Enabling the Next Generation of Scalable Clusters

William Gropp
www.cs.illinois.edu/~wgropp

State of the World

- Clock rate ride over; power a constraint
- New architectures considered
  - GPGPU – even though hard to program effectively
- Quick look at the state of the world
  - Clouds
  - GPGPU Clusters
  - “Conventional” parallel supercomputer
    - Note that the last is “sort of” general purpose, while the others are more narrowly focused

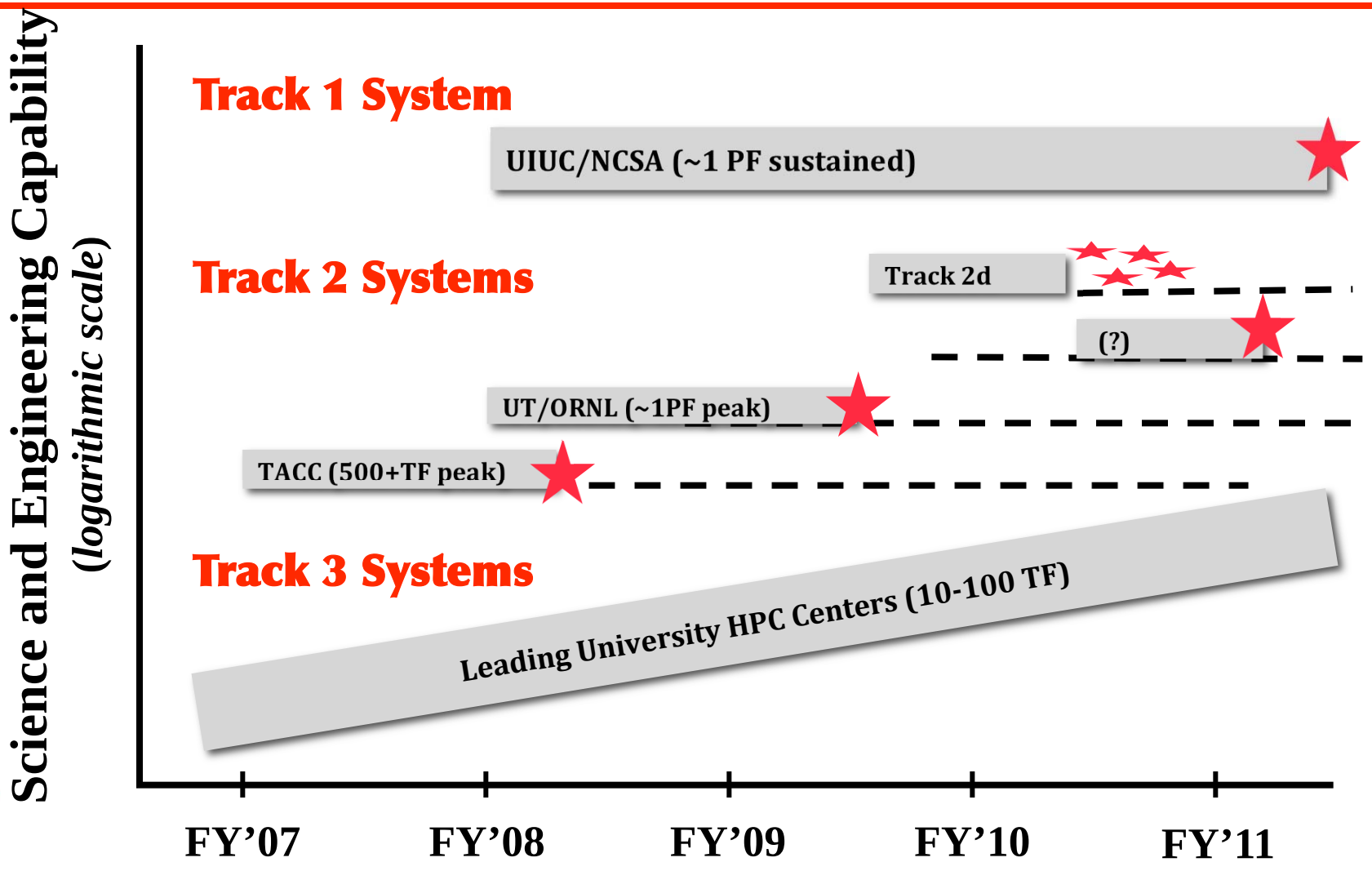
Clouds

- Clouds seem to be everywhere
  - Service oriented
  - Demand-driven pricing
  - Economy of scale
- For one view of what a cloud is and isn't, see "A View of Cloud Computing," Armbrust et al, Communications of the ACM Vol. 53 No. 4, Pages 50-58
- Includes 10 outstanding issues, including both performance and data transfer key to HPC use
  - None impossible, but
    - HPC often needs excellent networking
    - HPC often needs large scale I/O (and large is many TB)

