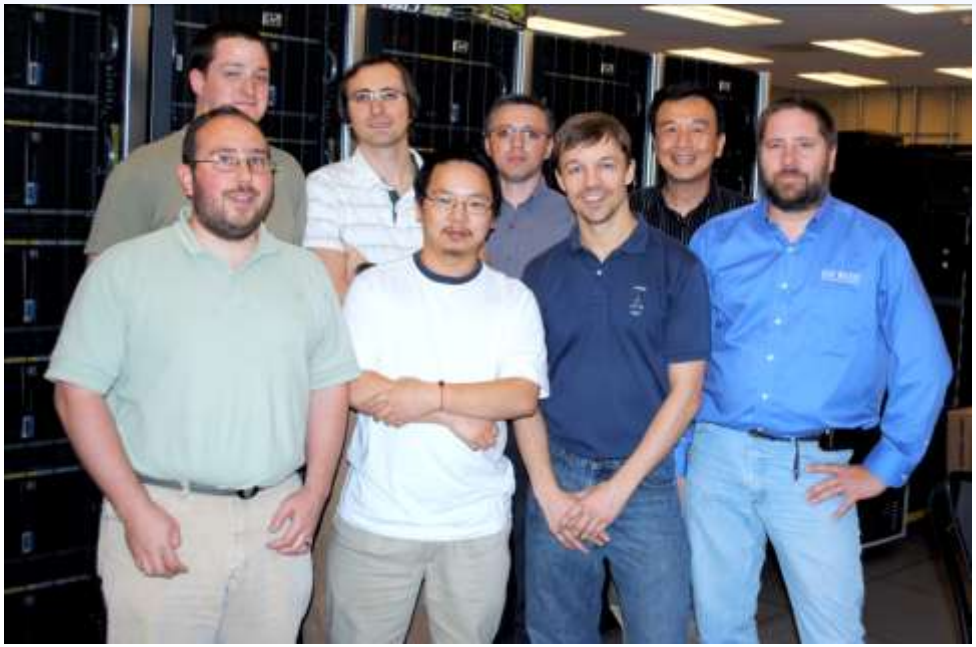
GPU Clusters for HPC

Volodymyr Kindratenko
Innovative Systems Laboratory

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

Acknowledgements

- ISL research staff
  - Jeremy Enos
  - Guochun Shi
  - Michael Showerman
  - Craig Steffen
  - Alexey Titov
- UIUC collaborators
  - Wen-Mei Hwu (ECE)
- Funding sources
  - NSF grants 0810563 and 0626354
  - NASA grant NNG06GH15G
  - NARA/NSF grant
  - IACAT's Center for Extreme-scale Computing



ISL Research

- Evaluation of emerging computing architectures
  - Reconfigurable computing
  - Many-core (GPU) architecture
  - Heterogeneous clusters
- Systems software research and development
  - Run-time systems
  - GPU accelerator cluster management
    - Tools and utilities: GPU memory test, power profiling, etc.
- Application development for emerging computing architectures
  - Computational chemistry (electronic structure, MD)
  - Computational physics (QCD)
  - Cosmology
  - Data mining

Top 10 from

TOP-500

Rank	Site	Computer
1	National Supercomputing Center in Tianjin	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C
2	DOE/SC/Oak Ridge National Laboratory	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz
3	National Supercomputing Centre in Shenzhen (NSCS)	Nebulae - Dawning TC3600 Blade, Intel X5650, NVIDIA Tesla C2050 GPU
4	GSIC Center, Tokyo Institute of Technology	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, NVIDIA GPU, Linux/Windows
5	DOE/SC/LBNL/NERSC	Hopper - Cray XE6 12-core 2.1 GHz
6	Commissariat a l'Energie Atomique (CEA)	Tera-100 - Bull bullx super-node S6010/S6030
7	DOE/NNSA/LANL	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband
8	National Institute for Computational Sciences/University of Tennessee	Kraken XT5 - Cray XT5-HE Opteron 6-core 2.6 GHz
9	Forschungszentrum Juelich (FZJ)	JUGENE - Blue Gene/P Solution
10	DOE/NNSA/LANL/SNL	Cielo - Cray XE6 8-core 2.4 GHz

Green500

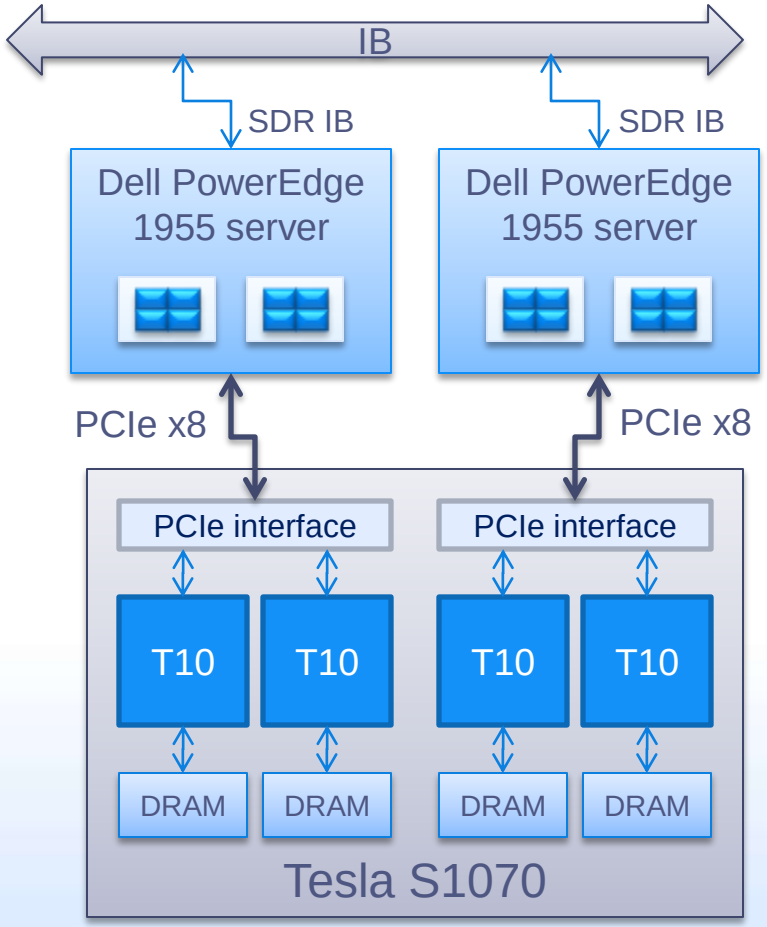
Rank	Site	Computer
1	IBM Thomas J. Watson Research Center	NNSA/SC Blue Gene/Q Prototype
2	GSIC Center, Tokyo Institute of Technology	HP ProLiant SL390s G7 Xeon 6C X5670, NVIDIA GPU, Linux/Windows
3	NCSA	Hybrid Cluster Core i3 2.93Ghz Dual Core, NVIDIA C2050, Infiniband
4	RIKEN Advanced Institute for Computational Science	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
5	Forschungszentrum Juelich (FZJ)	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus
5	Universitaet Regensburg	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus
5	Universitaet Wuppertal	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus
8	Universitaet Frankfurt	Supermicro Cluster, QC Opteron 2.1 GHz, ATI Radeon GPU, Infiniband
9	Georgia Institute of Technology	HP ProLiant SL390s G7 Xeon 6C X5660 2.8Ghz, NVIDIA Fermi, Infiniband QDR
10	National Institute for Environmental Studies	GOSAT Research Computation Facility, NVIDIA

QP: first GPU cluster at NCSA

- 16 HP xw9400 workstations
  - 2216 AMD Opteron 2.4 GHz dual socket dual core
  - 8 GB DDR2
  - PCI-E 1.0
  - Infiniband QDR
- 32 Quadro Plex Computing Servers
  - 2 Quadro FX 5600 GPUs
  - 2x1.5 GB GDDR3
  - 2 per host

Lincoln: First GPU-based TeraGrid production system

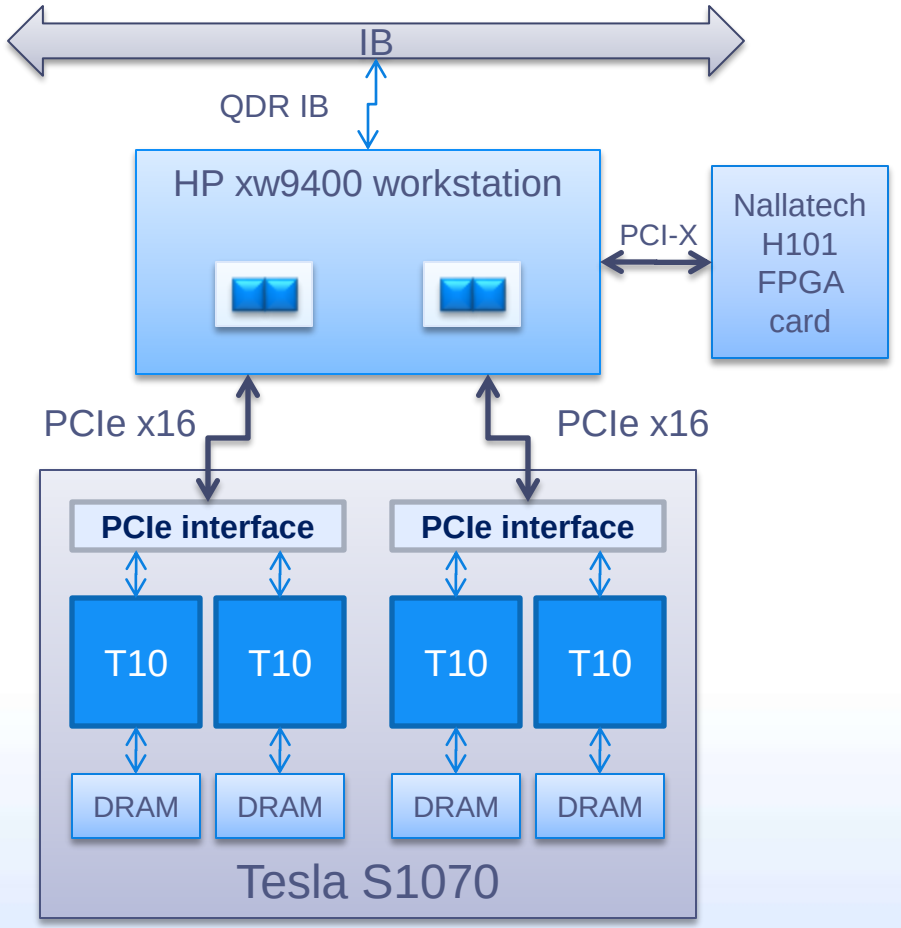
- Dell PowerEdge 1955 server
  - Intel 64 (Harpertown) 2.33 GHz dual socket quad-core
  - 16 GB DDR2
  - Infiniband SDR
- Tesla S1070 1U GPU Computing Server
  - 1.3 GHz Tesla T10 processors
  - 4x4 GB GDDR3 SDRAM
- Cluster
  - Servers: 192
  - Accelerator Units: 96



