GPU Clusters for High-Performance Computing

Jeremy Enos

Innovative Systems Laboratory

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

Presentation Outline

NVIDIA GPU technology overview

GPU clusters at NCSA

- AC
- Lincoln

GPU power consumption

Programming tools

- CUDA C
- OpenCL
- PGI x86+GPU

Application performance

GPU cluster management software

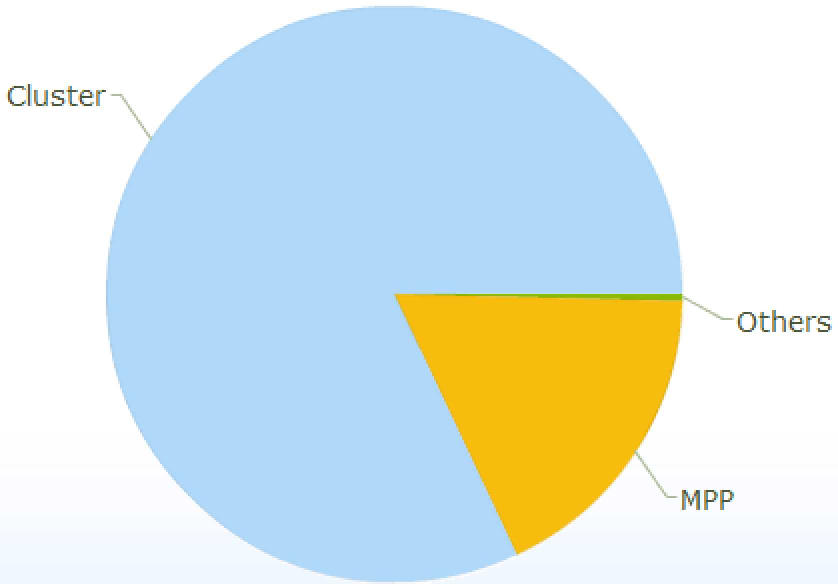
- The need for new tools
- CUDA/OpenCL wrapper library
- CUDA memtest

Balanced GPU accelerated system design considerations

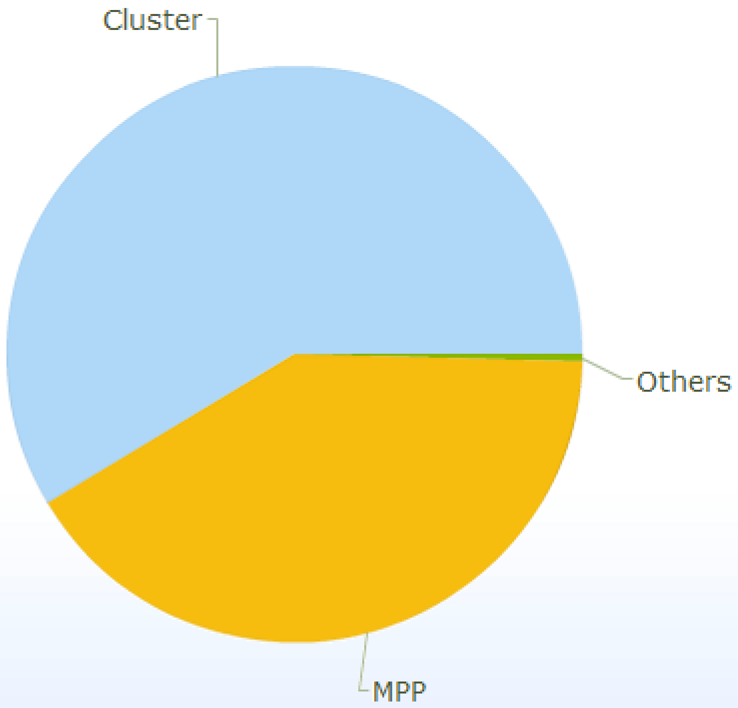
Conclusions

Why Clusters for HPC?

- **Clusters are a major workforce in HPC**
  - Q: How many systems in top500 are clusters?
  - A: 410 out of 500