GPGPU Equipped Clusters

- High compute density, good power efficiency
- But
  - ♦ Low level programming models
  - ♦ Complex performance models
- Still promising
  - ♦ RoadRunner at Los Alamos
  - ♦ Many GPU-enhanced clusters
  - ♦ May be in our future…

NSF's Strategy for High-end Computing



Diverse Large Scale Computational Science

Science areas	Multi-physics, Multi-scale	Dense linear algebra (DLA)	Sparse linear algebra (SLA)	Spectral Methods (FFT)s (SM-FFT)	N-Body Methods (N-Body)	Structured Grids (S-Grids)	Unstructured Grids (U-Grids)	Data Intensive
Nanoscience	X	X	X	X	X	X		
Chemistry	X	X	X	X	X			
Fusion	X	X	X			X	X	X
Climate	X		X	X		X	X	X
Combustion	X		X			X	X	X
Astrophysics	X	X	X	X	X	X	X	X
Biology	X	X					X	X
Nuclear		X	X		X			X
System Balance Implications	General Purpose balanced System	High Speed CPU, High Flop/s rate	High Performance Memory	High Interconnect Bisection bandwidth	High Performance Memory	High Speed CPU, High Flop/s rate	Irregular Data and Control Flow	High Storage and Network bandwidth

Focus on Sustained Performance

- **Blue Water's and NSF are focusing on *sustained* performance in a way few have been before.**
- *Sustained* is the computer's performance on a broad range of applications that scientists and engineers use every day.
  - Time to solution is the metric – not Ops/s
  - Tests include time to read data and write the results
- NSF's call emphasized sustained performance, demonstrated on a collection of application benchmarks (application + problem set)
  - Not just simplistic metrics (e.g. HP Linpack)
  - Applications include both Petascale applications (effectively use the full machine, solving scalability problems for both compute and I/O) and applications that use a fraction of the system
    - Metric is the time to solution
- Blue Waters project focus is on delivering sustained PetaFLOPS performance to all applications
  - Develop tools, techniques, samples, that exploit all parts of the system
  - Explore new tools, programming models, and libraries to help applications get the most from the system

Blue Waters Computing System

System Attribute	Typical Cluster (NCSA Abe)	Track 2 (TACC)	Blue Waters*
Vendor	Dell	Sun	IBM
Processor	Intel Xeon	AMD	Power 7
Peak Perf. (PF)	0.090	0.58	
Sustained Perf. (PF)	~0.005	~0.06	~1.0
Number of cores	9,600	62,976	>300,000
Amount of Memory (PB)	0.0144	0.12	>1.0
Amount of Disk Storage (PB)	0.1	1.73	>18
Amount of Archival Storage (PB)	5	2.5	~500
External Bandwidth (Gbps)	40	10	100-400

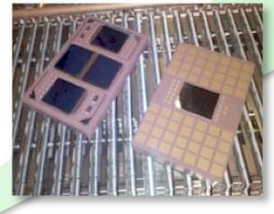
* *Reference petascale computing system (no accelerators).*

Building Blue Waters

Blue Waters will be the most powerful computer in the world for scientific research when it comes on line in Summer of 2011.