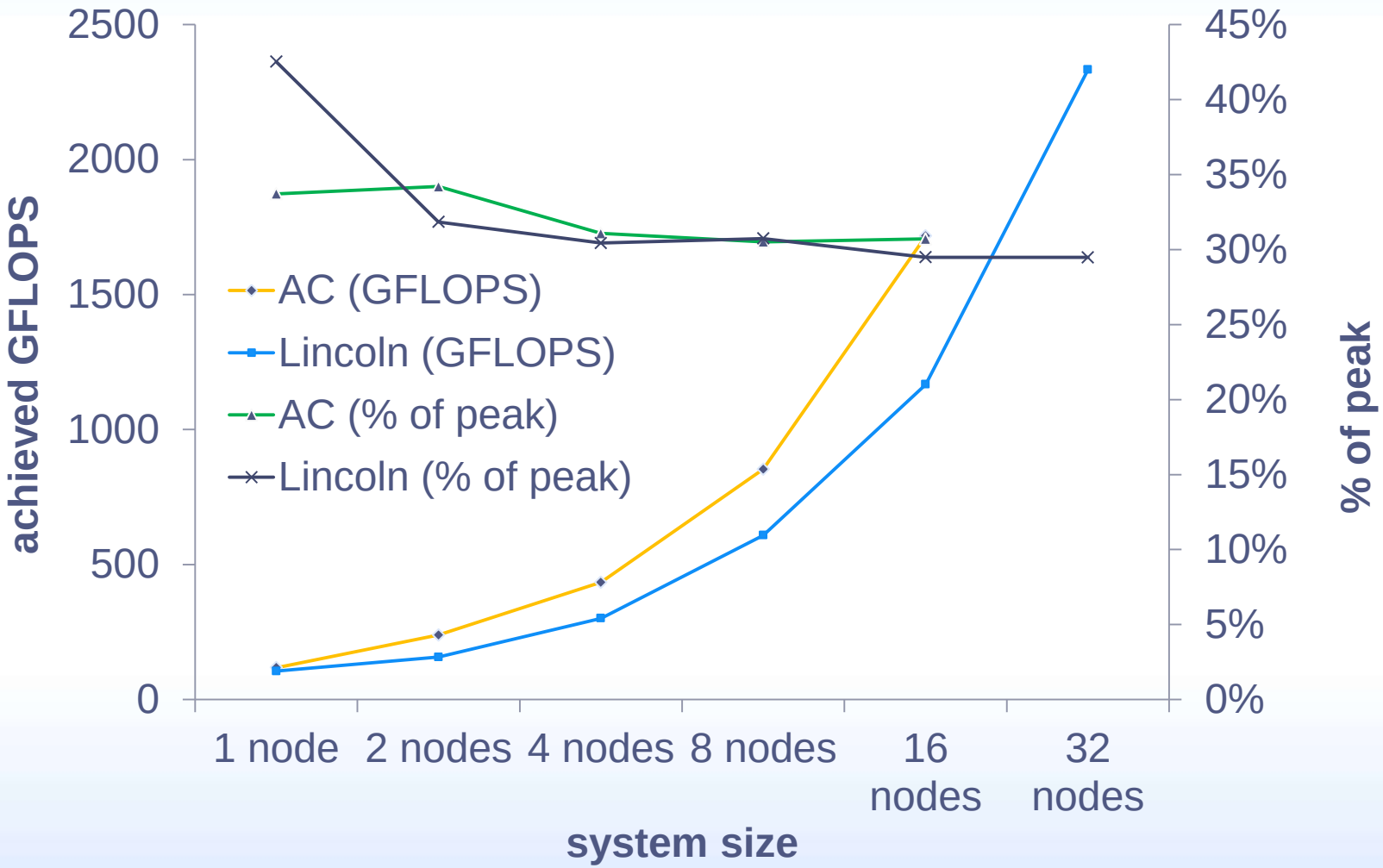
QP follow-up: AC

AC01-32 nodes

- HP xw9400 workstation
  - 2216 AMD Opteron 2.4 GHz dual socket dual core
  - 8GB DDR2 in ac04-ac32
  - 16GB DDR2 in ac01-03, “bigmem” on qsub line
  - PCI-E 1.0
  - Infiniband QDR
- Tesla S1070 1U GPU Computing Server
  - 1.3 GHz Tesla T10 processors
  - 4x4 GB GDDR3
  - 1 per host

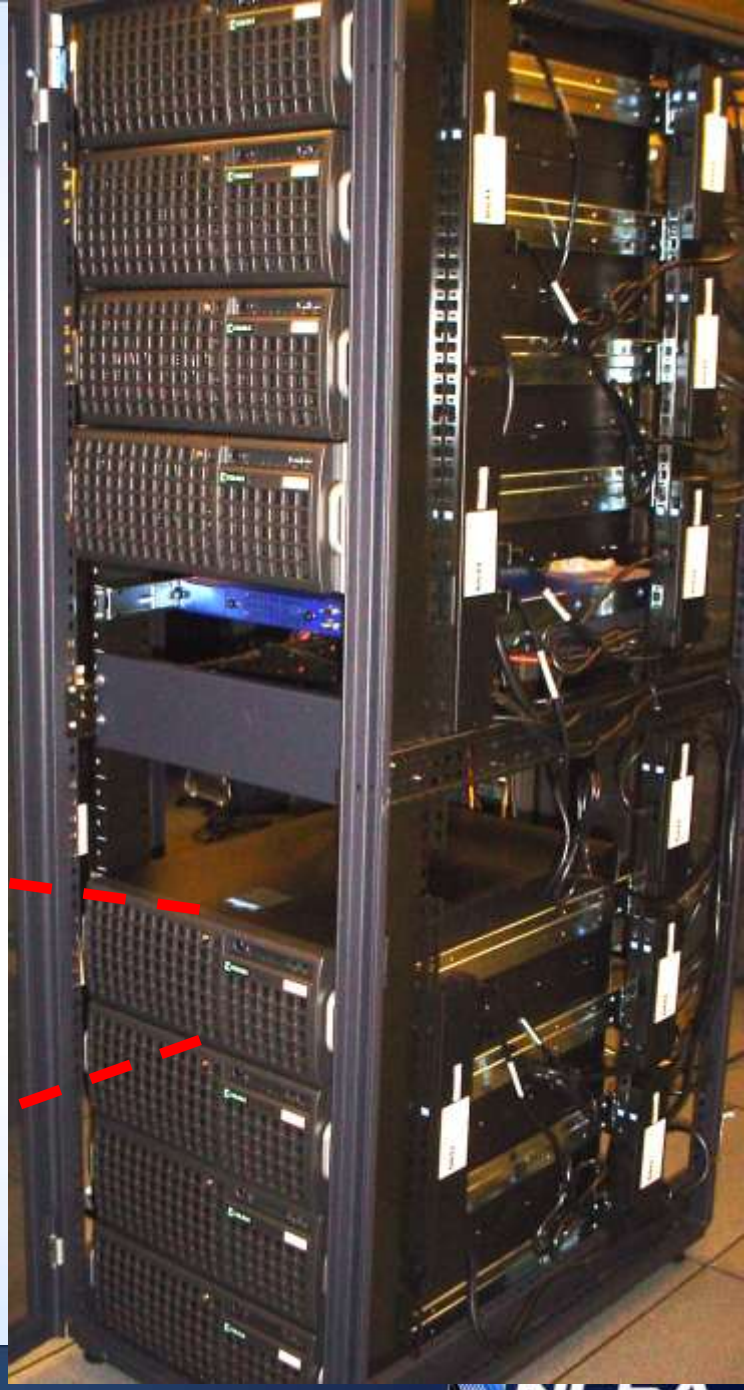


Lincoln vs. AC: HPL Benchmark



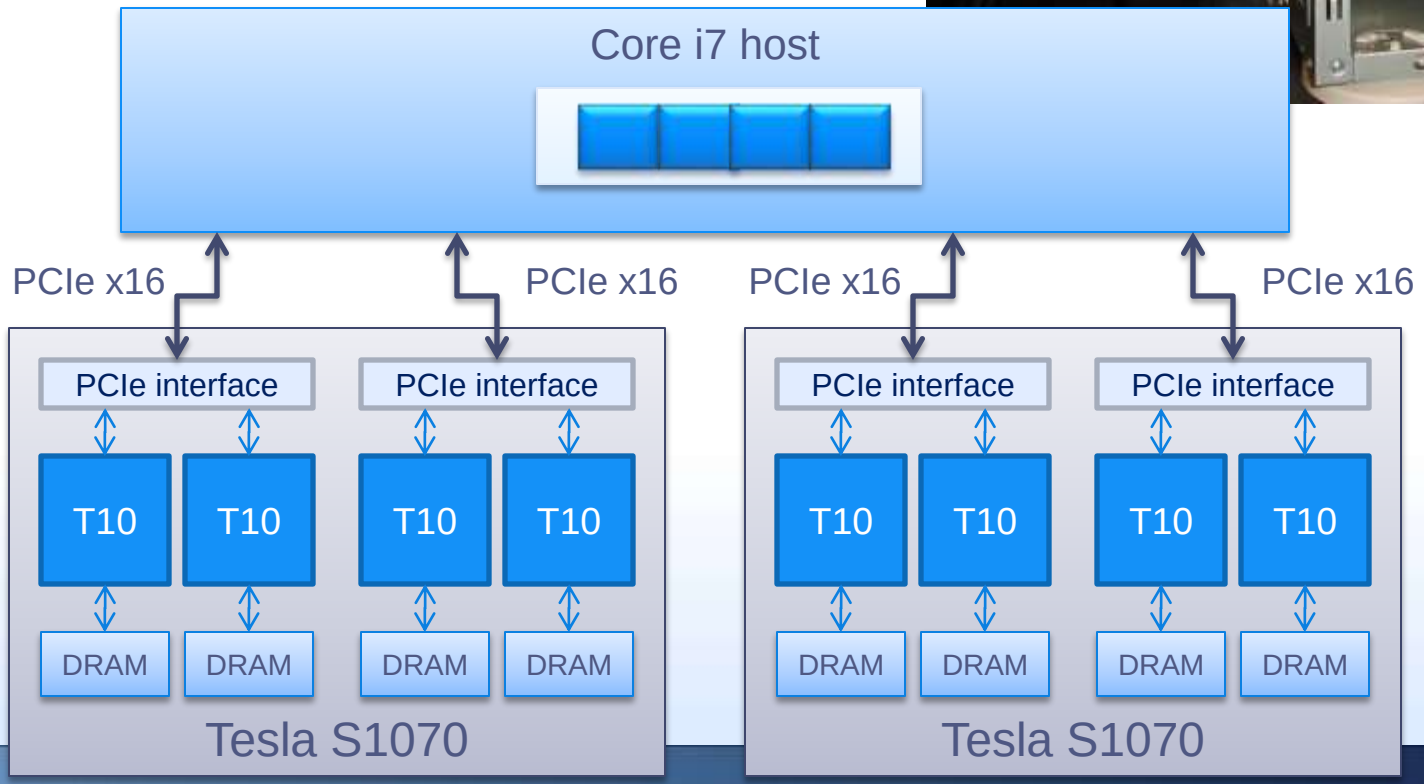
AC34-AC41 nodes

- Supermicro A+ Server
  - Dual AMD 6 core Istanbul
  - 32 GB DDR2
  - PCI-E 2.0
  - QDR IB (32 Gbit/sec)
  - 3 Internal ATI Radeon 5870 GPUs