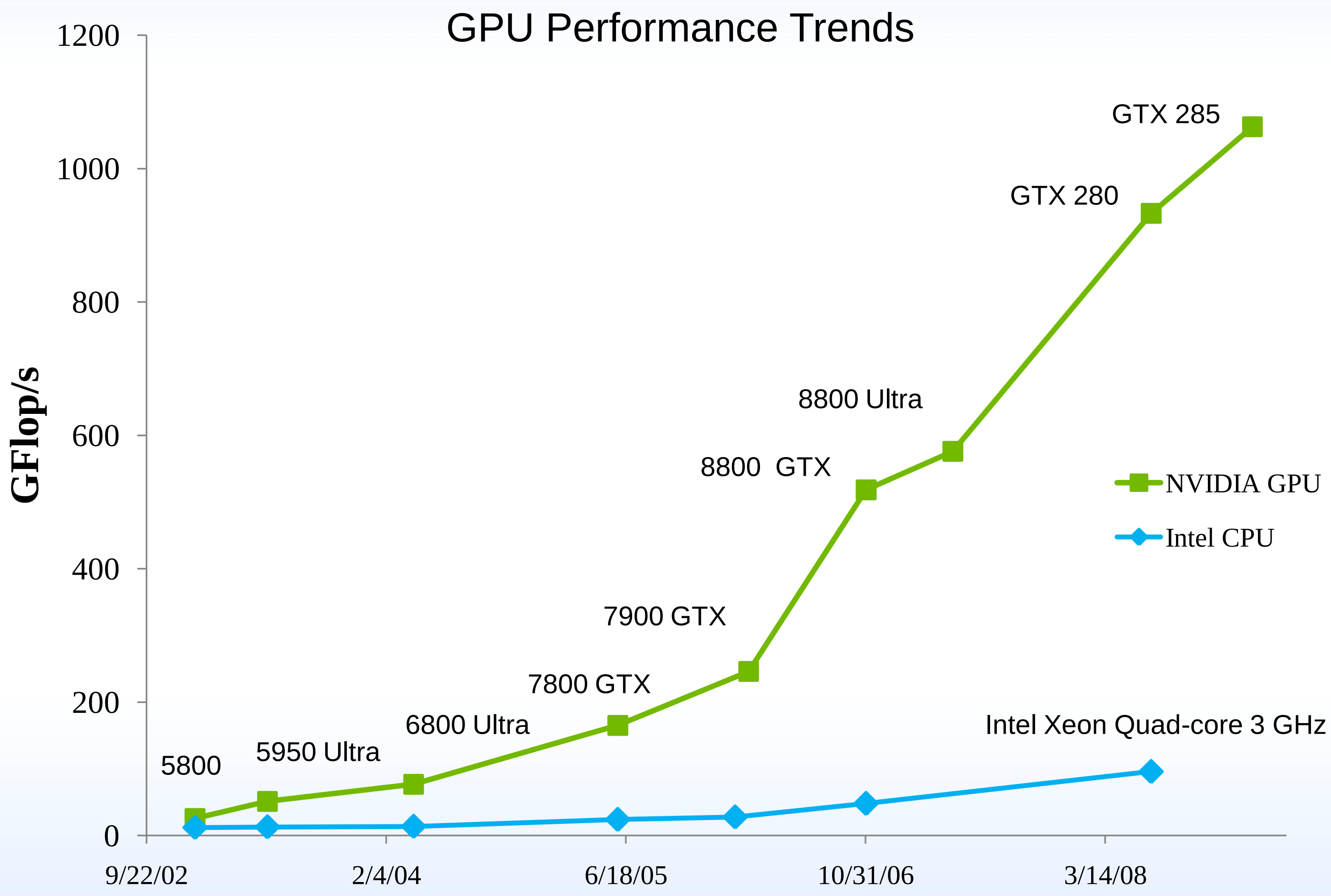


Top 500: Architecture

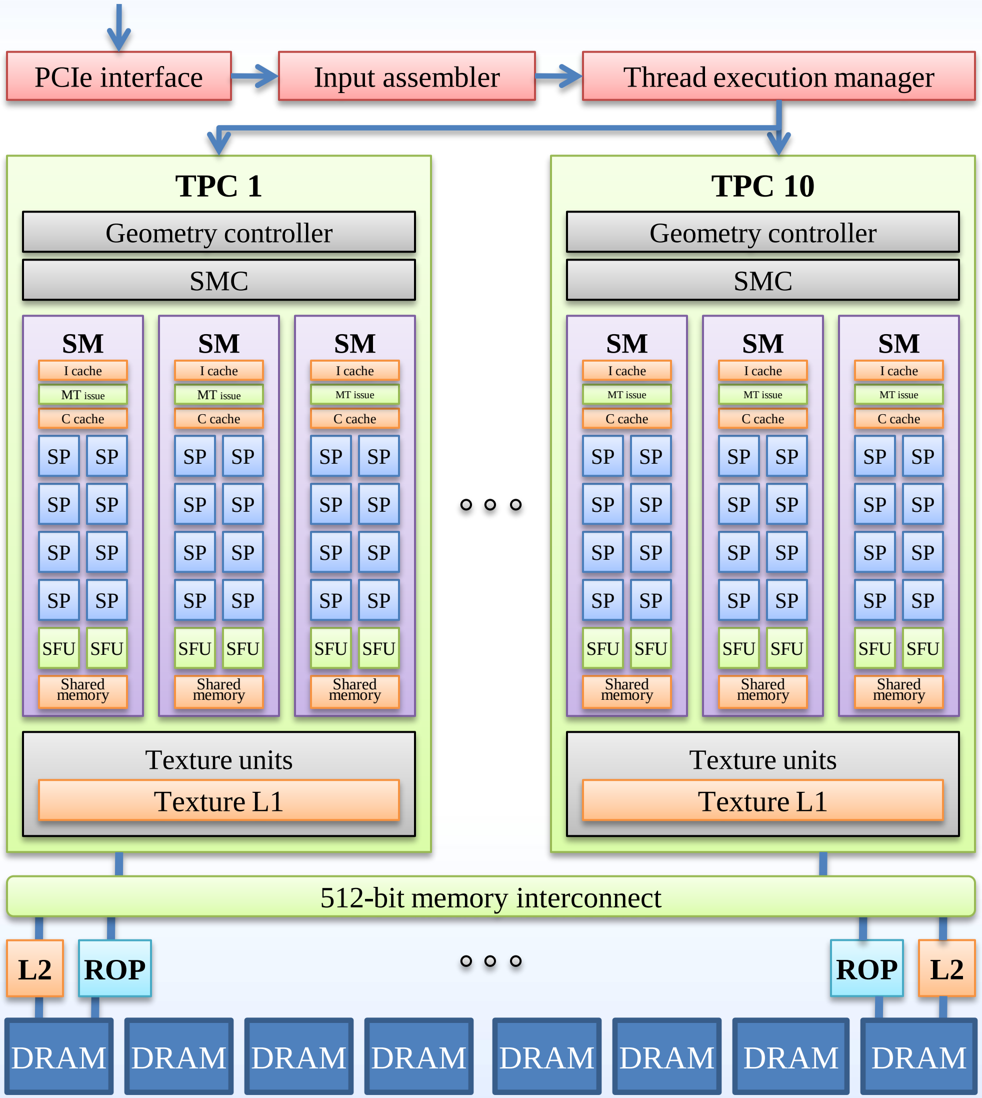


Top 500: Performance

Why GPUs in HPC Clusters?



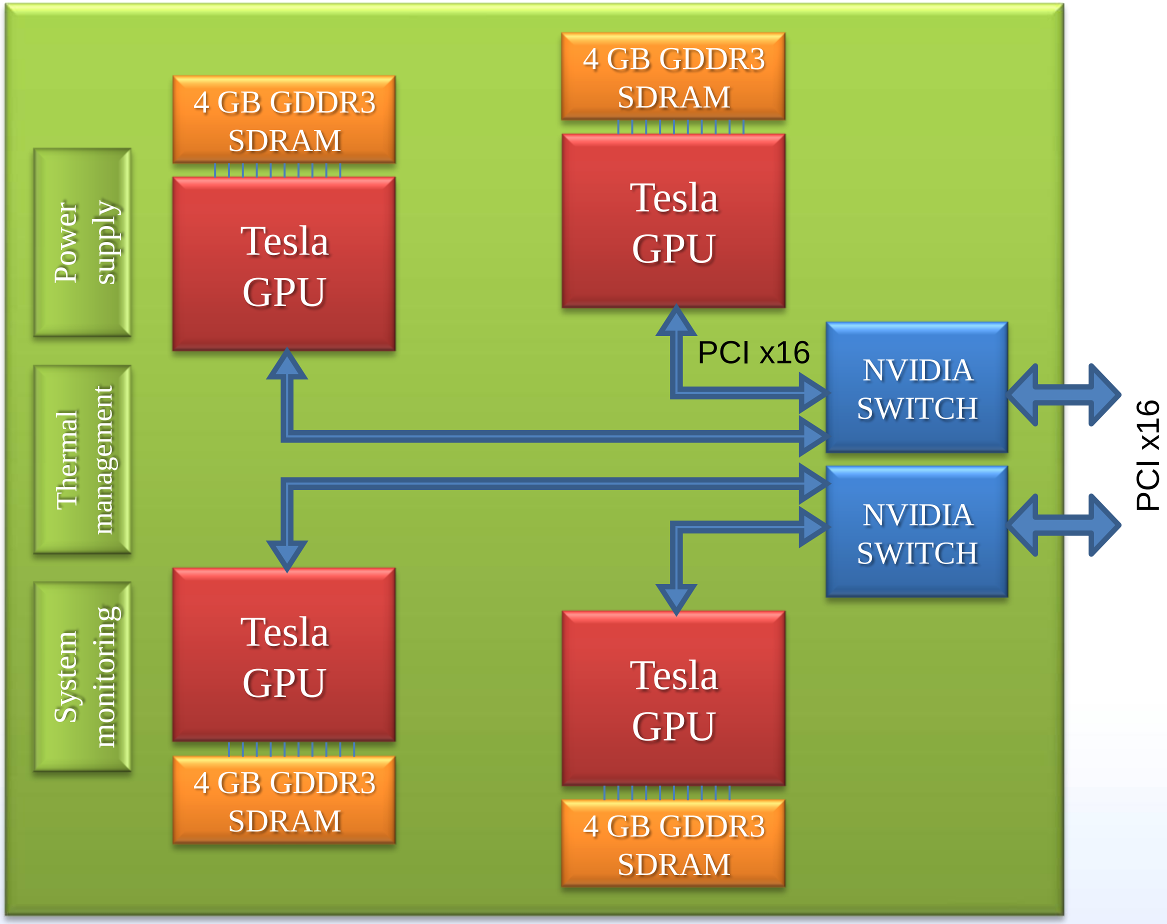
NVIDIA Tesla T10 GPU Architecture



- **T10 architecture**
  - 240 streaming processors arranged as 30 streaming multiprocessors
  - At 1.3 GHz this provides
    - 1 TFLOP SP
    - 86.4 GFLOP DP
  - 512-bit interface to off-chip GDDR3 memory
    - 102 GB/s bandwidth