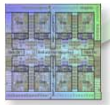
Blue Waters
~1 PF sustained
>300,000 cores
>1 PB of memory
>18 PB of disk storage
~500 PB of archival storage
>100 Gbps connectivity

Blue Waters Building Block
32 IH server nodes
32 TB memory
~256 TF (peak)
4 Storage systems
10 Tape drive connections



IH Server Node
8 MCM's (256 cores)
1 TB memory
~8 TF (peak)

Fully water cooled



Multi-chip Module
4 Power7 chips (SMP)
128 GB memory
512 GB/s memory bandwidth
~1 TF (peak)

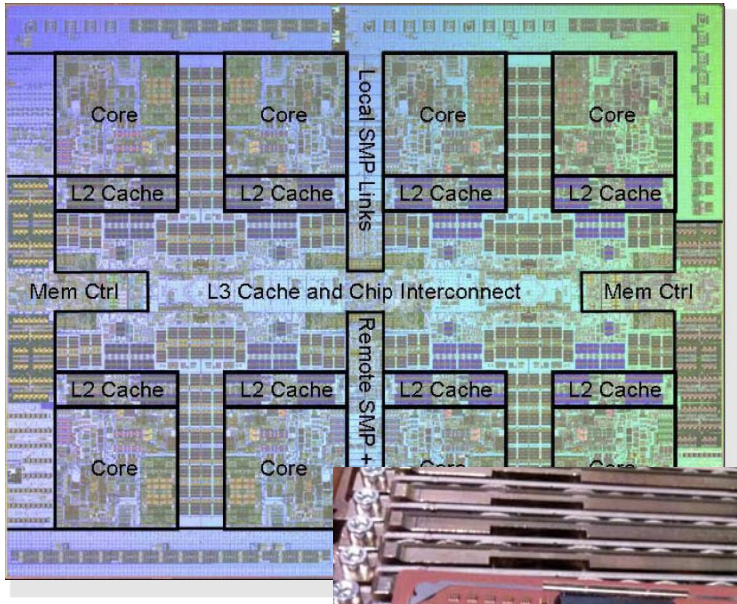
Router
1,128 GB/s bandwidth

Power7 Chip
8 cores, 32 threads
L1, L2, L3 cache (32 MB)
Up to 256 GF (peak)
45 nm technology

Blue Waters is built from components that can also be used to build systems with a wide range of capabilities—from deskside to beyond Blue Waters.

Power7 Chip: Computational Heart of Blue Waters

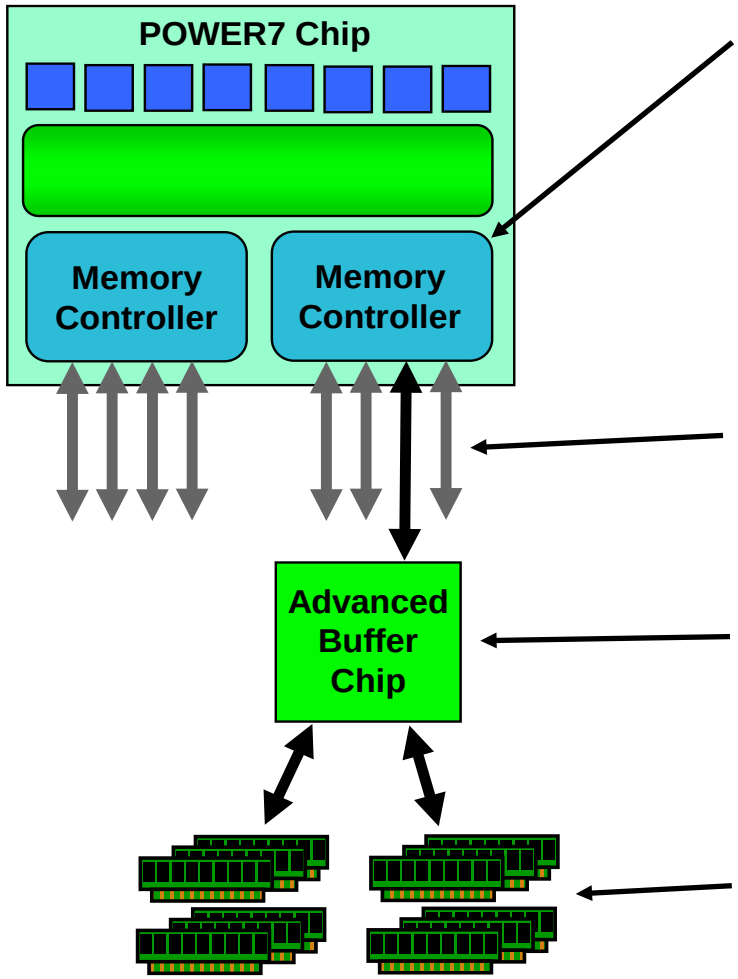
- Base Technology
  - 45 nm, 576 mm2
  - 1.2 B transistors
- Chip
  - 8 cores
  - 12 execution units/core
  - 1, 2, 4 way SMT/core
  - Up to 4 FMAs/cycle
  - Caches
    - 32 KB I, D-cache, 256 KB L2/core
    - 32 MB L3 (private/shared)
  - Dual DDR3 memory controllers
    - 128 GB/s peak memory bandwidth (1/2 byte/flop)
  - Clock range of 3.5 – 4 GHz