AC33 node

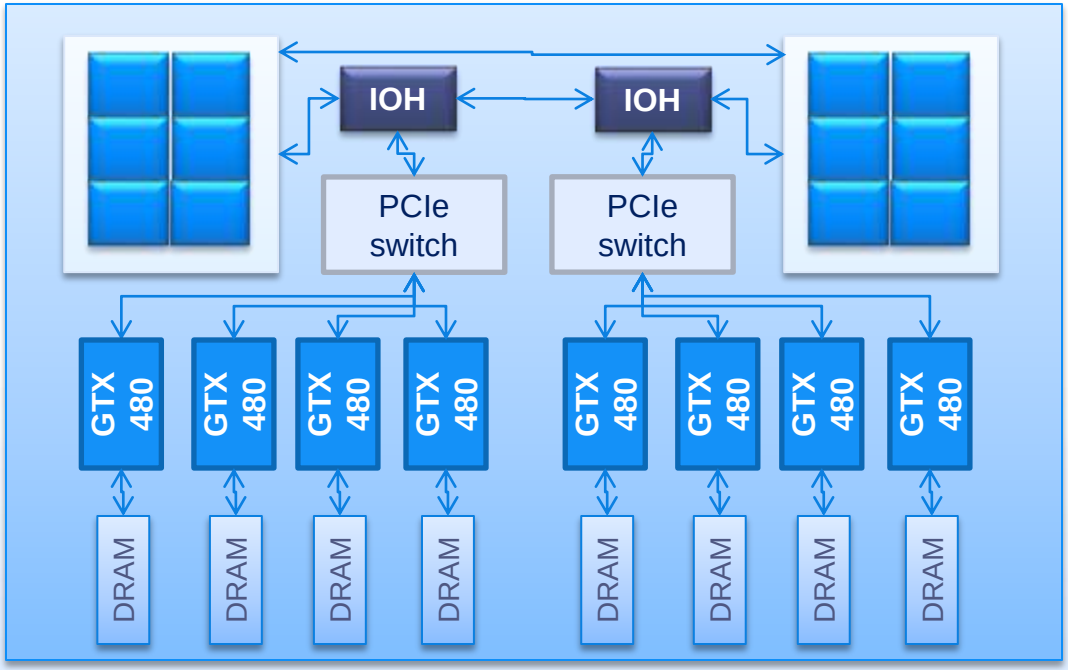
- CPU cores (Intel core i7): 8
- Accelerator Units (S1070): 2
- Total GPUs: 8
- Host Memory: 24-32 GB DDR3
- GPU Memory: 32 GB
- CPU cores/GPU ratio: 1:2
- PCI-E 2.0
- Dual IOH (72 lanes PCI-E)



NCSA

AC42

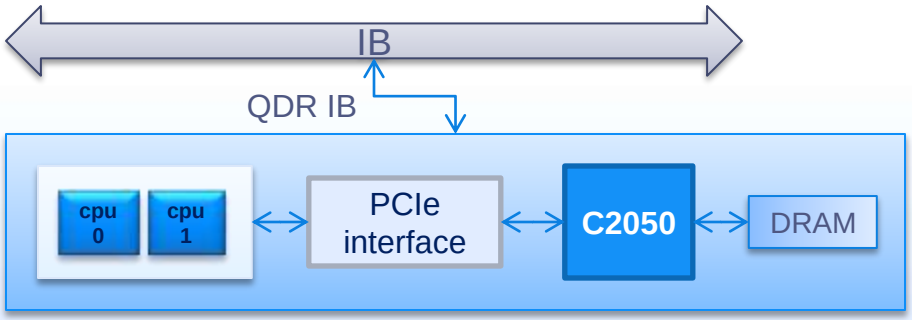
- TYAN FT72-B7015
  - X5680 Intel Xeon 3.33 GHz (Westmere-EP) dual-sosket hexa-core
  - Tylersburg-36D IOH
  - 24 GB DDR3
  - 8 PCI-E 2.0 ports
    - switched
- NVIDIA GTX 480
  - 480 cores
  - 1.5 GB GDDR5



EcoG: #3 on Green500 list

EcoG nodes designed for low power

- EVGA P55V 120-LF-E651-TR Micro ATX Intel Motherboard
  - Core i3 530 2.93 GHz single-socket dual-core
  - 4 GB DDR3
  - PCIe x16 Gen2
  - QDR Infiniband
- Tesla C2050
  - 448 cores
  - 3 GB GDDR5
- Cluster
  - 128 nodes



GPU Cluster Software

- Shared system software
  - Torque / Moab
  - ssh
- Programming tools
  - CUDA C SDK
  - OpenCL SDK
  - PGI+GPU compiler
  - Matlab
  - Intel compiler
- Other tools
  - mvapich2 mpi (IB)

- Unique to AC
  - CUDA wrapper
  - memtest
  - Power profiling

Need for GPU-aware cluster software stack

- Issues unique to compute nodes with GPUs
  - Thread affinity mapping to maximize host-GPU bandwidth
  - CUDA API/driver software bugs (mainly in initial product releases)
  - ...
- Issues unique to the GPU cluster
  - Efficient GPUs sharing in a multi-user environment
  - GPU memory cleanup between different users
  - …
- Other issues of interest
  - Are GPUs reliable? (Non-ECC memory in initial products)
  - Are they power-efficient?
  - …