NVIDIA Tesla S1070 GPU Computing Server

- **4 T10 GPUs**



GPU Clusters at NCSA

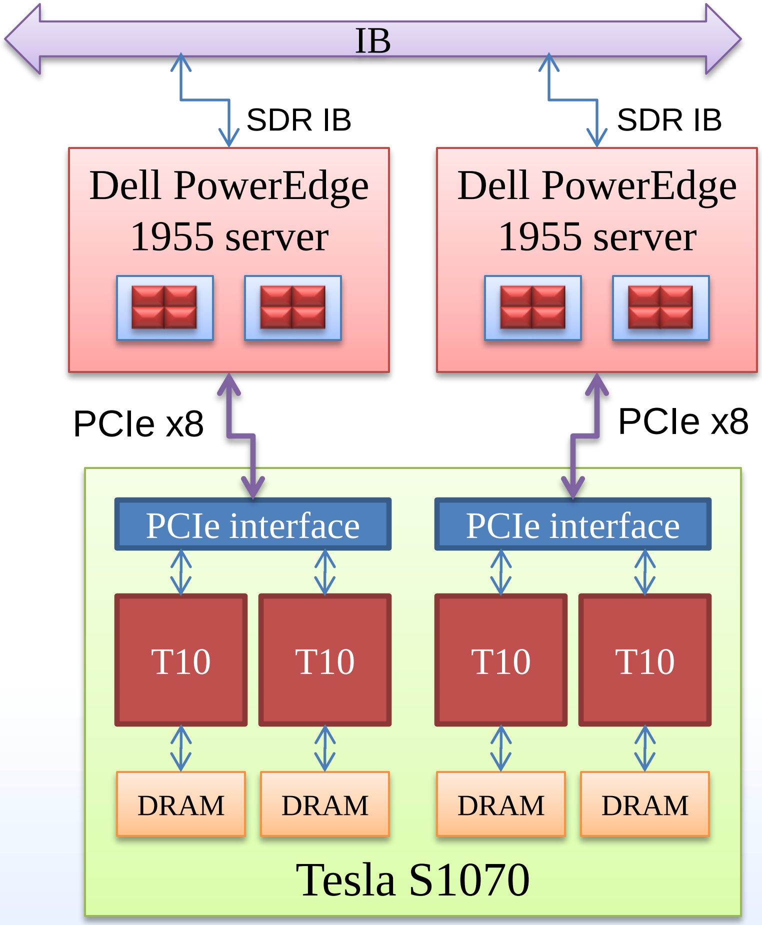
- **Lincoln**
  - Production system available via the standard NCSA/TeraGrid HPC allocation

- **AC**
  - Experimental system available for anybody who is interested in exploring GPU computing

Intel 64 Tesla Linux Cluster *Lincoln*

- **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

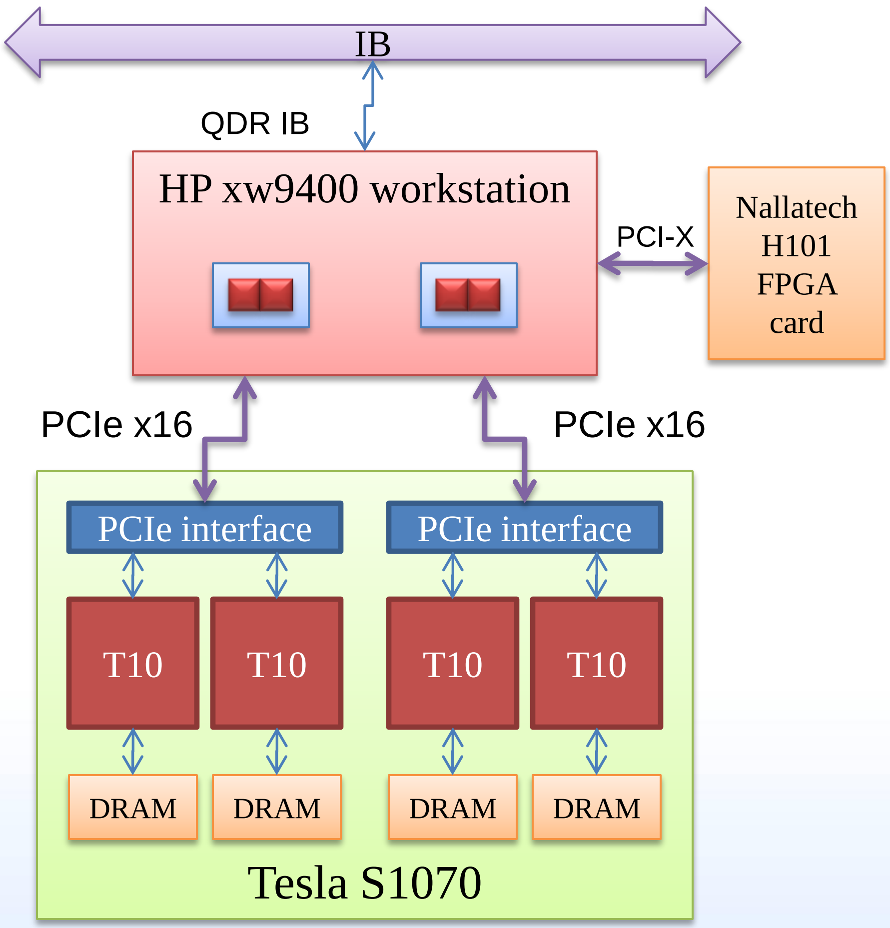
- **Two Compute Nodes**



AMD Opteron Tesla Linux Cluster *AC*

- **HP xw9400 workstation**
  - 2216 AMD Opteron 2.4 GHz dual socket dual core
  - 8 GB DDR2
  - Infiniband QDR
- **Tesla S1070 1U GPU Computing Server**
  - 1.3 GHz Tesla T10 processors
  - 4x4 GB GDDR3 SDRAM
- **Cluster**
  - Servers: 32
  - Accelerator Units: 32