Power7 Chip

Quad-chip MCM

Memory Solutions



1) Dual Integrated DDR3 Controllers
- Massive 16KB scheduling window per POWER7 chip insures high channel and DIMM utilization
- Sparse access acceleration
- Advanced Energy Management
- Numerous RAS advances

2) Eight high speed 6.4 GHz channels
- New low power differential signaling
- Sustained 100+ GB/s per socket

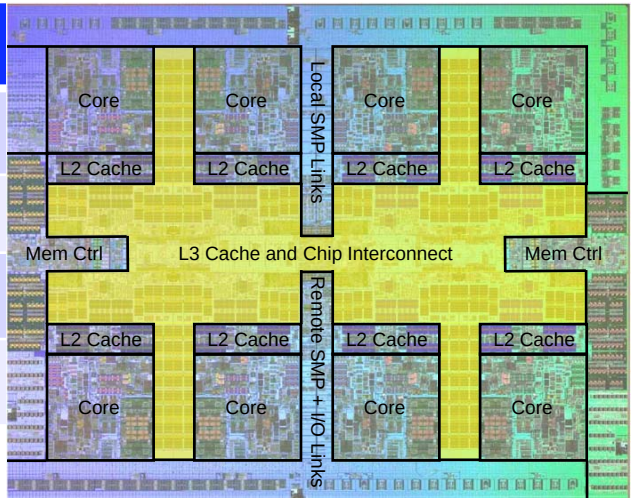
3) New DDR3 buffer chip architecture
- Larger capacity support (32 GB / core)
- Energy Management support
- RAS enablement

4) DDR3 DRAMs
- Supports 800, 1066, 1333, and 1600

Cache Structure Innovation

- Combines dense, low power attributes of eDRAM with the speed and bandwidth advantages of SRAM – all on the same chip
- Provides low latency L1 and L2 dedicated per core
  - ~3x lower latency than L3 Local region
  - Keeps a 256KB working set
  - Reduced L3 power requirements and improves throughput
- Provides large, shared L3
  - ~3x lower latency than memory
  - Automatically migrates per core private working set footprints (up to 4MB) to fast local region per code at ~5x lower latency than the full L3 cache
  - Automatically clones shared data to multiple per core private regions
  - Enables a subset of cores to utilize the entire, large shared L3 Cache when remaining cores are not using it.

Cache Level	Capacity	Array	Policy	Comment
L1 Data	32 KB	Fast SRAM	Store - thru	Local thread storage update
Private L2	256KB	Fast SRAM	Store-In	De-coupled global storage update
Fast L3 “Private”	Up to 4 MB	eDRAM	Partial Victim	Reduced power footprint (up to 4 MB)
Shared L3	32MB	eDRAM	Adaptive	Large 32MB shared footprint

