Scientific Computing at Jefferson Lab
GPUs and Lustre

Sandy Philpott
Scientific Computing
Jefferson Lab

HEPiX Cornell 11/4/10

Scientific Computing at JLab

JLab has significant computing capacity

- 1400 compute servers (batch)
- a leading edge GPU cluster

  single most powerful dedicated Lattice QCD (LQCD) resource
- a growing repository of 6 GeV data, more than 4 PB today
- growing disk capacity and bandwidth,

  ~ 600 TB disk, half on a 20 Gb/s Infiniband fabric
- growing expertise with leading edge open source tools

  Lustre file system, Auger / PBS / Maui batch system, Jasmine storage

Software developments

- multi-threading and multi-GPU libraries for high performance computing
- LQCD algorithm development, performance optimizations
- highly scalable analysis database for LQCD
- system management tools

  batch, disk management

  power & system monitoring

Conventional Batch Systems

Single process + multi-core (single node) jobs

Largest tasks:

Experimental Physics Detector Simulation

Data Analysis

Hardware

- farm of ~100 nodes, of which ~60 are recent generation
  (older 32 bit nodes will be retired by the end of 2010)
- cache disk (70 TB) and work disk (110 TB)
- 8 LTO-4 + 4 LTO-5 tape drives, aggregate bandwidth of 1 GB/s

Annual upgrades

- Requirements driven
- Jlab 6 GeV budget supports modest growth, tracking Moore's Law both for compute performance and disk capacity

High Performance Computing

Multi-core jobs, up to 1024 cores and/or GPU accelerated

Funded primarily for Lattice QCD Theory Calculations

Some allocation for Accelerator Modeling & FEL

Hardware

- x86 clusters (2007, 2009, 2010), mostly dual quad core
- high performance networks, up to 40 Gb/s Infiniband
- significant GPU resource: 532 GPUs

  one GPU outperforms 200 cores
- 400 Tbytes disk space, aggregate bandwidth > 4 GB/s

Project driven upgrades, typically every 2 years

Disruptive Technology: GPUs

A Graphics Processing Unit is a device that does simple arithmetic (+-*/) very fast.

High performance is driven by the video gaming market (both PC/workstation, and game consoles): matrix transformations (movement, rotation, perspective), lighting effects, and other visual depth queues.

In order to yield high frame rates at high resolution with a large number of objects in a scene, each with many facets to render (to appear realistically smooth), a very high floating point rate is required. But the software is relatively simple: lots of linear algebra.

GPUs are now so fast, that they vastly outperform CPUs in doing algebra, and have become a target for computational science.

Modern GPUs

Characteristics of GPUs:

- Lots of simple cores with fast context switching to hide latency
- SIMD architecture (single instruction, multiple data)
- High memory bandwidth
- Complex memory hierarchy, to achieve high bandwidth at low cost
- Recent evolution: IEEE floating point to intentionally support science: GPGPU = general purpose GPU