- **Compute Node**



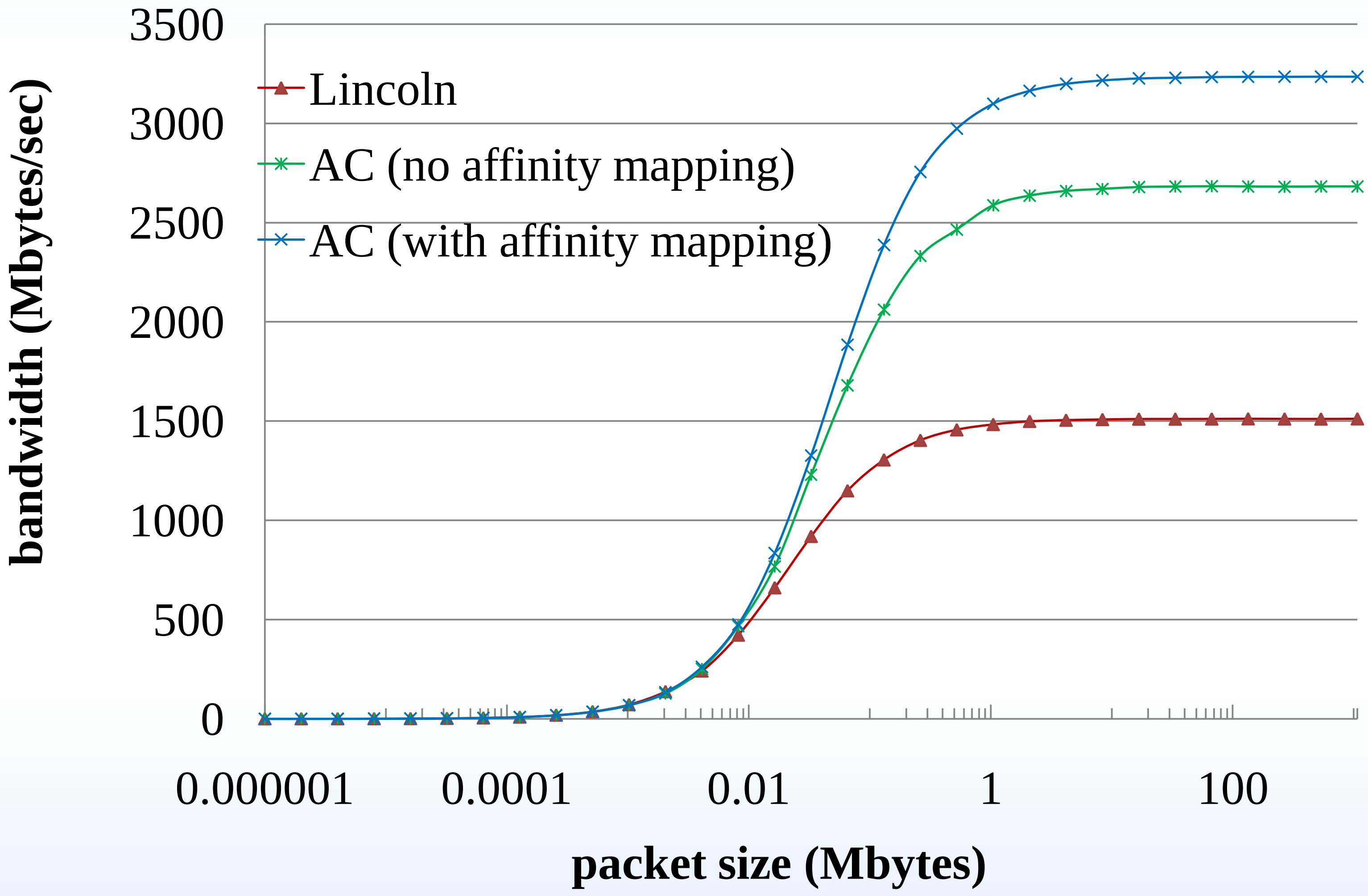
AC Cluster

Lincoln vs. AC: Configuration

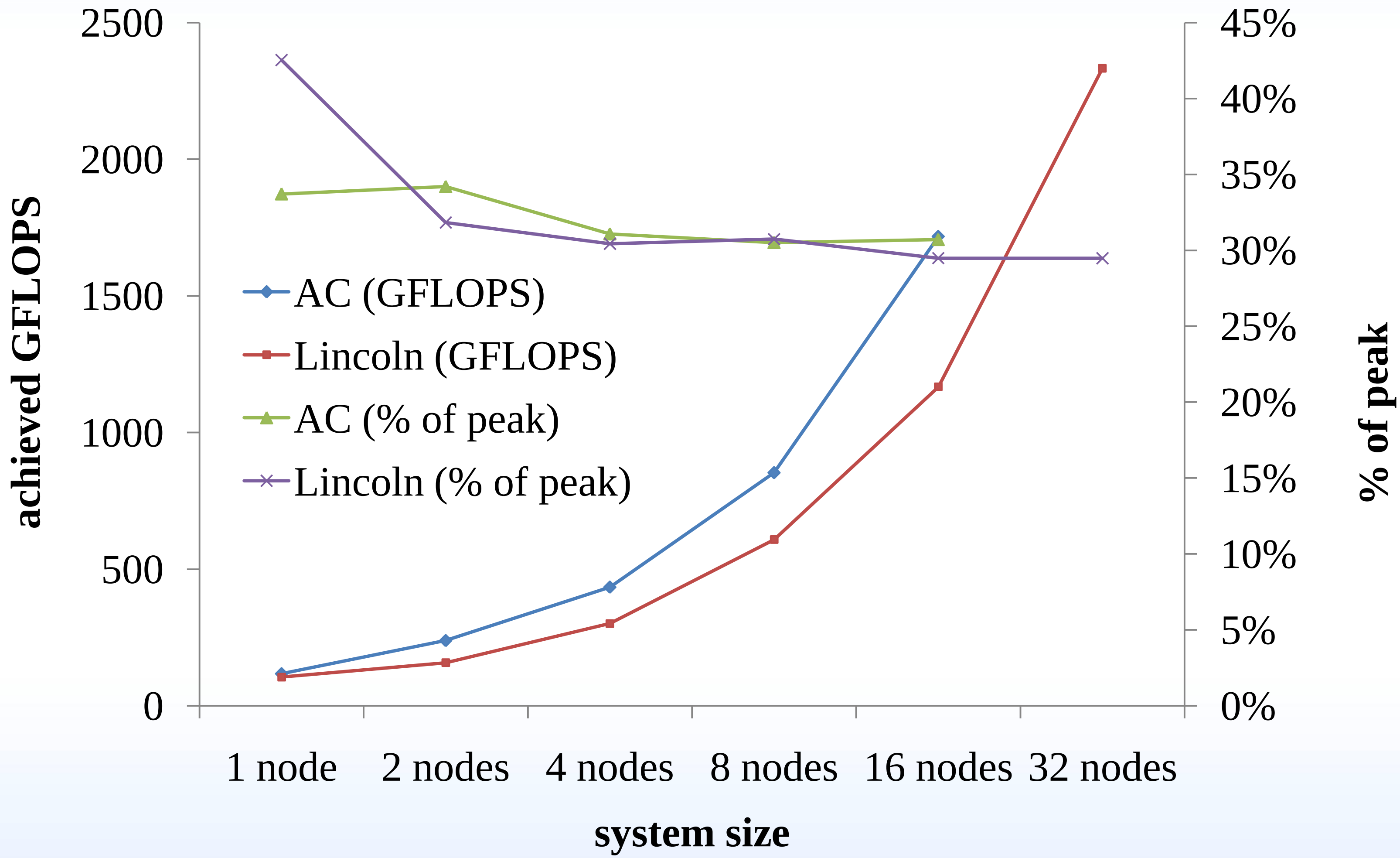
- **Lincoln**
  - Compute cores
    - CPU cores: 1536
    - GPU units: 384
    - CPU/GPU ratio: 4
  - Memory
    - Host memory: 16 GB
    - GPU Memory: 8 GB/host
    - Host mem/GPU: 8 GB
  - I/O
    - PCI-E 2.0 (x8)
    - GPU/host bandwidth: 4 GB/s
    - IB bandwidth/host: 8 Gbit/s
- **AC**
  - Compute cores
    - CPU cores: 128
    - GPU units: 128
    - CPU/GPU ratio: 1
  - Memory
    - Host memory: 8 GB
    - GPU Memory: 16 GB/host
    - Host mem/GPU: 2 GB
  - I/O
    - PCI-E 1.0 (x16)
    - GPU/host bandwidth: 4 GB/s
    - IB bandwidth/host: 16 Gbit/s

Both systems originated as extensions of existing clusters, they were not designed as Tesla GPU clusters from the beginning. As the result, their performance with regards to GPUs is suboptimal.