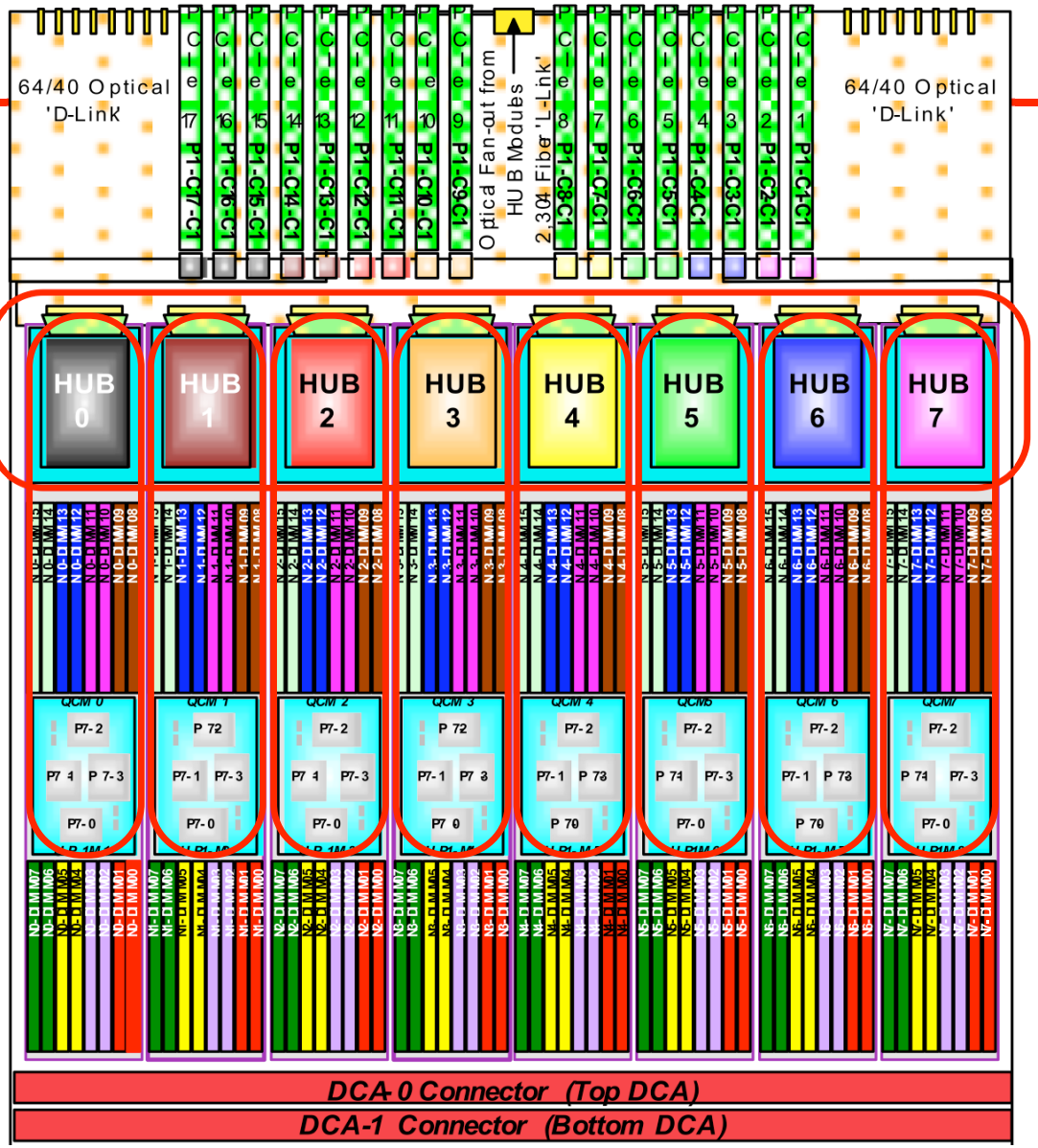


1.1 TB/s HUB

- 192 GB/s Host Connection
- 336 GB/s to 7 other local nodes in the same drawer
- 240 GB/s to local-remote nodes in the same supernode (4 drawers)
- 320 GB/s to remote nodes
- 40 GB/s to general purpose I/O