Effects of NUMA

- On some systems

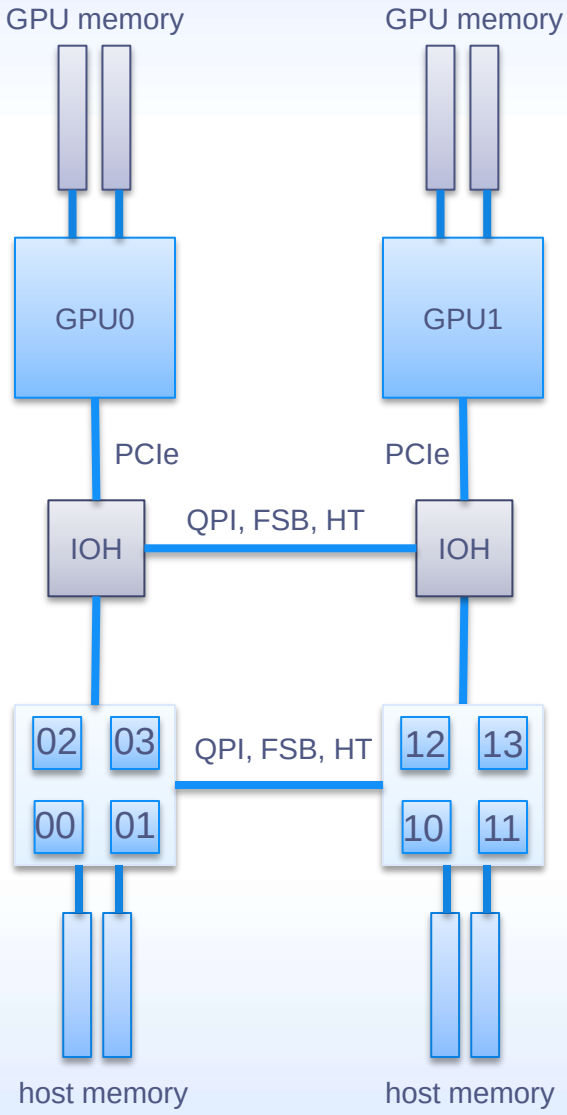
  access time from CPU00 to GPU0

  ≠

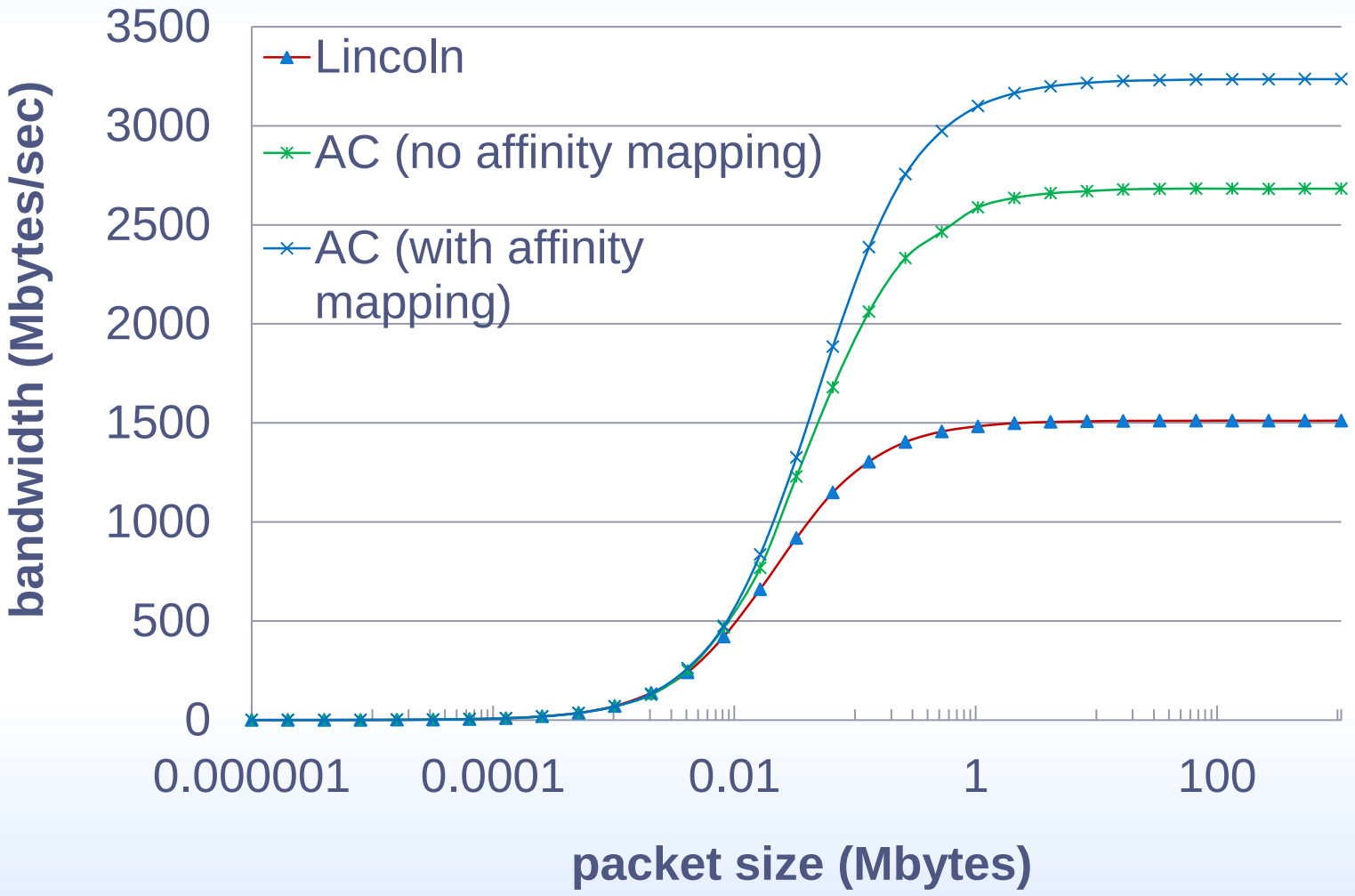
  access time from CPU10 to GPU0

  - Latency and achievable bandwidth are affected

- Solution: automatic affinity mapping for user processes depending on the GPUs used

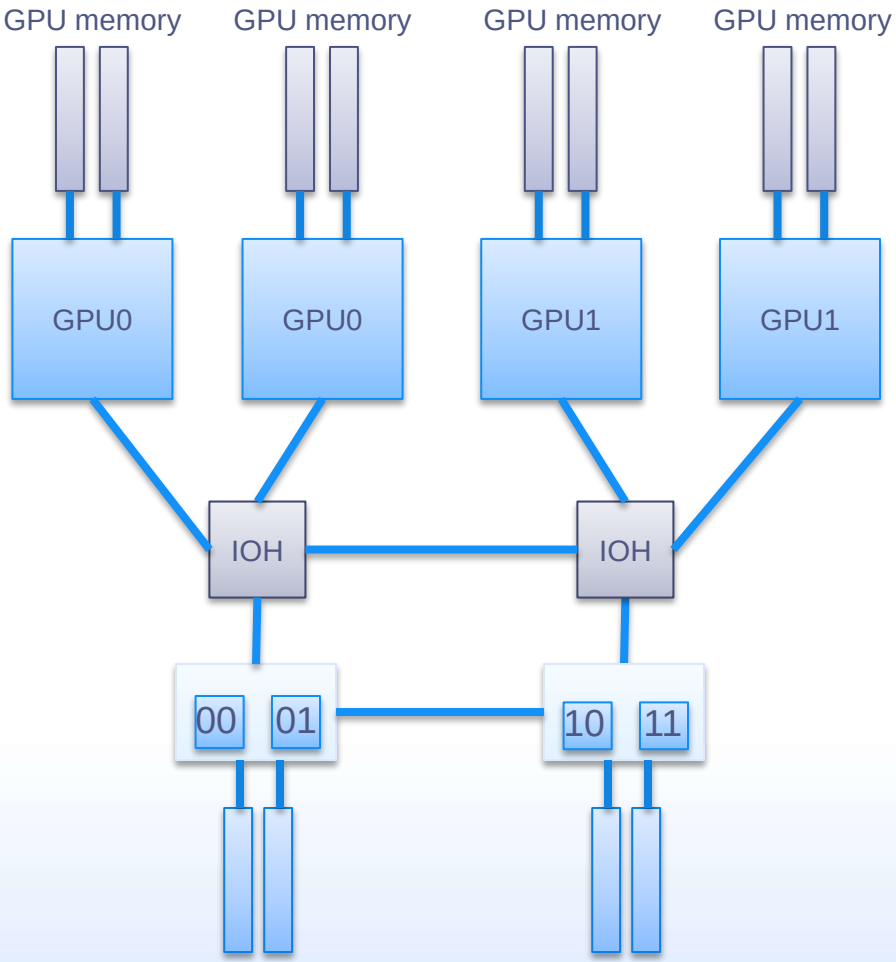


Host to device Bandwidth Comparison



Efficient GPU resources sharing

- In a typical cluster environment, user obtains exclusive access to the entire node
- A typical GPU cluster node has few (2-4) GPUs
- But a typical GPU cluster application is designed to utilize only one GPU per node
  - Giving access to the entire cluster node is wasteful if the user only needs one GPU
- Solution: allow user to specify how many GPUs his application needs and fence the remaining GPUs for other users



CUDA/OpenCL Wrapper

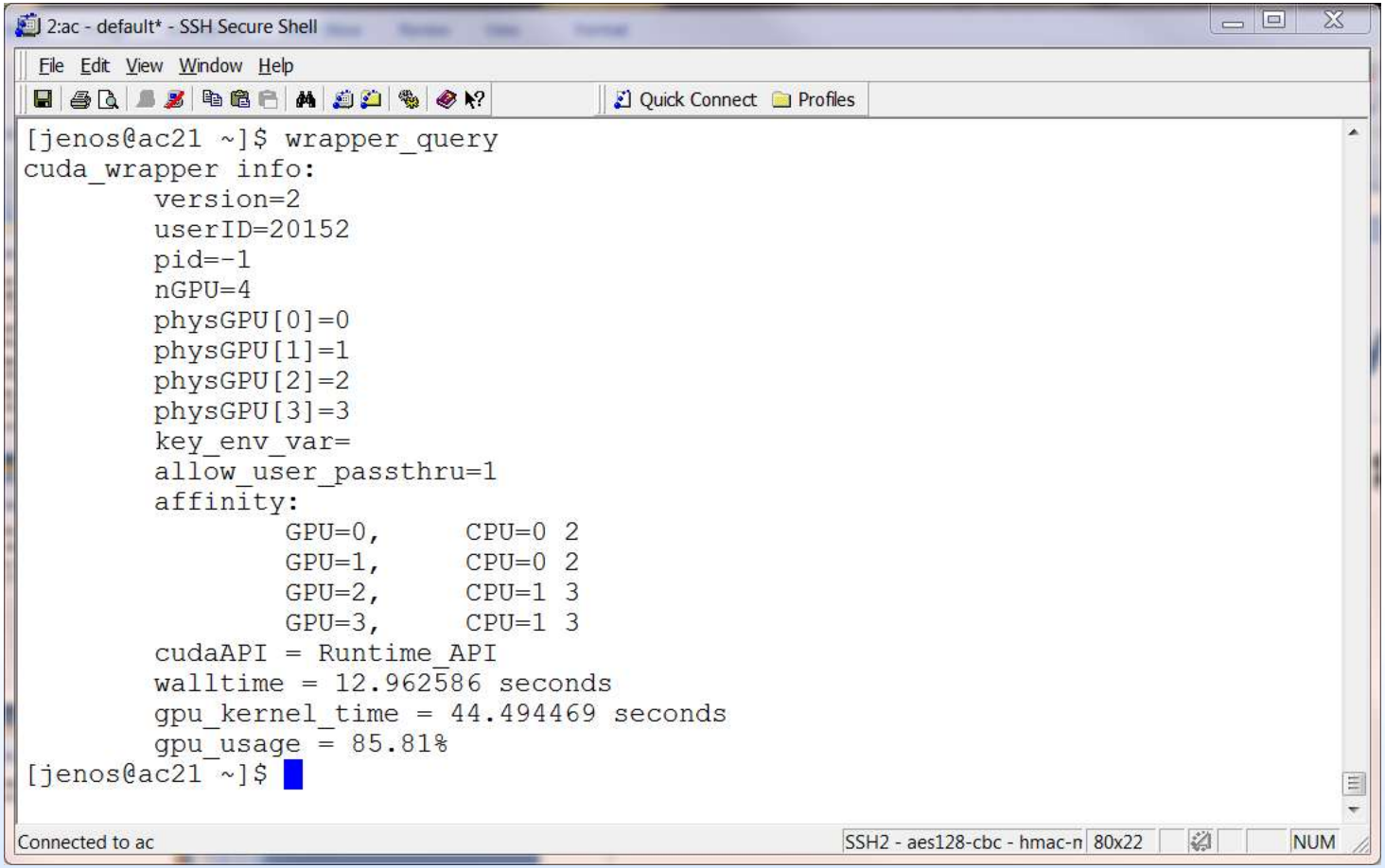
- Basic operation principle:
  - Use /etc/ld.so.preload to overload (intercept) a subset of CUDA/OpenCL functions, e.g. {cu|cuda}{Get|Set}Device, clGetDeviceIDs, etc.
  - Transparent operation
- Purpose:
  - Enables controlled GPU device visibility and access, extending resource allocation to the workload manager
  - Prove or disprove feature usefulness, with the hope of eventual uptake or reimplementation of proven features by the vendor
  - Provides a platform for rapid implementation and testing of HPC relevant features not available in NVIDIA APIs
- Features:
  - NUMA Affinity mapping
    - Sets thread affinity to CPU core(s) nearest the gpu device
  - Shared host, multi-gpu device fencing
    - Only GPUs allocated by scheduler are visible or accessible to user
    - GPU device numbers are virtualized, with a fixed mapping to a physical device per user environment
    - User always sees allocated GPU devices indexed from 0

CUDA/OpenCL Wrapper

- Features (cont'd):
  - Device Rotation (deprecated)
    - Virtual to Physical device mapping rotated for each process accessing a GPU device
    - Allowed for common execution parameters (e.g. Target gpu0 with 4 processes, each one gets separate gpu, assuming 4 gpus available)
    - CUDA 2.2 introduced *compute-exclusive* device mode, which includes fallback to next device. Device rotation feature may no longer needed.
  - Memory Scrubber
    - Independent utility from wrapper, but packaged with it
    - Linux kernel does no management of GPU device memory
    - Must run between user jobs to ensure security between users
- Availability
  - NCSA/UofI Open Source License
  - https://sourceforge.net/projects/cudawrapper/