Lincoln vs. AC: Host-device Bandwidth



Lincoln vs. AC: HPL Benchmark



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

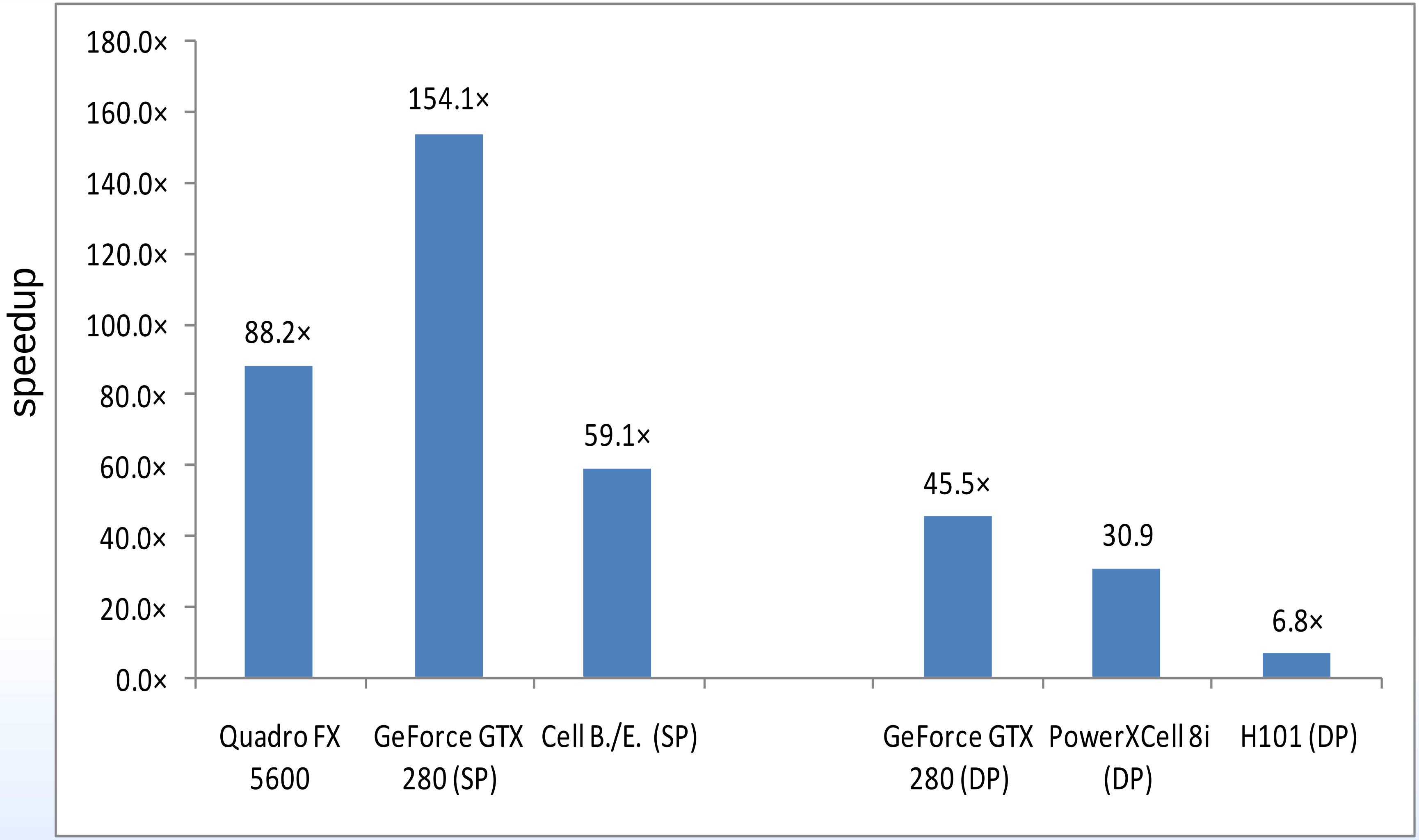
Cluster Management Tools

- **Deployment**
  - SystemImager (AC)
  - Perceus (Lincoln)
- **Workload Management**
  - Torque/MOAB
  - access.conf restrictions unless node used by user job
  - Job preemption config used to run *GPU memtest* during idle periods (AC)
- **Monitoring**
  - CluMon (Lincoln)
  - Manual monitoring (AC)

GPU Clusters Programming

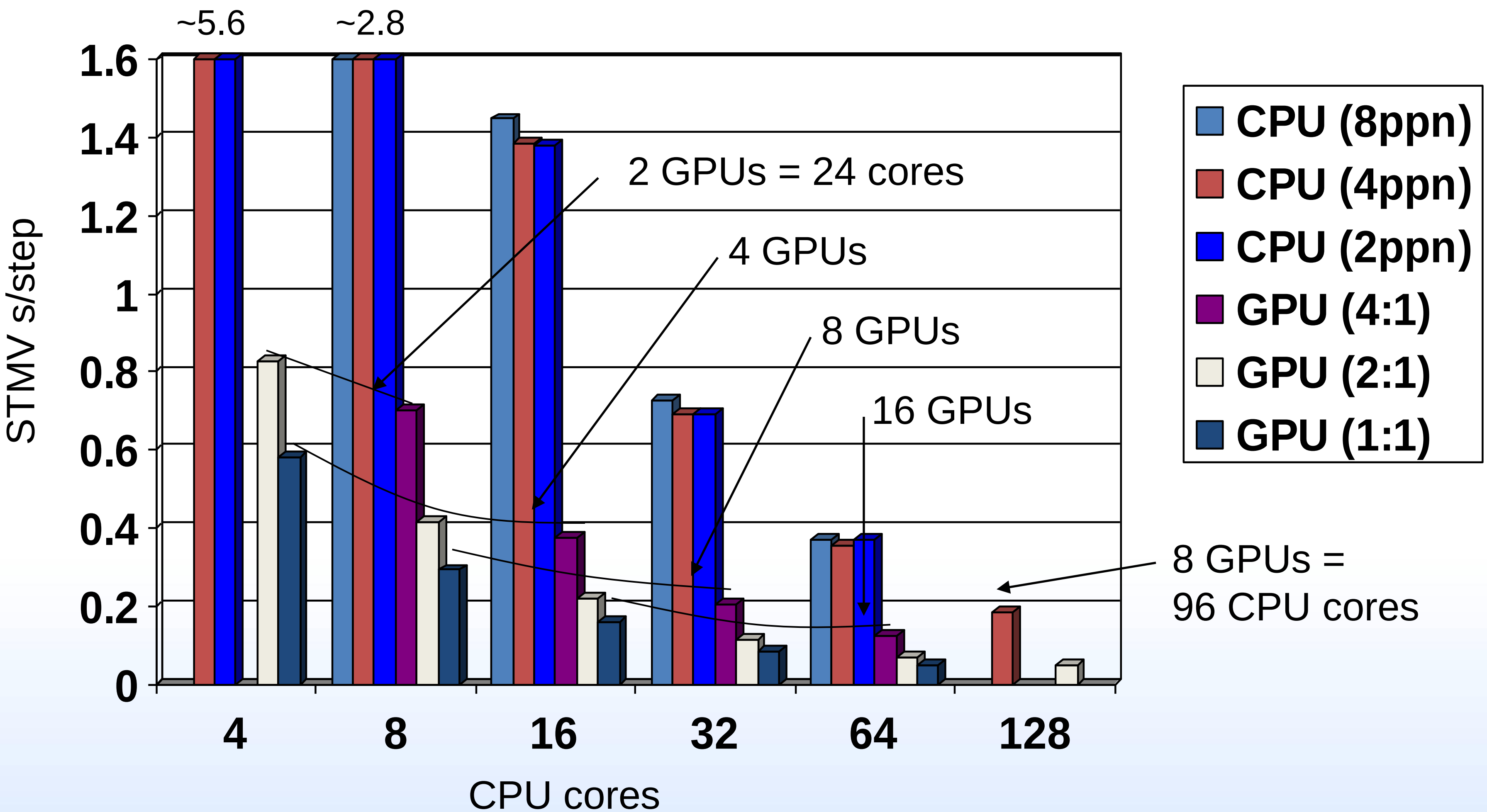
- **Programming tools**
  - CUDA C 2.2 SDK
    - CUDA/MPI
  - OpenCL 1.0 SDK
  - PGI+GPU compiler
- **Some Applications (that we are aware of)**
  - NAMD (Klaus Schulten's group at UIUC)
  - WRF (John Michalakes, NCAR)
  - TPACF (Robert Brunner, UIUC)
  - TeraChem (Todd Martinez, Stanford)
  - …

TPACF

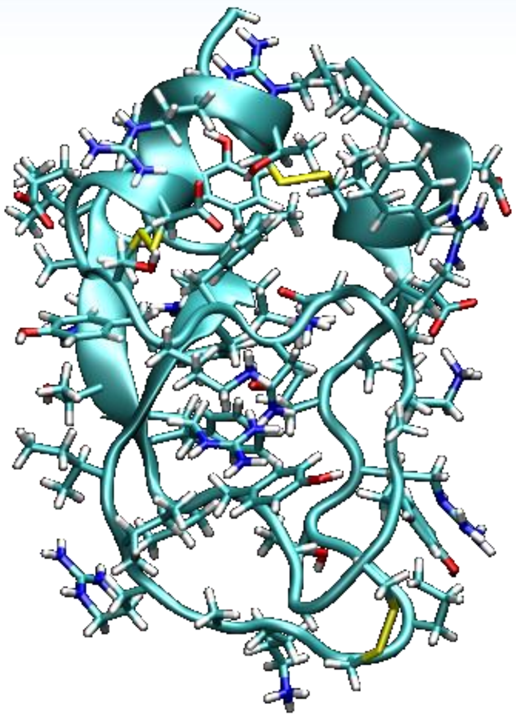


speedup
180.0×
160.0×
140.0×
120.0×
100.0×
80.0×
60.0×
40.0×
20.0×
0.0×
88.2×
154.1×
59.1×
45.5×
30.9
6.8×
Quadro FX 5600
GeForce GTX 280 (SP)
Cell B./E. (SP)
GeForce GTX 280 (DP)
PowerXCell 8i (DP)
H101 (DP)