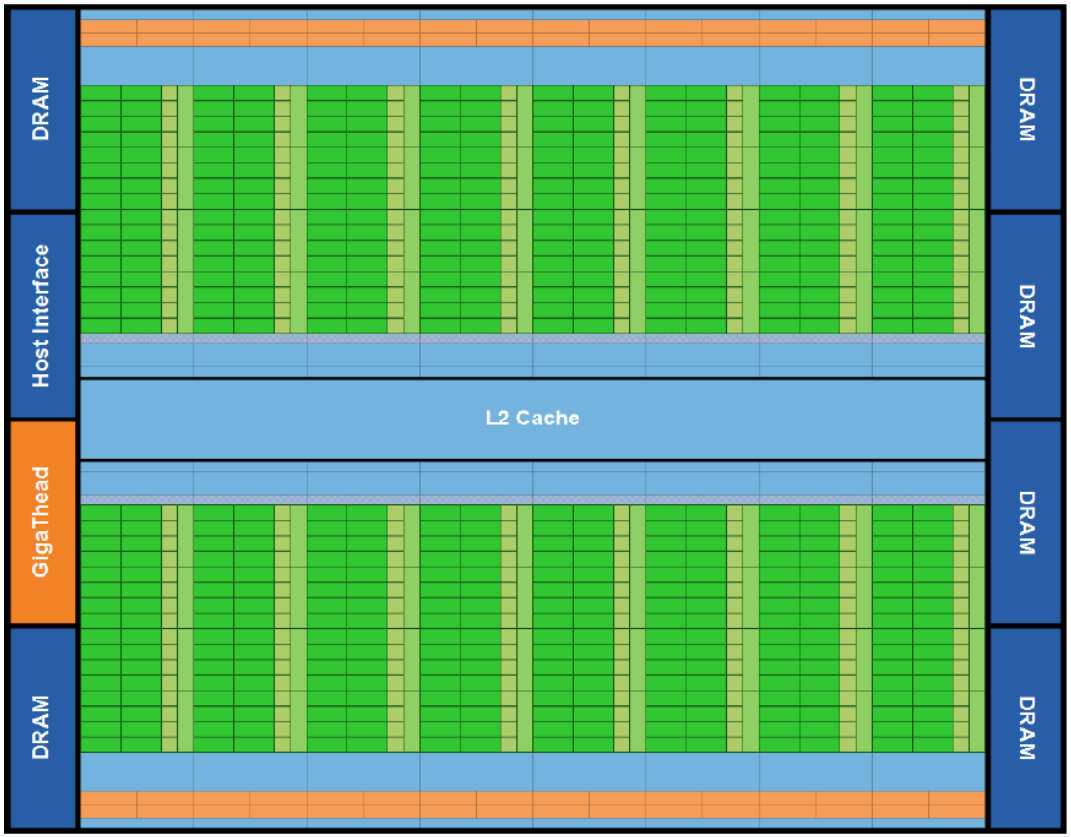
Commodity Processors	CPU	NVIDIA GPU
#cores	12	480
Clock speed	3 GHz	1.4 GHz
Main memory bandwidth	20 GB/s	120 GB/s
I/O bandwidth	7 GB/s (dual QDR IB)	5 GB/s
Power	80 watts	225 watts

Fermi GPU Architecture

Each GPU has…

- 16 Streaming *Multiprocessors* each containing…
  - 32 hyper-threaded scalar processors, 16 double precision units
    - 512 cores per GPU, each with 4-way hyperthreading
  - 4 Special Function Units
  - large register file
  - L1 cache
- 6 Memory controllers (GDDR5) w/ ECC

At 3 billion gates, this is the most complex chip in existence. Yield is not great, so some chips have parts disabled (fewer cores).



Fermi GPU

Major Changes from last year's version:

- up to 512 cores vs 240
- cache memory: L1 (single SM) and L2 (all cores)
- ECC memory, 2.6 GB or 5.2 GB (or 3 - 6 with ECC disabled)
- 8x double precision peak rate of prior GPU; (1 aggregate now >600 Gflops
- wider memory bus, GDDR-5 instead of GDDR-3 (allows future versions to increase bandwidth)

Software model remains similar…

lots of cores, context switching every clock cycle

GPU Software Model

- Lots of virtual cores (far more than actual), each running one simple thread (e.g. one iteration of one loop)
- Threads organized into blocks
- 2D structure for threads within a block, and blocks within a task
- All threads are independent (i.e., it doesn't matter in what order the threads execute; perfect data independence)