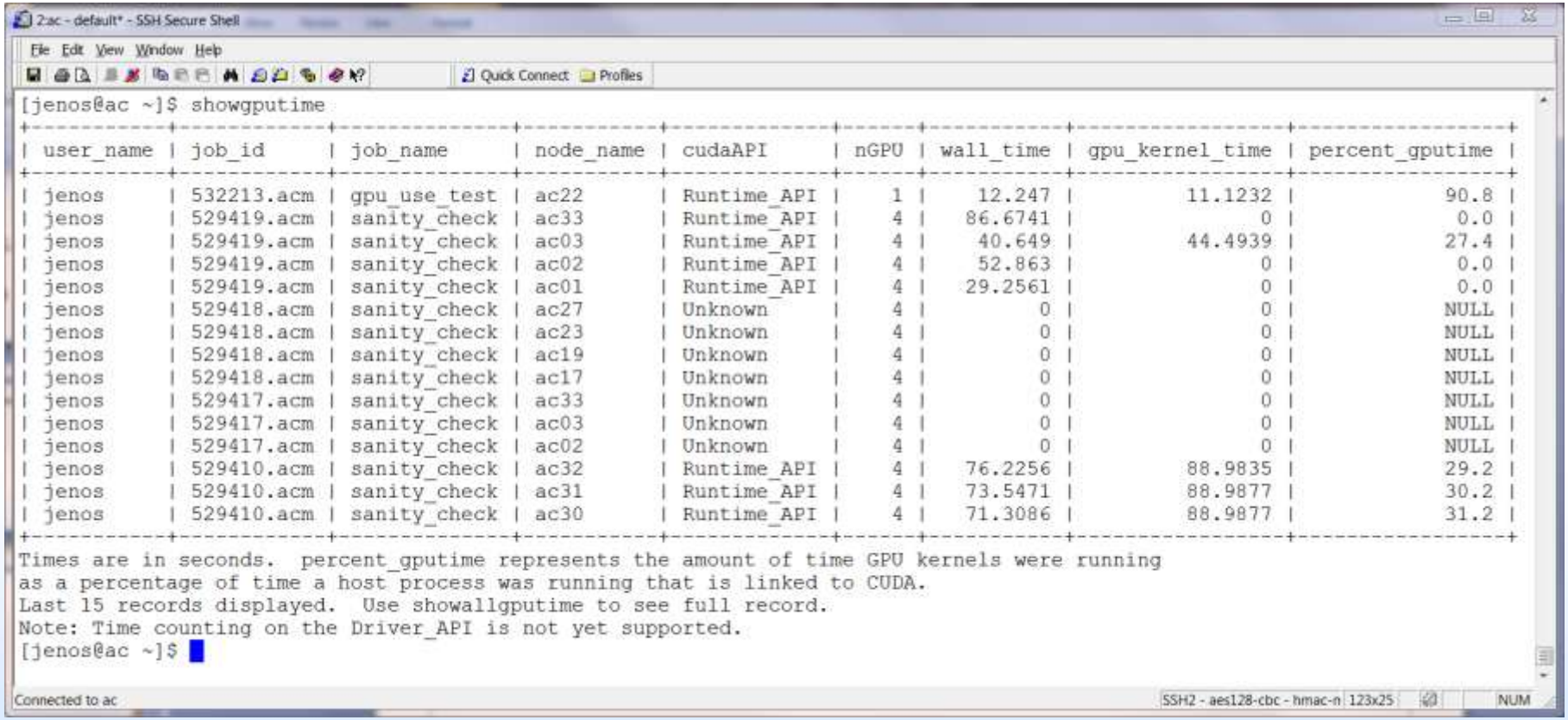
wrapper_query utility

- Within any job environment, get details on what the wrapper library is doing



showgputime

- Shows percent time CUDA linked processes utilized GPU
- Displays last 15 records (showallgputime shows all)
- Requires support of cuda_wrapper implementation



Are GPUs reliable?

- No ECC in initial product releases
  - Not a big deal when a GPU is used for what it was indented: image rendering
  - Could be a problem when executing a scientific application
    - Can we trust the computed results?
    - How do we know the results are correct?

- Fermi architecture now has ECC memory protection
- However, two years ago it was not clear if NVIDIA was going to add ECC
  - We have done a GPU memory reliability study

CUDA Memtest

- Features
  - Full re-implementation of every test included in memtest86
  - Random and fixed test patterns, error reports, error addresses, test specification
  - Includes additional stress test for software and hardware errors
  - Email notification
- Usage scenarios
  - Hardware test for defective GPU memory chips
  - CUDA API/driver software bugs detection
  - Hardware test for detecting soft errors due to non-ECC memory
  - Stress test for thermal loading
- No soft error detected in 2 years x 4 gig of cumulative runtime
- But several Tesla units in AC and Lincoln clusters were found to have hard memory errors (and thus have been replaced)
- Availability
  - NCSA/UofI Open Source License
  - https://sourceforge.net/projects/cudagpumemtest/

GPU Node Pre/Post Allocation Sequence

- Pre-Job (minimized for rapid device acquisition)
  - Assemble detected device file unless it exists
  - Sanity check results
  - Checkout requested GPU devices from that file
  - Initialize CUDA wrapper shared memory segment with unique key for user (allows user to ssh to node outside of job environment and have same gpu devices visible)
- Post-Job
  - Use quick memtest run to verify healthy GPU state
  - If bad state detected, mark node offline if other jobs present on node
  - If no other jobs, reload kernel module to “heal” node (for CUDA driver bug)
  - Run memscrubber utility to clear gpu device memory
  - Notify of any failure events with job details via email
  - Terminate wrapper shared memory segment
  - Check-in GPUs back to global file of detected devices

Are GPUs power-efficient?

- GPUs are power-hungry
  - GTX 480 - 250 W
  - C2050 - 238 W
- But does the increased power consumption justify their use?
  - How much power do jobs use?
  - How much do they use for pure CPU jobs vs. GPU-accelerated jobs?
  - Do GPUs deliver a hoped-for improvement in power efficiency?
  - How do we measure actual power consumption?
  - How do we characterize power efficiency?

Power Profiling Tools

Goals:

- Accurately record power consumption of GPU and workstation (CPU) for performance per watt efficiency comparison
- Make this data clearly and conveniently presented to application users
- Accomplish this with cost effective hardware

Solution:

- Modify inexpensive power meter to add logging capability
- Integrate monitoring with job management infrastructure
- Use web interface to present data in multiple forms to user

