
(from a hardware perspective)

Outline

- Quick overview of GPUs &
  how they address (some) LQCD requirements
- LQCD performance on GPUs
- Performance constraints
  - Amdahl's Law constraints
  - I/O limitations
- Future accelerator architectures

Modern GPUs

Characteristics of GPUs:

- Lots of simple cores with fast context switching and many contexts per core to hide memory latency
- SIMD architecture (single instruction, multiple data)
- High memory bandwidth
- Complex memory hierarchy, to achieve high bandwidth at low cost

Commodity Processors	**CPU**	**NVIDIA Fermi GPU**
#cores	10	512
Clock speed	~3 GHz (2.53 best flops/$)	1.5 GHz
Main memory bandwidth (streams)	37 GB/s (2.53 / 1066)	177 GB/s
I/O bandwidth for scaling	7 GB/s (dual QDR IB)	11 GB/s (bidirectional on PCIe)
Power	80 watts	225 watts

GPU Best Fit

1. High data parallelism
   – Perform the same set of operations on many data items

2. High flop count on a small kernel
   – Problem needs to fit into the GPU (1 - 6 Gbytes) (or multi-GPU if communications are modest)
   – Has little data dependent branching, no recursion
   – Needs to do enough work to amortize cost of pushing the problem into the GPU and getting the results back (sometimes it is possible to pipeline the problem and data)

3. High memory bandwidth requirements
   – But problem doesn't fit into CPU cache

Quick Look: LQCD on GPUs

Lattice QCD is very memory bandwidth intensive

- 1 instruction per byte of memory touched
- Size of lattice problem ~1000 x CPU cache memory size
- Large faction of work is in solving for a matrix inverse (the inverter)
- Flops (sustained) is roughly memory bandwidth (single precision)
  - Dual Nehalem: Streams 37 GB/s,
    LQCD: ~20 Gflops (latest R&D code gives > 30)
  - GPU (GTX-580 gaming card): Streams ~177 GB/s,
    LQCD: > 330 Gflops (split precision gives 2x boost)

Problem sizes today are 1-4 GPUs, moving to 8 (minimum)

GPU programming is difficult (2 person years for key kernel), but the end result is compelling for many science problems

ARRA GPU Cluster

125 nodes, 500 GPUs

2.4 / 2.53 GHz Nehalem / Westmere

48 GB memory per node

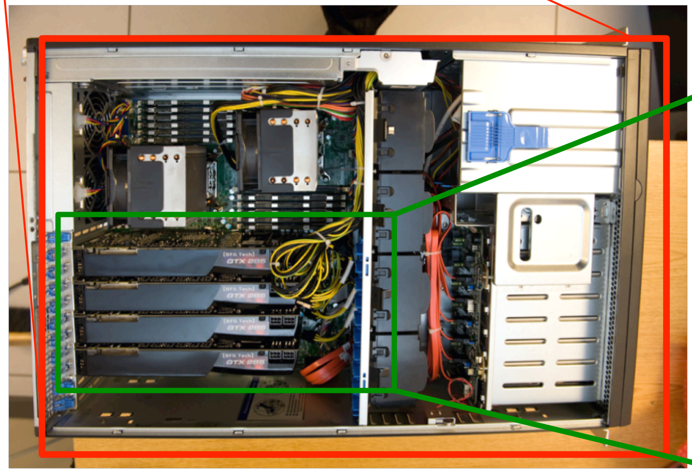
85 with SDR Infiniband (file I/O)

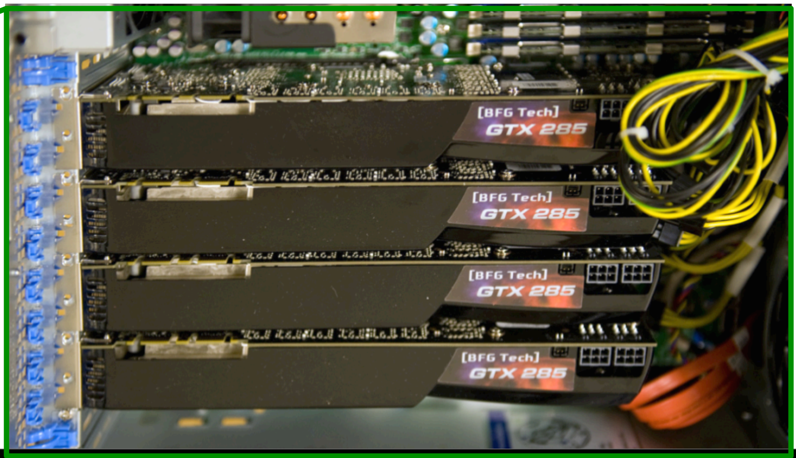
32 with QDR in 4x slot == ½ QDR

8 with dual QDR

Optimized for 4-8 GPUs / job

One quad-GPU node =
one rack of Infiniband nodes





GPU Comparison *(in use at JLab)*

Card	GPU	#cores	clock speed (GHz)	memory size (GB)	raw memory bandwidth (GB/s)	clover inverter (Gflops)[1]	cost
GTX-285	GT200b	240	1.47	2	159	135	$500
C1060	GT200b	240	1.30	4	102	100	$1500
GTX-480	Fermi	480	1.40	1.25	177	270	$500
C2050[2]	Fermi	448	1.15	2.67	144	185	$2100