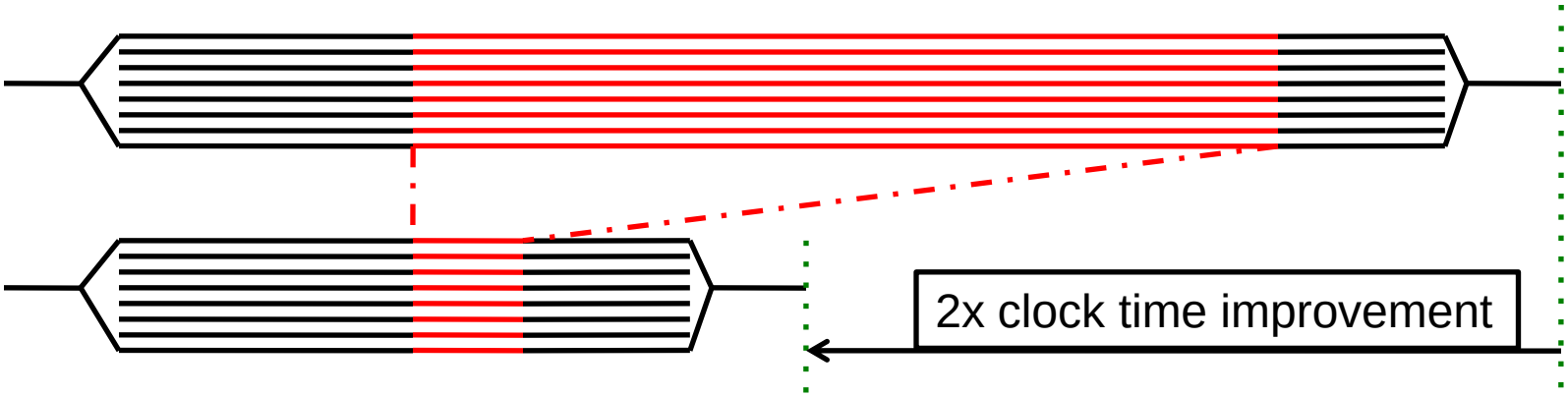
Parallel Software Models

- on CPU: thread libraries to use all the cores
  - OpenMP is the easiest
  - also: POSIX threads (Pthreads), MPI
- on GPU: parallel languages
  - CUDA: mature, NVIDIA specific, widely used
    (Compute Unified Device Architecture, refers to hardware & language + library)
  - OpenCL: new standard
    - Conceptually similar to CUDA but with different vocabulary
    - Can also run on CPU cores (common abstraction)
    - Task parallel as well as data parallel models
  - Much less mature compilers exist for GPUs than CPUs
    - Need to do manual optimizations, like loop unrolling,
      data movement to faster scratch pad (latest GPU also has a cache)
    - Simple code: 30 Gflops, hand optimized: 260 Gflops !

Amdahl's Law (Problem)

A major challenge in exploiting GPUs is Amdahl's Law:

If 60% of the application running time is GPU accelerated by 6x, the net gain is only 2x:



Also disappointing:
after acceleration, the GPU is idle 80% of the time!

Conclusion: one needs to move more code to the GPU, and/or create task level parallelism (overlap CPU and GPU)

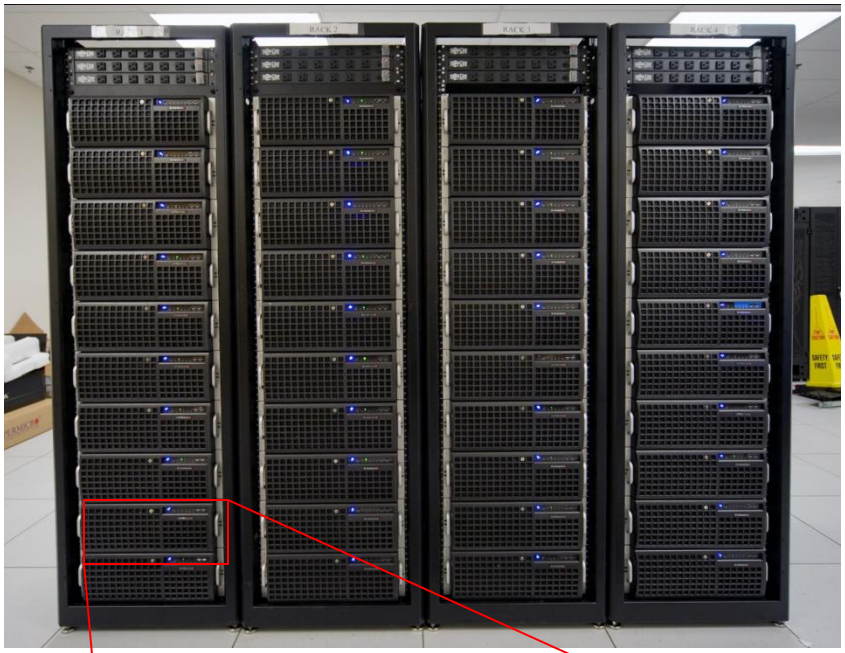
Quick Look: Lattice QCD

- Lattice QCD is very memory bandwidth intensive
  - 1 instruction per byte of memory touched; repetitive sweeps through memory
  - Size of lattice problem ~1000 x CPU cache memory size, so on conventional computer limited by memory bandwidth & latency
  - Flops (sustained) is about ½ memory bandwidth
  - Dual Nehalem:

    Streams 37 GB/s LQCD: ~20 Gflops
  - GPU (GTX-480 gaming card):

    Streams 120 GB/s LQCD: 250 Gflops (mixed precision)
- GPU programming is difficult (1 person-year to achieve 250 Gflops), but the end result is compelling for many science problems