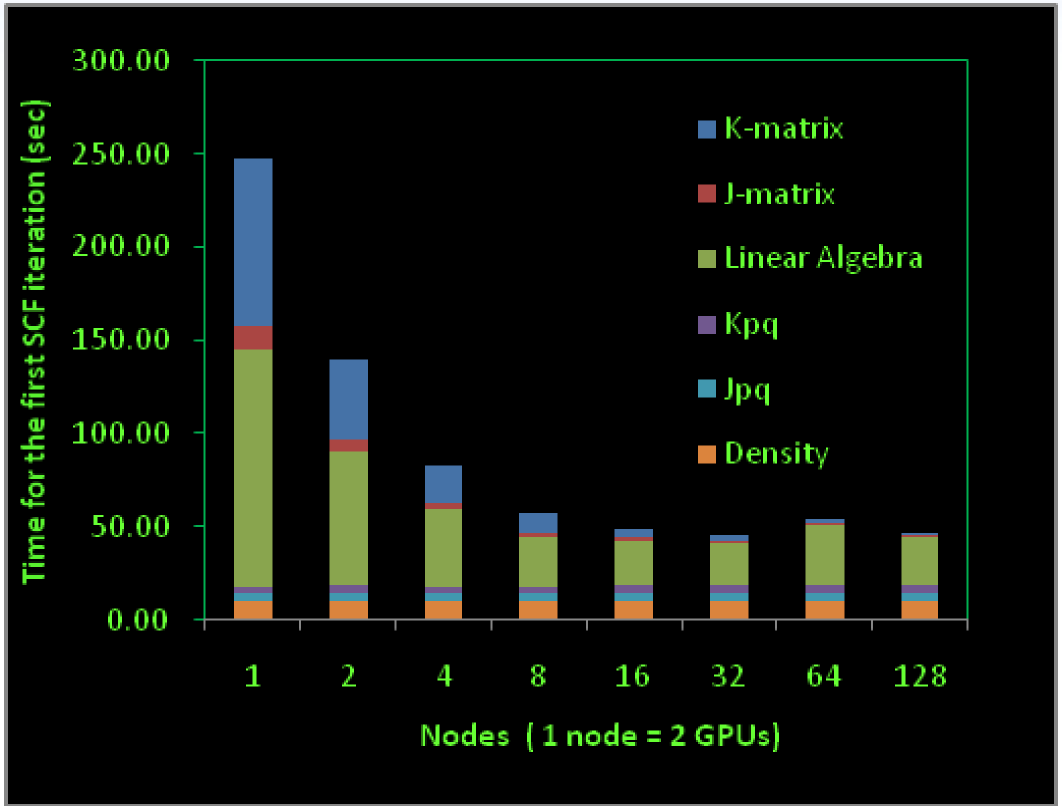
NAMD on Lincoln Cluster Performance
(8 cores and 2 GPUs per node, very early results)



TeraChem



Bovine pancreatic trypsin inhibitor (BPTI) 3-21G, 875 atoms, 4893 basis functions



MPI timings and scalability

GPU (computed with SP): K-matrix, J-matrix
CPU (computed with DP): the rest

CUDA Memtest

- **4GB of Tesla GPU memory is not ECC protected**
- **Hunt for "soft error"**
- **Features**
  - Full re-implementation of every test included in memtest86
  - Random and fixed test patterns, error reports, error addresses, test specification
  - Email notification
  - Includes additional stress test for software and hardware errors
- **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
- **No soft error detected in 2 years x 4 gig of cumulative runtime**
- **Availability**
  - NCSA/UofI Open Source License
  - https://sourceforge.net/projects/cudagpumemtest/

CUDA/OpenCL Wrapper Library

- **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
- **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 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 Library

- **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/

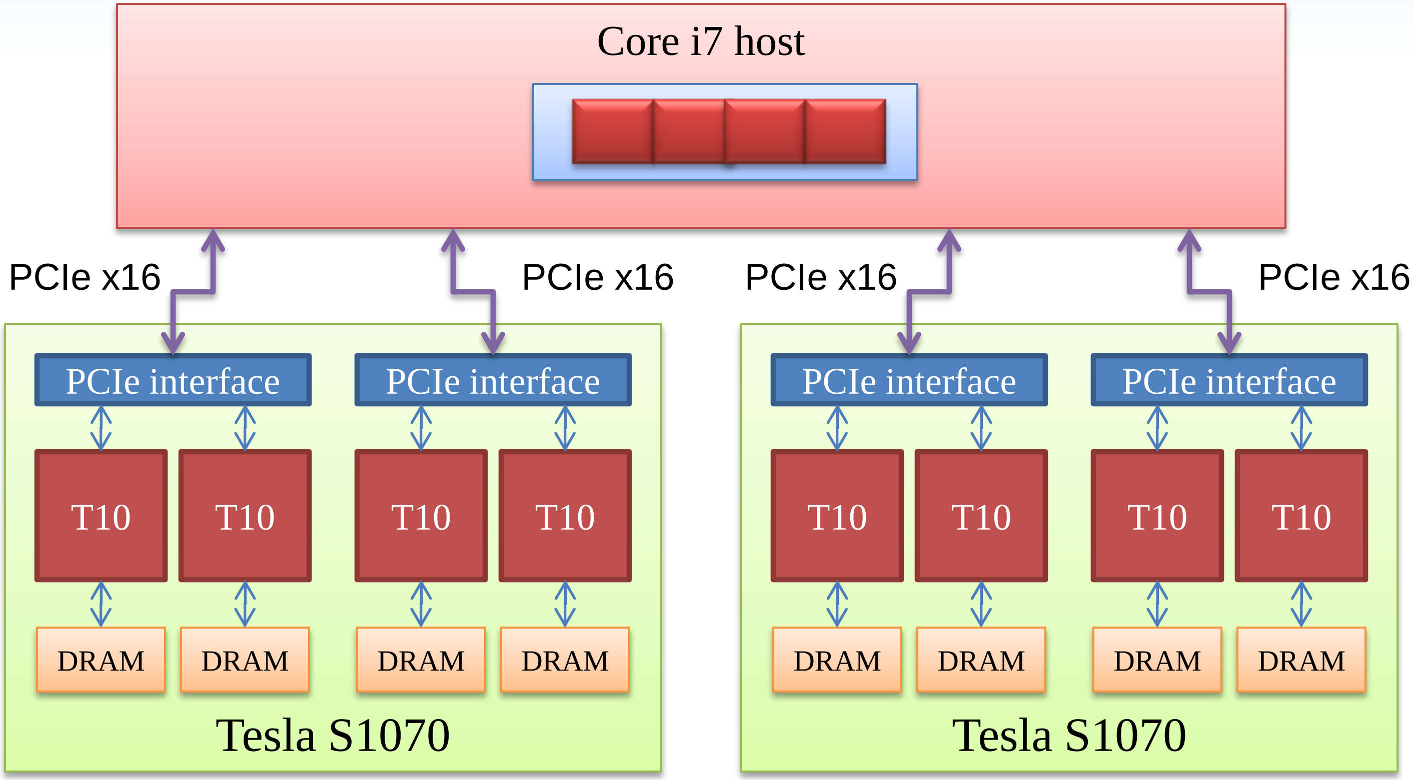
GPU Node Pre/Post Allocation Sequence

- **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 2.2 driver bug)
  - Run memscrubber utility to clear gpu device memory
  - Notify of any failure events with job details
  - Terminate wrapper shared memory segment
  - Check-in GPUs back to global file of detected devices