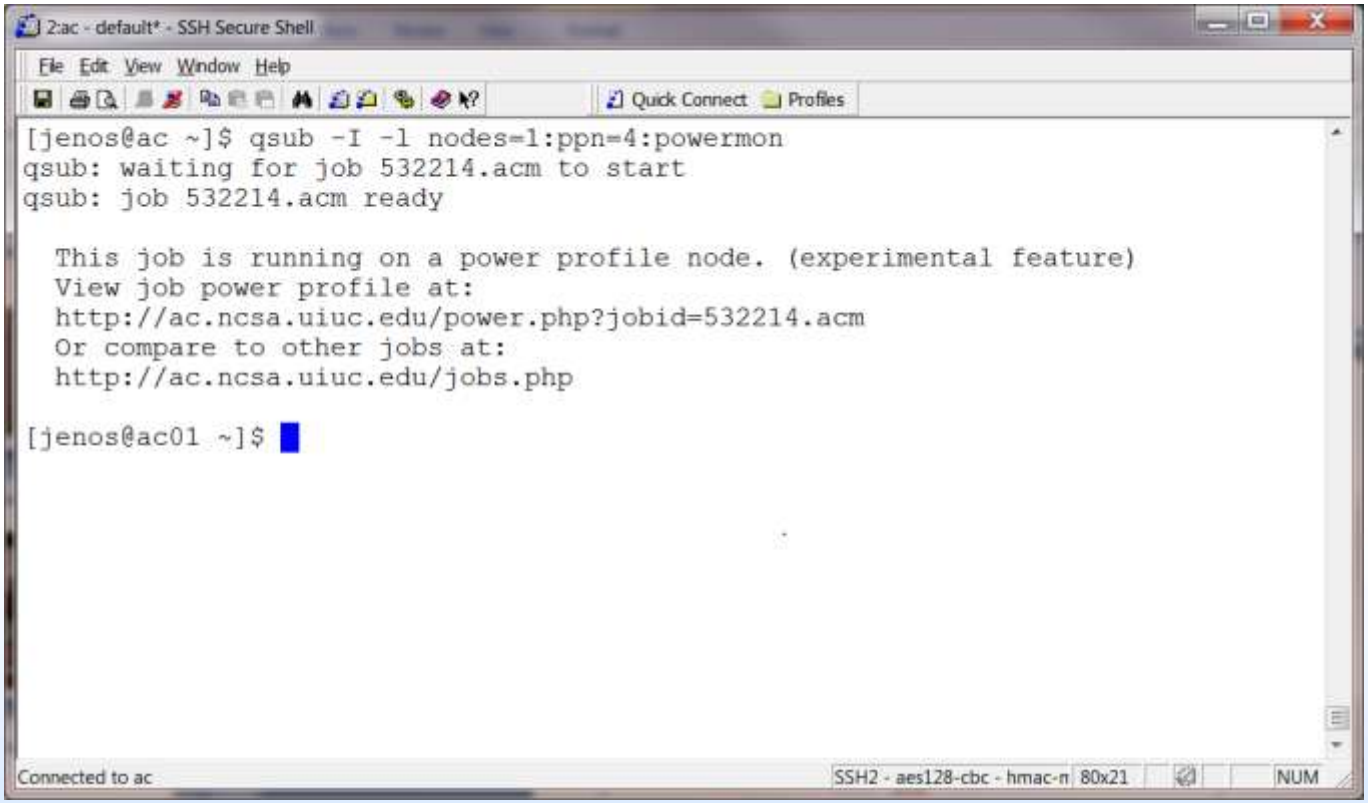
Power Profiling Hardware

- Tweet-a-Watt

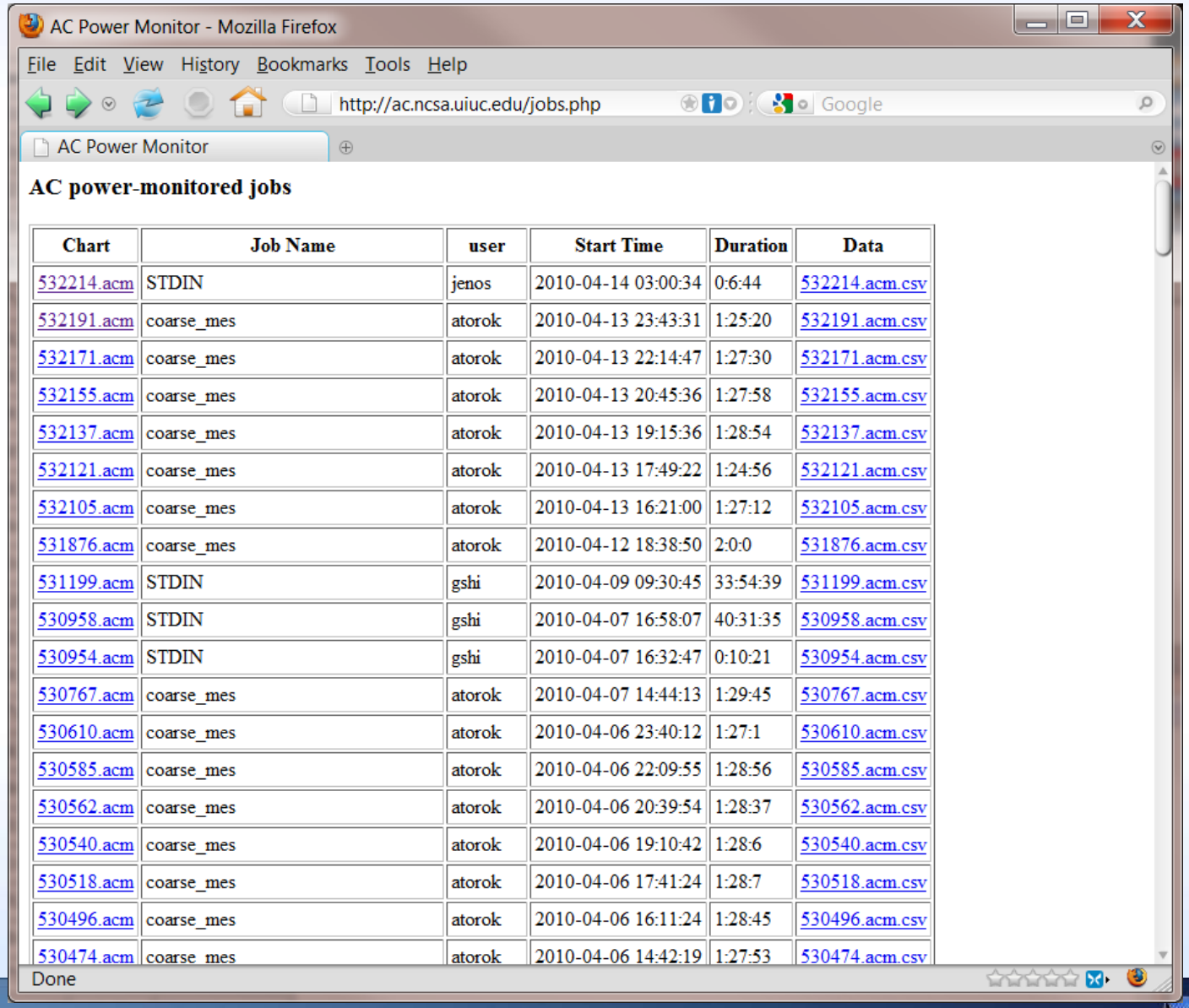
- Wireless receiver (USB)
- AC01 host and associated GPU unit are monitored separately by two Tweet-a-Watt transmitters
- Measurements are reported every 30 seconds
- < $100 total parts

Power Profiling Walk Through

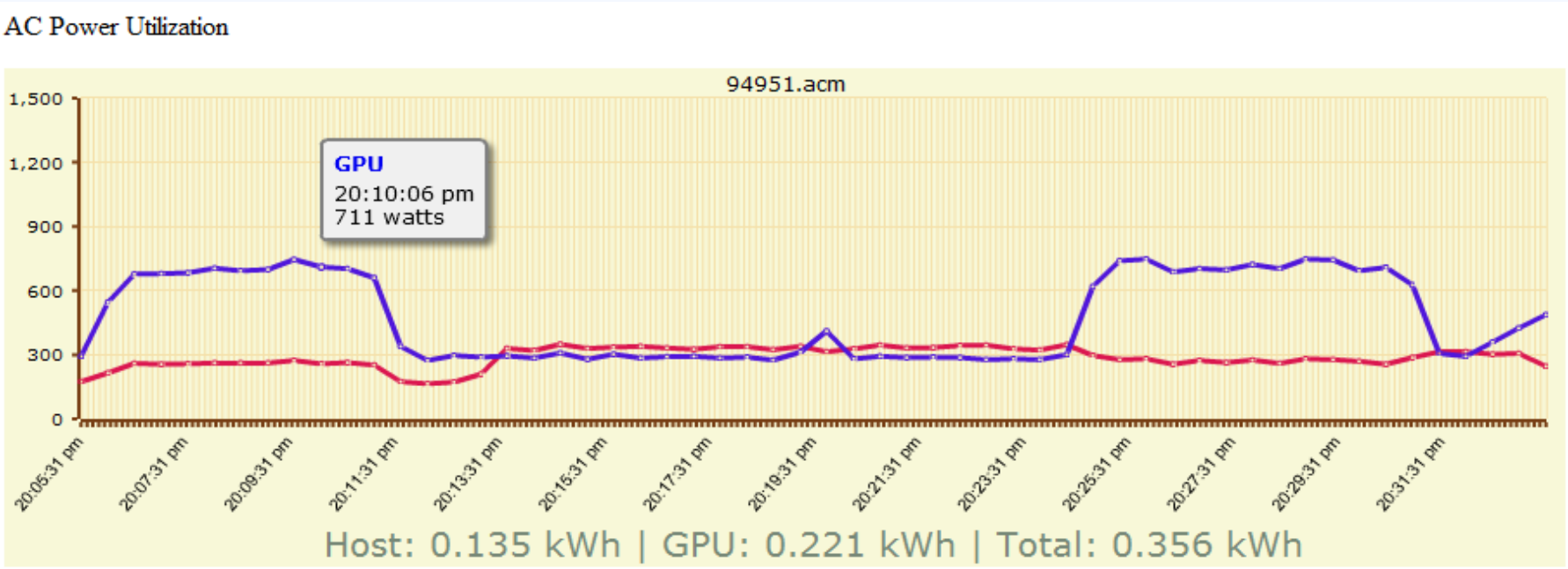
- Submit job with prescribed resource (powermon)
- Run application as usual, follow link(s)



Power Profiling Walk Through



Power Profiling Walk Through



JSON Data

- Mouse-over value displays
- Under curve totals displayed
- If there is user interest, we may support calls to add custom tags from application

AC GPU Cluster Power Considerations

State	Host Peak (Watt)	Tesla Peak (Watt)	Host power factor (pf)	Tesla power factor (pf)
power off	4	10	.19	.31
start-up	310	182		
pre-GPU use idle	173	178	.98	.96
after NVIDIA driver module unload/reload[1]	173	178	.98	.96
after deviceQuery[2] (idle)	173	365	.99	.99
GPU memtest #10 (stress)	269	745	.99	.99
after memtest kill (idle)	172	367	.99	.99
after NVIDIA module unload/reload[3] (idle)	172	367	.99	.99
VMD Madd	268	598	.99	.99
NAMD GPU STMV	321	521	.97-1.0	.85-1.0[4]
NAMD CPU only ApoA1	322	365	.99	.99
NAMD CPU only STMV	324	365	.99	.99

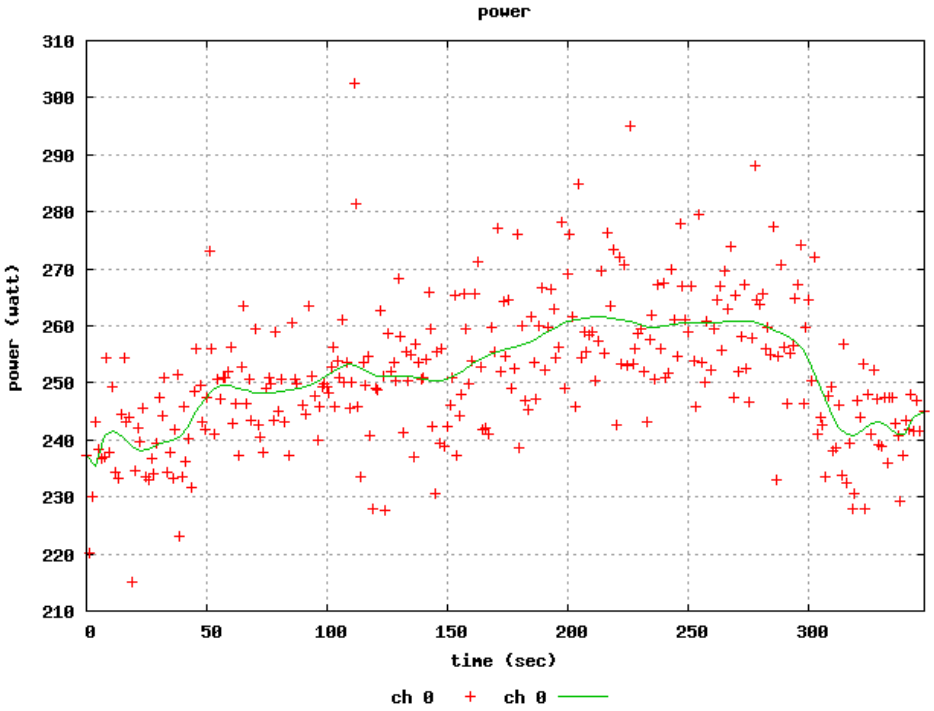
1. Kernel module unload/reload does not increase Tesla power
2. Any access to Tesla (e.g., deviceQuery) results in doubling power consumption after the application exits
3. Note that second kernel module unload/reload cycle does not return Tesla power to normal, only a complete reboot can
4. Power factor stays near one except while load transitions. Range varies with consumption swings