Hardware: ARRA GPU Cluster



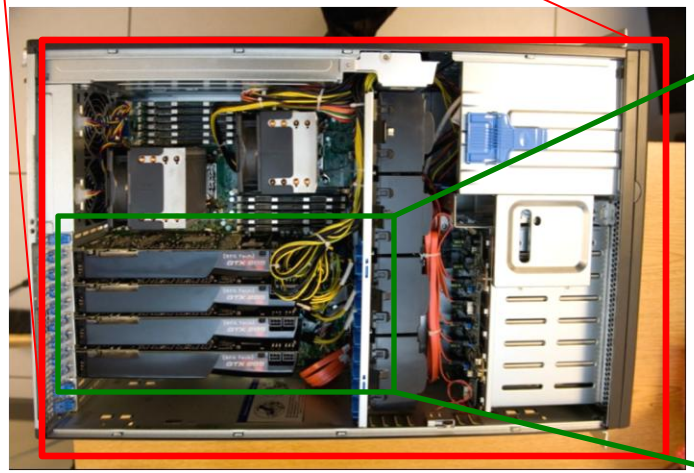
Host:

dual 2.4 or 2.53 GHz Xeon
48 GB memory / node
117 nodes, 4 GPUs / node

50 nodes w/ 4 GTX-480 GPUs
32 nodes w/ 4 C2050 + QDR IB
35 nodes w/ 4 GT200b GPUs

One quad GPU node outperforms one rack of conventional nodes!

Cost is 10x lower, as is electricity use!



10x Reduction in $/MFlops

- Usage model
  - Single job / GPU: multiple jobs / host
  - Parallel GPU: one host, or multi-host
  - Scaling to teraflops, but constrained by Amdahl's law

    (4 GPUs accelerates dual Xeon host by 50x, so if 80% of the run-time is accelerated, final speed up is <5x)

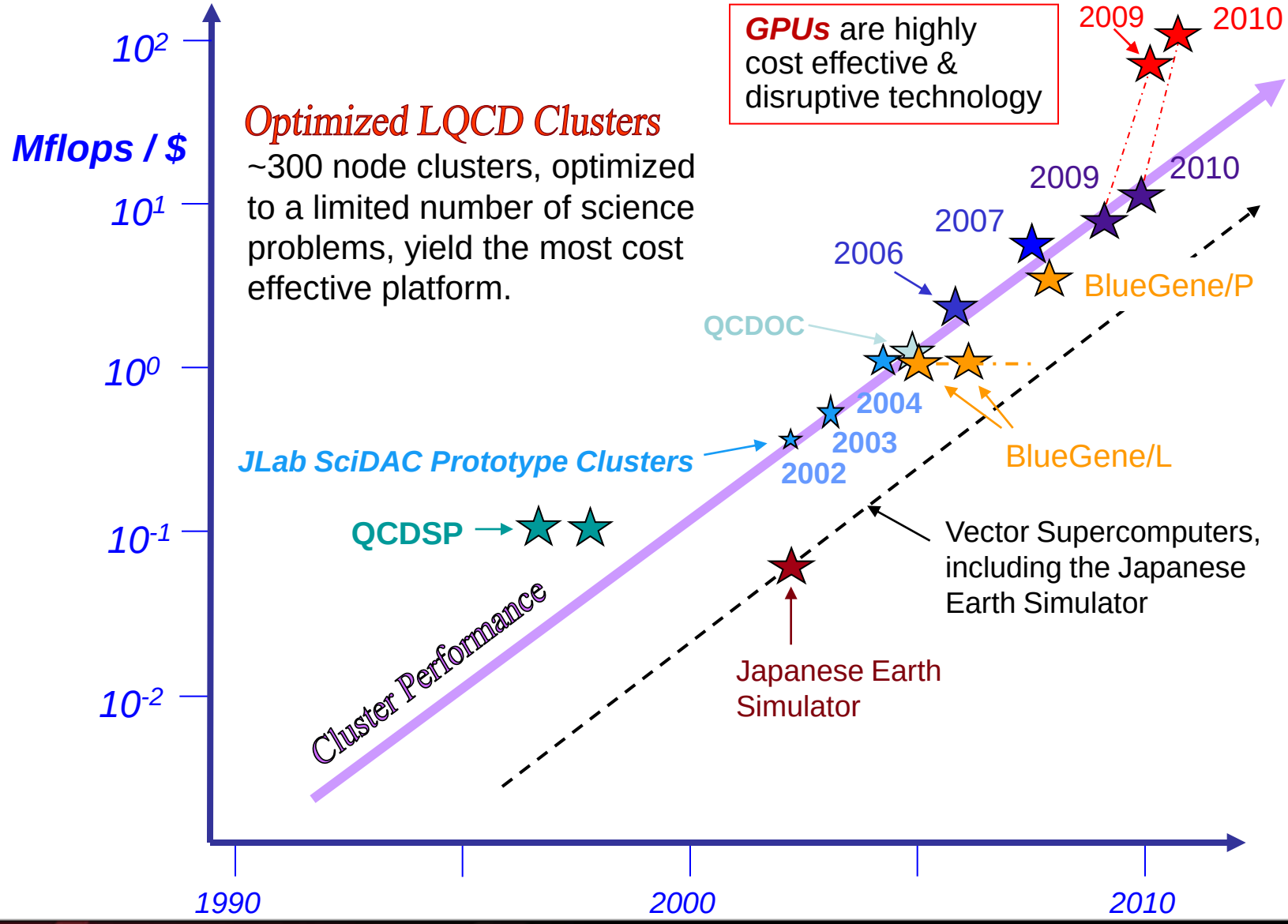
- High Capacity Computing Example at JLab
  - Dual Xeon 2.4 GHz host with Infiniband, ~$3,700