[1] Newest development code gets up to 310 Gflops on GTX-480; data is this talk uses older 270 Gflops; all numbers are for mixed precision

[2] C2050 evaluated with ECC enabled

The Fermi Tesla line of cards (C2050) has a significant advantage in having ECC memory so that more than just inverters can be safely executed. This comes at a steep price: 4x on GPU price, and 1.5x on lower performance. Integrated into a host this yields a price performance difference between the two of 3x.

Conclusion: judicious use of gaming cards is a very good idea as long as we have inverter heavy loads (which we do).

GPU Job Effective Performance

Comparing GPUs to regular clusters can't be done on the basis of inverter performance (Amdahl's Law problem), so instead we compare job clock times, and from that derive an "effective" equivalent performance, which is the cluster inverter performance multiplied by the job clock time reduction.

The following table shows the number of core-hours in a job needed to match one GPU-hour in a job. Last project used 32 single GPU nodes and was I/O bound.

The allocation-weighted performance of the cluster is **63 TFlops**.

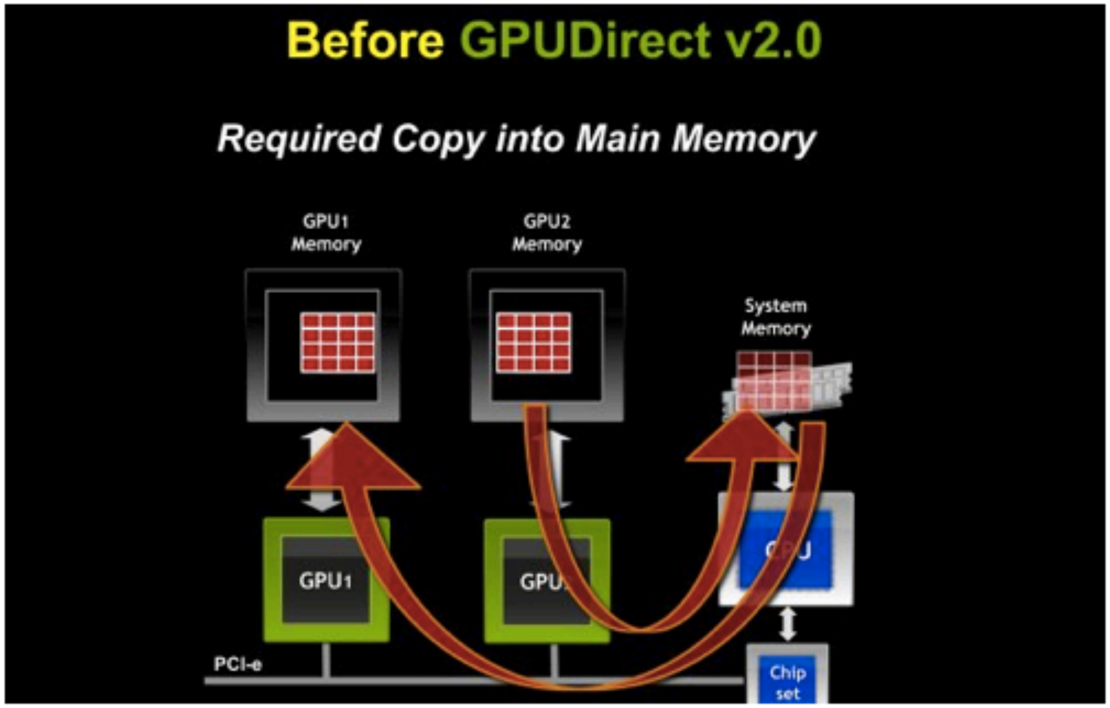
Project	2010-2011 GPU Hours	#GPUs, nodes	Xeon core hours / GPU hour (job time)	Effective Performance Gflops/node	GPU used
Spectrum	1,359,000	4, 1	90	800	(average)
thermo	503,000	4, 1	45	400	(average)
disco	459,000	4, 1	46	410	C2050
Tcolor	404,000	4, 1	20*	175*	GTX285
emc	311,000	4, 1	40	350	(average)
gwu	136,000	32, 32	24*	50*	GTX285