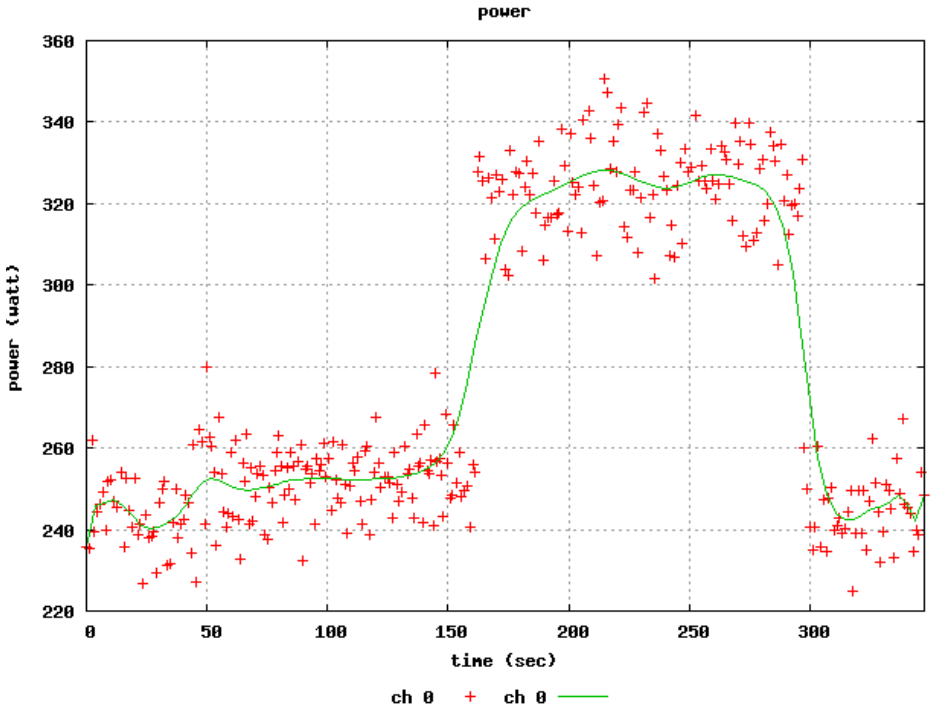
Power Profiling Hardware (2)

- Improved accuracy
- Increased granularity (every .20 seconds)
- Current flexible
- Voltage flexible
- 4 monitored ports
- ~$50 total parts

Application Power profile example

- No GPU used
  - 240 Watt idle
  - 260 Watt computing on a single core
- Computing on GPU
  - 240 Watt idle
  - 320 Watt computing on a single core





Improvement in Performance-per-watt

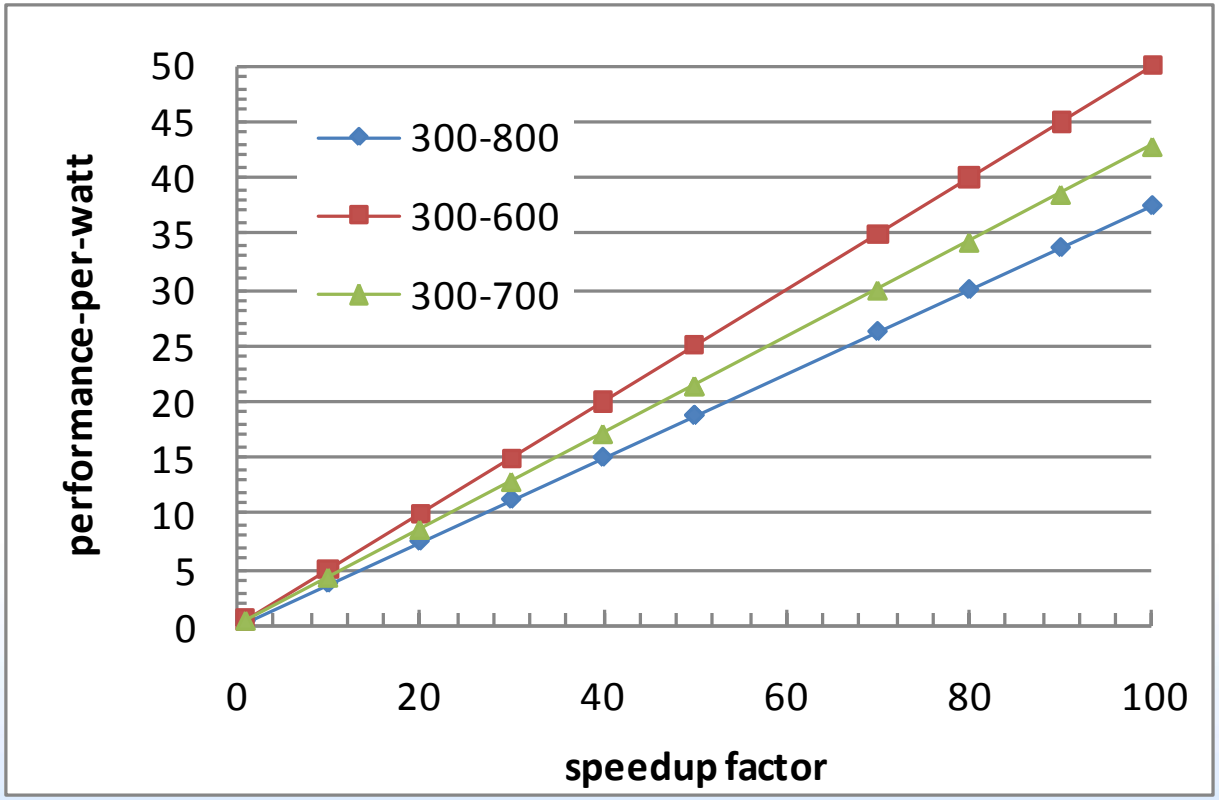
$$e = p/p_a * s$$

p – power consumption of non-accelerated application
p_a – power consumption of accelerated application
s – achieved speedup

Application	t (sec)	t_a (sec)	s	p (watt)	p_a (watt)	e
NAMD	6.6	1.1	6	316	681	2.78
VMD	1,465.2	57.5	25.5	299	742	10.48
QMCPACK			61.5	314	853	22.6
MILC	77,324	3,881	19.9	225	555	8.1

Speedup-to-Efficiency Correlation

- The GPU consumes roughly double the CPU power, so a 3x GPU is require to break even



Applications

- Applications
  - Cosmology
  - Computational chemistry
  - **Quantum chromodynamics**
  - Data mingin

Lattice QCD: MILC

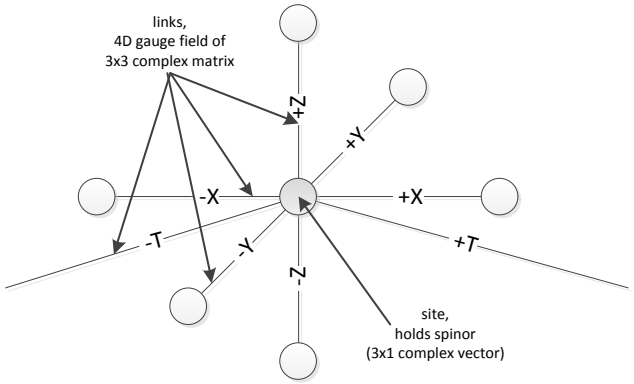
- Simulation of the 4D SU(3) lattice gauge theory
- Solve the space-time 4D linear system $M\phi = b$ using CG solver
  - $\phi_{i,x}$ and $b_{i,x}$ are complex variables carrying a color index $i = 1,2,3$ and a 4D lattice coordinate x.
  - $M = 2maI + D$
  - I is the identity matrix
  - $2ma$ is constant, and
  - matrix D (called "Dslash" operator) is given by

$$\mathrm{D_{x,i;y,j}} = \sum_{\mu=1}^{4} \left(\mathrm{U}^{F\,i,j}_{x,\mu} \delta_{\mathrm{y,x+\hat{\mu}}} - \mathrm{U}^{F\,\dagger\,i,j}_{x-\hat{\mu},\mu} \delta_{\mathrm{y,x-\hat{\mu}}} \right) + \sum_{\mu=1}^{4} \left(\mathrm{U}^{L\,i,j}_{x,\mu} \delta_{\mathrm{y,x+3\hat{\mu}}} - \mathrm{U}^{L\,\dagger\,i,j}_{x-3\hat{\mu},\mu} \delta_{\mathrm{y,x-3\hat{\mu}}} \right)$$

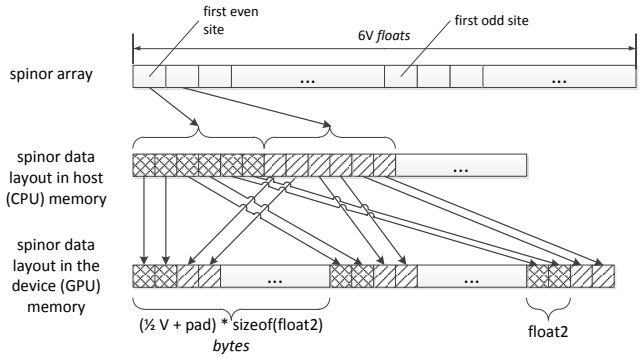
```
Input:
 D: dslash operator
 A_h: D^+D + 4ma^2 in high precision
 A_l: D^+D + 4ma^2 in low precision
 b: source vector
 x: guess solution vector
Output:
 y: solution vector

y ← 0
i ← 0
r ← b - A_h x                                   D+K7
d ← r
δnew ← r^T r                                    K1
δ_0 ← b^T b                                     K1
while i < i_max and δ > δ_0 ε^2 do
  q ← A_l d                                     D
  α ← δnew / (d^T q)                            K2
  δ_old ← δ_new
  r ← r - α q                                   K3
  δ_new ← r^T r
  β ← δ_new / δ_old
  if (high-precision update is needed)
    x ← x + α d                                 K4
    y ← x + y                                   K5
    r ← b - A_h y                               D+K7
    δ_new ← r^T r                               K1
    x ← 0
    β ← δ_new / δ_old
    d ← r + β d                                 K8
  else
    x ← x + α d                                 K6
    d ← r + β d
  i ← i + 1
```

GPU Implementation strategy: optimize for memory bandwidth

- 4D lattice

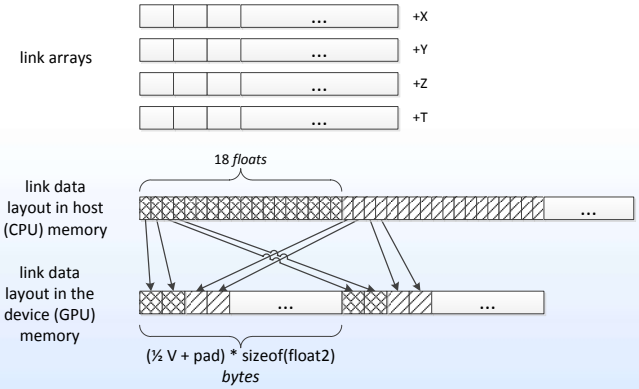


- spinor data layout



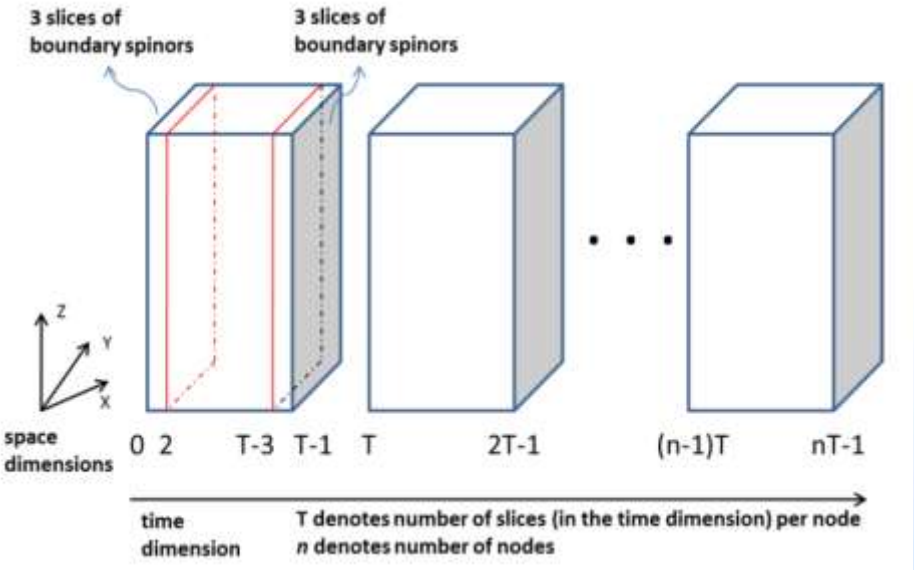
- flop-to-byte ratio of the Dslash operation is 1,146/1,560=0.73
- flop-to-byte ratio supported by the C2050 hardware is 1,030/144=7.5
- thus, the Dslash operation is memory bandwidth-bound