    $0.20 / LQCD Mflops
  - 4 gaming cards, each $500
  - Quad GPU performance: up to 1000 Gflops, $5,700

    on kernel, we achieve $0.006 / LQCD Mflops

    w/ Amdahl's Law we achieve ~ $0.01 / MFlops

Science per Dollar for LQCD Applications



A Large Capacity Resource

532 GPUs at Jefferson Lab (July)

- ★ 200,000 cores (1,600 million core hours per year)
- ★ 600 Tflops peak single precision
- ★ 100 Tflops aggregate sustained in the LQCD matrix inverter, (mixed half / single precision)
- ★ Significant increase in dedicated USQCD resources

All this for only $1M with hosts, networking, etc.

Disclaimer: to exploit this performance, code has to be run on the GPUs, not the CPU (Amdahl's Law problem). This is a software development problem.

Lustre

300TB on commodity hardware

- Metadata Server
  - Penguin dual Nehalem from Penguin Computing
    - Reused head node from our Skyld R&D cluster
  - 6 * 1TB Western Digital disks
  - RAID10
- Object Storage Servers
  - 14 SuperMicro dual quad core systems from AMAX
  - 3ware 9650 24-port RAID controller
  - 24 * 1TB Hitachi disks

Installation Fall 2009, Lustre 1.8.1

OSS configurations

Original configuration as purchased from AMAX

- All 24 disks in one RAID6 set
  - operating system
  - filesystem journal
  - data

- Under load, controllers would hang, operating system would crash in kjournald
  - Clients would wait…
  - Filesystem corruption?

- Became clear that reconfiguration is required

OSS Reconfigurations

- Separate the single RAID6 set into
  - Operating system: RAID1
  - Filesystem journal
  - Data: 2 8+2 RAID6 sets
- Install the latest Lustre version, 1.8.4
- Upgrade the RAID controller firmware
- Upgrade the motherboard BIOS
  - Which failed, and left a system down until a replacement motherboard arrived

- Do we actually need more than one controller?
  - Or maybe just a different controller?

Filesystem Check

- First, run a Lustre filesystem check
  - MDS
  - OSSs
  - Global

- MDS: had to stop the fsck after 24+ hours

```
fsck.ext3 -fy --mdsdb mdsdb /dev/md1
```

- 200,000+ small files lost from one user
  - what was the circumstance that caused this?
  - Lustre only used for volatile files, so no loss of critical data

Remove/Add OSSs

To reconfigure each OSS, they have to be drained first

- Our first experience with the Lustre tools in detail
  - Parallel copies, but not the first iteration
- Lustre showed OSTs that no longer existed
- df would hang…

All OSSs upgraded to 1.8.4 first

- but the MDS started having problems - TCP over IB stopped working
- Ah! The MDS didn't get the 1.8.4 upgrade…
  - Done now; no further problems to date

It takes a long time to reconfigure 14 nodes…in progress.

Lustre Plans

- Finish OSS reconfigurations
- Install new failover MDS hardware – already onsite
  - 2 Dell R710s
  - Dell MD3000 RAID
- Purchase more OSS hardware
  - From AMAX?
    - 36 * 2TB disks
    - LSI MegaRAID controller(s)?
- Upgrade to Lustre 2
  - integration to our JASMine mass storage system
- Follow the Lustre consortium plans closely…