Designing a Balanced Compute Host

- **Consider:**
- **Potential performance bottlenecks**
  - Memcopy bandwidth is limiting factor to significant portion of applications
  - FLOP:I/O ratio increasing relative to existing general purpose clusters
  - Current NVIDIA driver limited to 8 GPU support
  - PCI-E lane oversubscription
  - Host memory bandwidth
- **User feedback**
  - Typically want at least as many CPU cores as GPU units
  - Typically want host memory to match or surpass attached GPU total
- **Get candidate test system before locking decision**

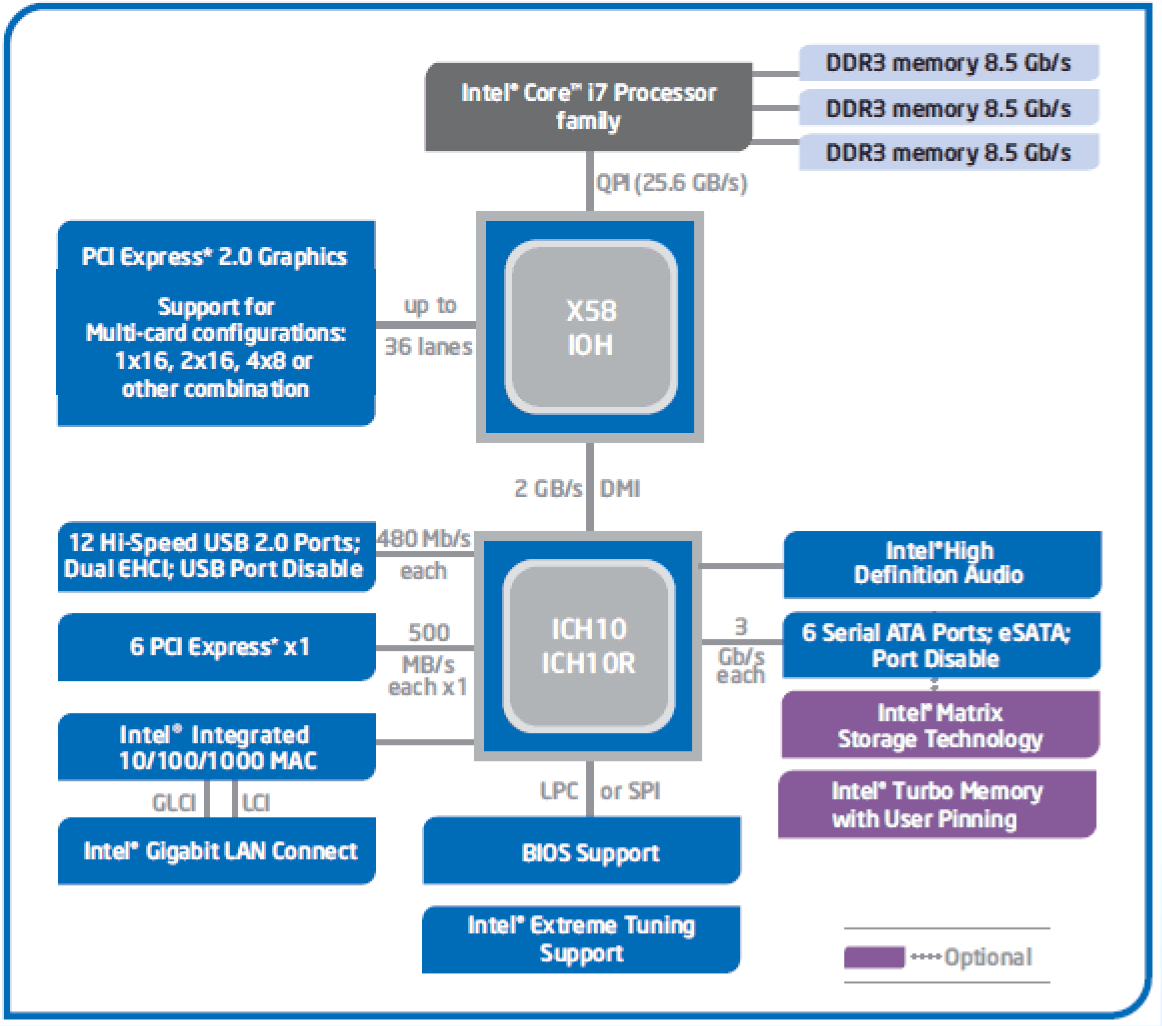
Test System



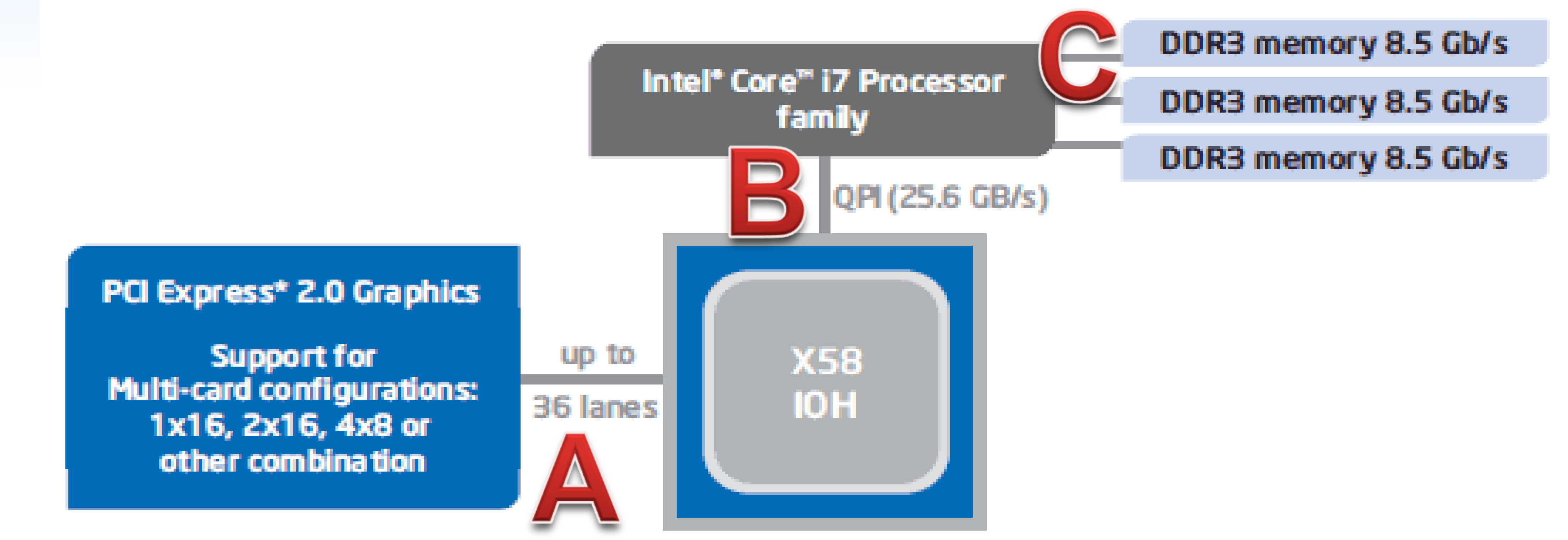
- CPU cores (Intel core i7): 4
- Accelerator Units (S1070): 2
- Total GPUs: 8
- CPU cores/GPU ratio: 0.5
- Theoretical peak PCIe bandwidth: 18 GB/s

- Host Memory: 24 GB
- GPU Memory: 32 GB

Intel X58 Chipset (Test System)