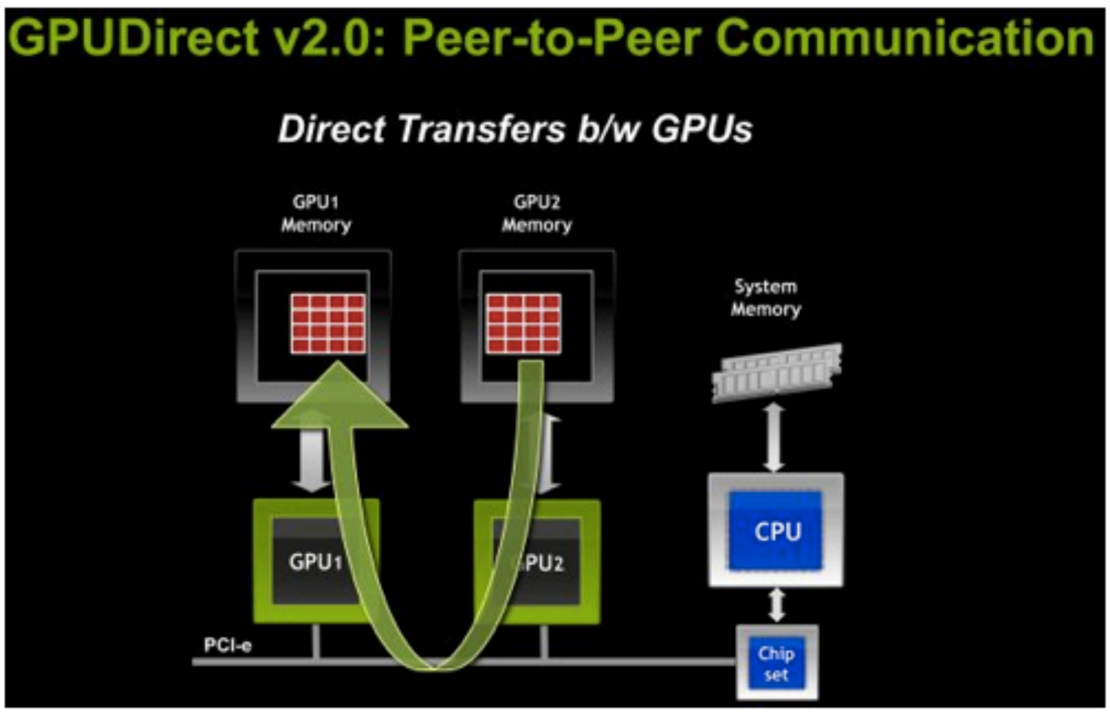
*using only 2009 generation GPUs, which are 2x slower

Weak & Strong Scaling

- Key points in inverter behavior
  - communicate surface (face) of sub-cube in 1 to 5 dimensions
  - concurrently compute internal update
  - update surface after face exchange

- I/O Limitations:
  - GPUs are I/O bound: only 11 GB/s (PCI) I/O for 200 Gflops compared to 6 GB/s and 20 Gflops on dual Xeon QDR node
  - Observed: 32^3 x 256 problem fits on 8 GPUs,
    only needs ½ GB/s between the two hosts (only ¼ of I/O on IB)
    (SDR in x4 slot, slice on time dimension only)
  - Strong scaling to 32 GPUs using full QDR, GPUDirect
  - If scaling is worse than expected, add second QDR rail
  - Weak scaling for 48^3 x 384 lattice (5x) to 128 GPUs
  - Switch to dual rail FDR when GPUs get 2x faster





GPUDirect

Transferring data via host memory cuts performance in half

Direct device to device communications will accelerate our theory modeling with little software change

PCIe switch chips (PLX) create a network within a host, with 10 GB/s bandwidth per GPU, with 4-8 GPUs/host

Dealing with Amdahl's Law

- Currently the largest production use of GPUs is the inverter ("small" kernel, carefully optimized, 3 person-years and climbing)
- Only part of the analysis work in so inverter heavy such that quad GPU nodes are optimal (Amdahl's Law problem)
- Going forward, need more code on the GPU
  - extreme by hand optimization: too manpower intensive
  - JIT based, use of patterns to translate: feasible, lower acceleration, but probably good enough
  - wait for hybrid CPUs, and let vendors deliver compiler that can handle all the code

Summary

- General Purpose GPUs are excellent compute engines (high flop rate, low watts/flops)
- Difficult to program (disruptive), but worth the effort in many cases
- Extremely cost effective for aggregate HPC (1-10 Tflops/job or 1K-10K cores: 10x compared to next best solution)
- Software and software tools have to evolve to better exploit heterogeneous architectures
- We are watching & exploring multiple architectures & approaches