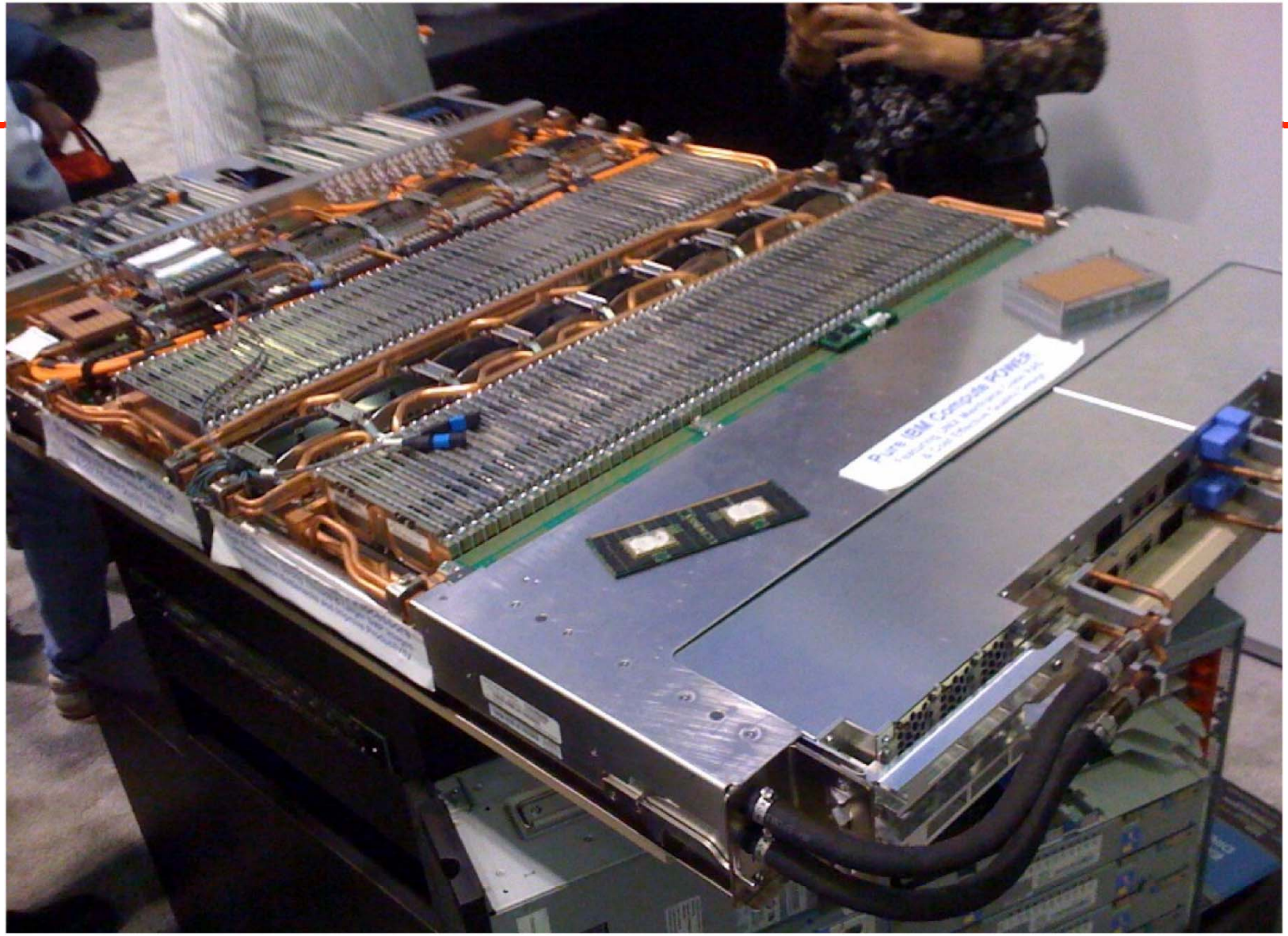
ONE DRAWER

8 MCMs, 32 chips, 256 cores



First Level Interconnect

- L-Local
- HUB to HUB Copper Wiring
- 256 Cores

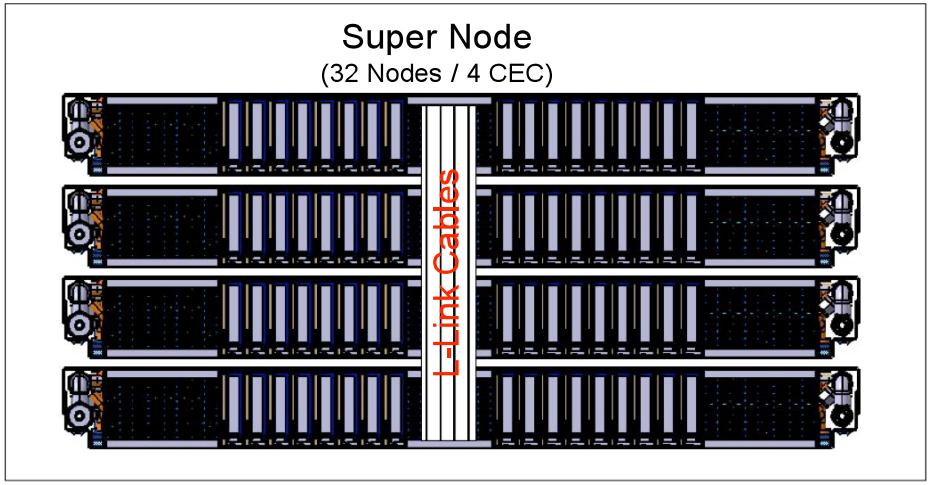


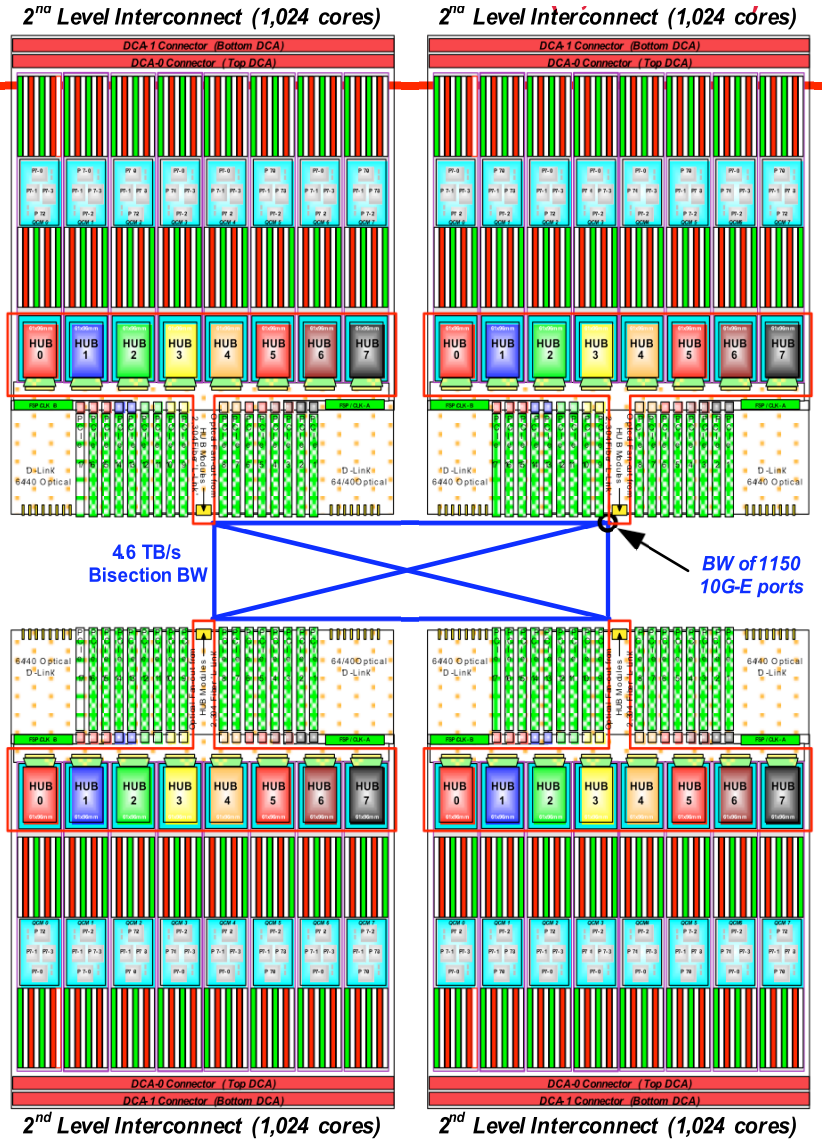
ONE SUPERNODE

4 drawers, 32 MCMs, 128 chips, 1024 cores

<u>Second Level Interconnect</u>

- Optical 'L-Remote' Links from HUB
- Construct Super Node (4 CECs)
- 1,024 Cores
- Super Node





Rack

- 990.6w x 1828.8d x 2108.2
- 39”w x 72”d x 83”h
- ~2948kg (~6500lbs)