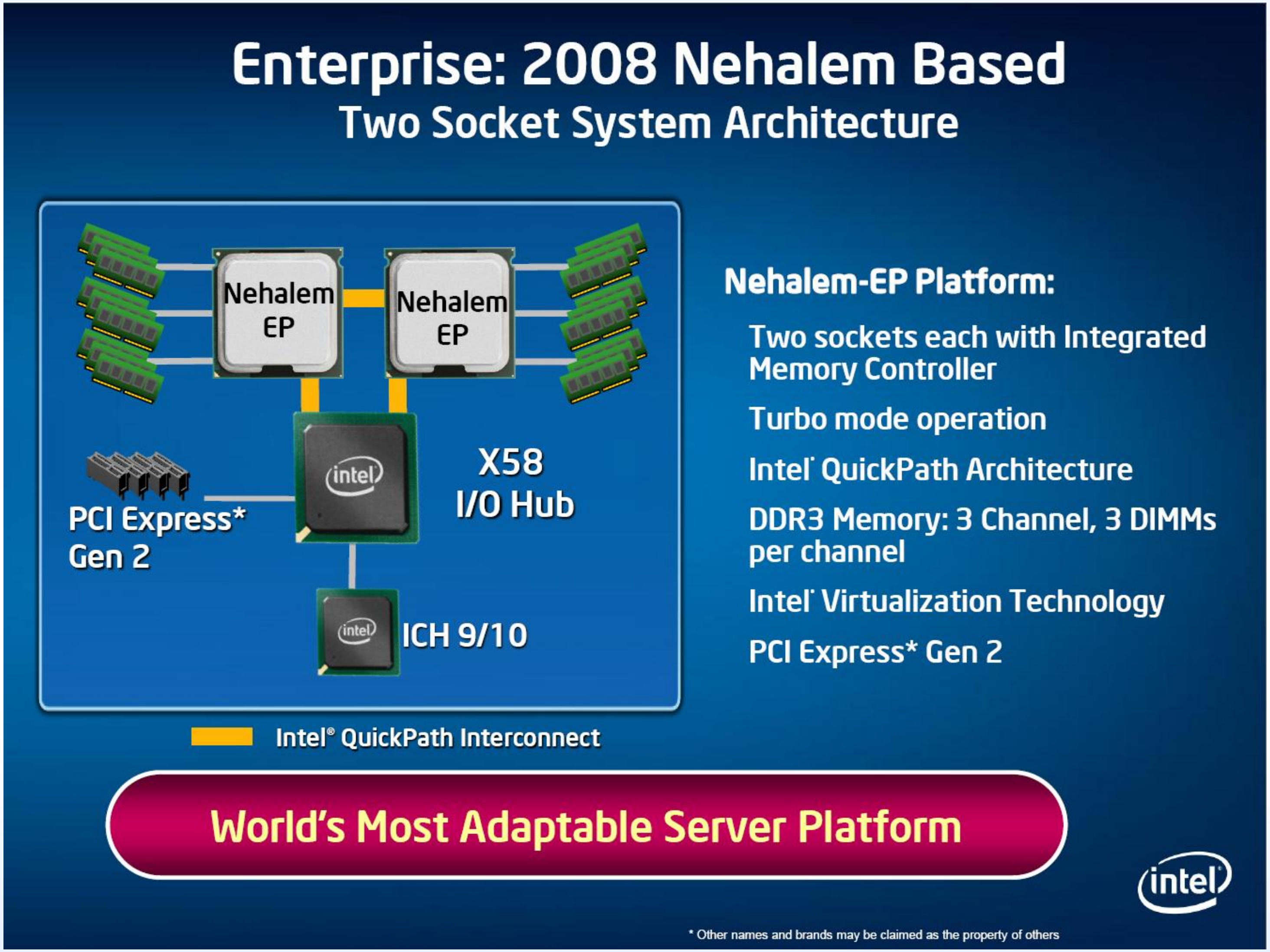
Designing a Balanced Compute Host



A=	36 x 500MB/sec/lane = 18GB/sec peak (1)	*(PCI-E Gen 2)*
B=	½ x 25.6GB/sec = 12.8 GB/sec peak	*(25.6 is a bi-directional bw)*
C=	192 bits * 800MHz = 17.8 GB/sec peak	*(assuming 3 banks populated)*

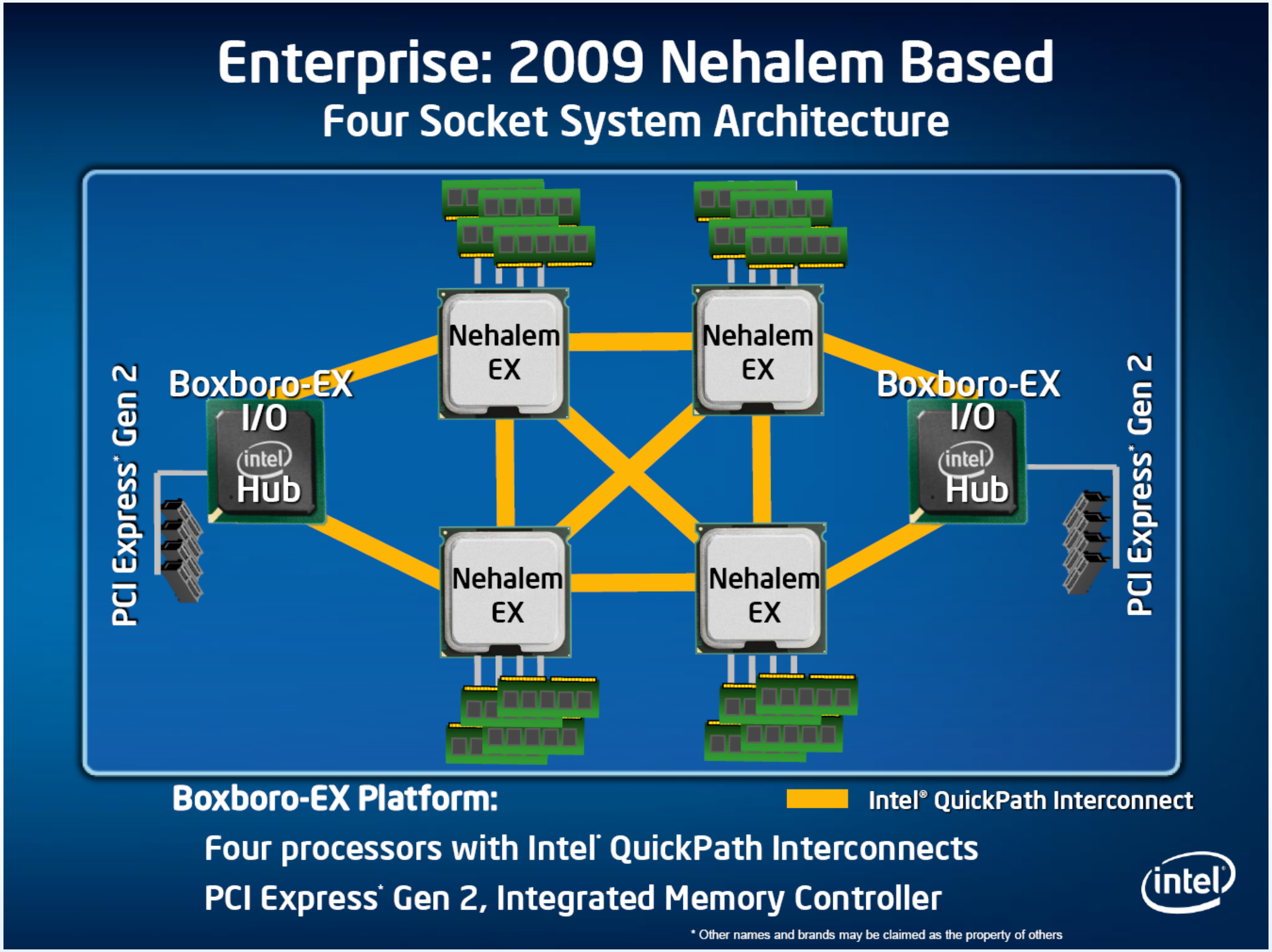
1. GPU memcopy traffic is serialized

Future Options



http://download.intel.com/pressroom/kits/events/idffall_2008/SSmith_briefing_roadmap.pdf

Future Options



http://download.intel.com/pressroom/kits/events/idffall_2008/SSmith_briefing_roadmap.pdf

Conclusions

- **GPU acceleration producing real results**
- **System tools becoming GPU aware, but still some gaps to fill**
- **Balanced system design depends on application requirements, but some basic guidelines apply**

Jeremy Enos jenos@ncsa.uiuc.edu

Thank you.