- link data layout



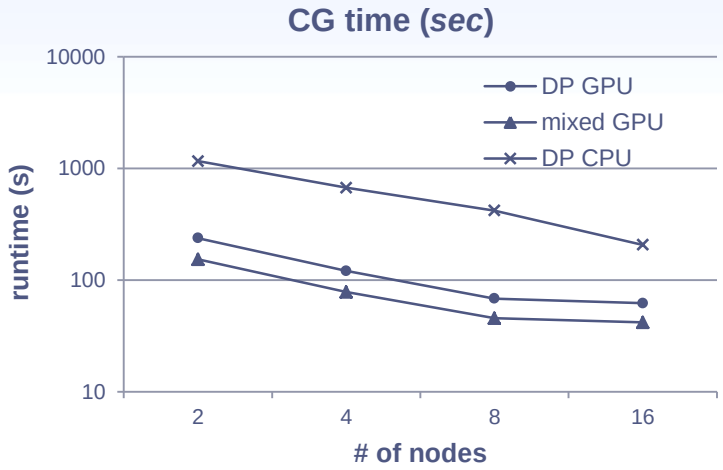
Parallelization strategy: split in T dimension

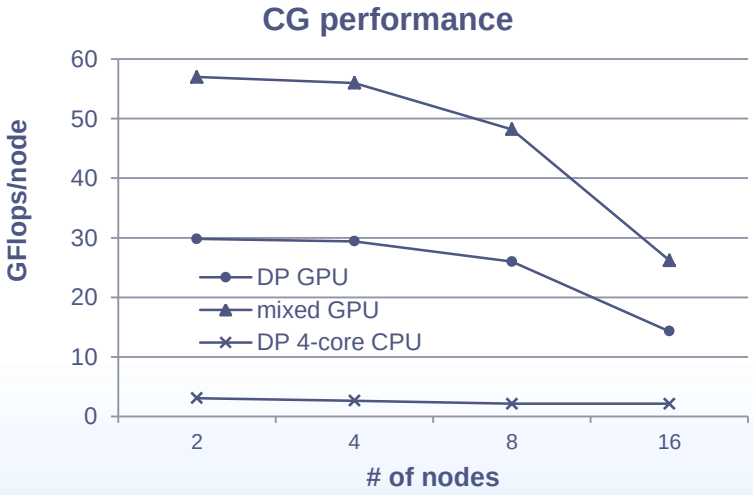
- 4D lattice is partitioned in the time dimension, each node computes T slices
- Three slices in both forward and backward directions are needed by the neighbors in order to compute new spinors

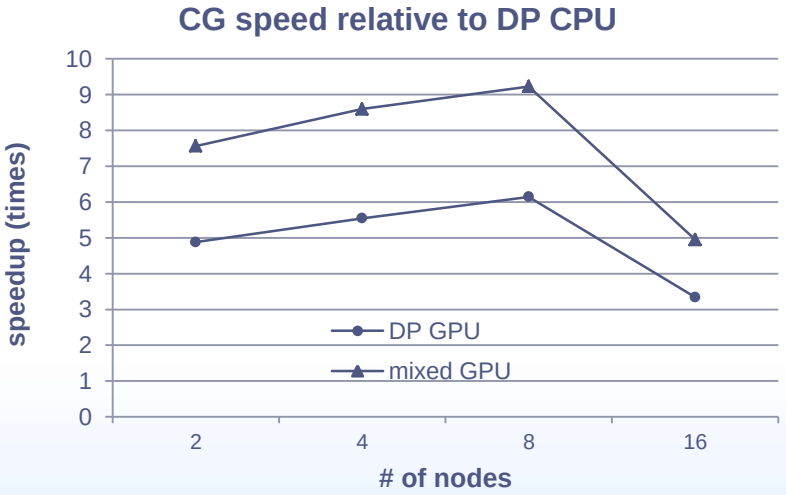


- dslash kernel is split into
  - interior kernel which computes the internal slices (2<t<T-3) of sub-lattice and the space contribution of the boundary sub-lattices, and
  - exterior kernel which computes the time dimenstion contribution for the boundary sub-lattice. The exterior kernel depends on the data from the neighbors.
- The interior kernel and the communication of boundary data can be overlapped using CUDA streams

Results for CG solver alone



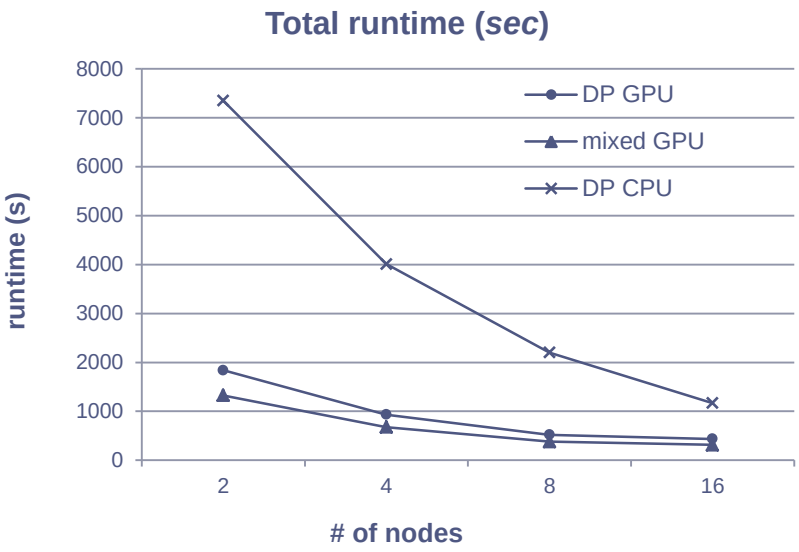


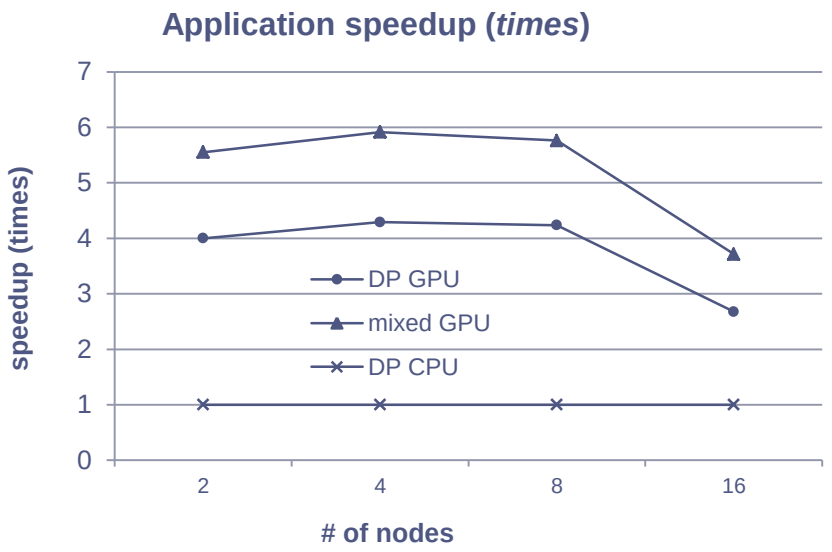


one CPU node = 8 Intel Nehalem 2.4 Ghz CPU cores
one GPU node = 1 CPU core + 1 C2050 GPU

lattice size $28^3 \times 96$

Results for entire application (Quantum Electrodynamics)





References

- GPU clusters
  - V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. Hwu, *GPU Clusters for High-Performance Computing*, in Proc. IEEE International Conference on Cluster Computing, Workshop on Parallel Programming on Accelerator Clusters, 2009.
  - M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, W. Hwu, *QP: A Heterogeneous Multi-Accelerator Cluster*, In Proc. 10th LCI International Conference on High-Performance Clustered Computing – LCI'09, 2009.
- Memory reliability
  - G. Shi, J. Enos, M. Showerman, V. Kindratenko, *On testing GPU memory for hard and soft errors*, in Proc. Symposium on Application Accelerators in High-Performance Computing – SAAHPC'09, 2009
- Power efficiency
  - J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, J. Phillips, *Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters*, In Proc. Work in Progress in Green Computing, 2010.
- Applications
  - S. Gottlieb, G. Shi, A. Torok, V. Kindratenko, *QUDA programming for staggered quarks*, In Proc. The XXVIII International Symposium on Lattice Field Theory – Lattice'10, 2010.
  - G. Shi, S. Gottlieb, A. Totok, V. Kindratenko, *Accelerating Quantum Chromodynamics Calculations with GPUs*, In Proc. Symposium on Application Accelerators in High-Performance Computing - SAAHPC'10, 2010.
  - A. Titov, V. Kindratenko, I. Ufimtsev, T. Martinez, *Generation of Kernels to Calculate Electron Repulsion Integrals of High Angular Momentum Functions on GPUs – Preliminary Results*, In Proc. Symposium on Application Accelerators in High-Performance Computing - SAAHPC'10, 2010.
  - G. Shi, I. Ufimtsev, V. Kindratenko, T. Martinez, *Direct Self-Consistent Field Computations on GPU Clusters*, In Proc. IEEE International Parallel and Distributed Processing Symposium – IPDPS, 2010.
  - D. Roeh, V. Kindratenko, R. Brunner, *Accelerating Cosmological Data Analysis with Graphics Processors*, In Proc. 2nd Workshop on General-Purpose Computation on Graphics Processing Units – GPGPU-2, pp. 1-8, 2009.