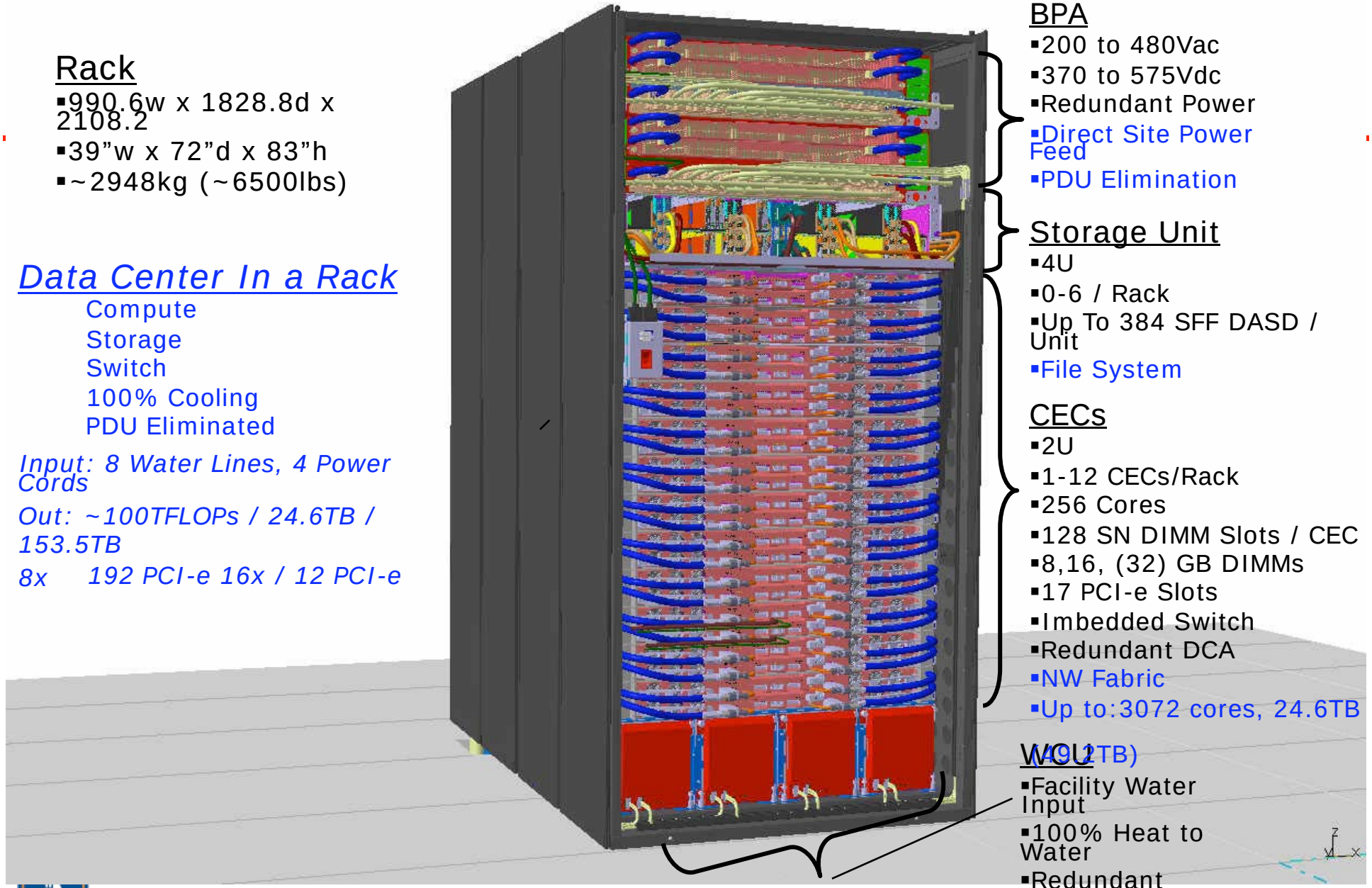
Data Center In a Rack

Compute
Storage
Switch
100% Cooling
PDU Eliminated

Input: 8 Water Lines, 4 Power Cords

Out: ~100TFLOPs / 24.6TB / 153.5TB

8x 192 PCI-e 16x / 12 PCI-e



BPA

- 200 to 480Vac
- 370 to 575Vdc
- Redundant Power
- Direct Site Power Feed
- PDU Elimination

Storage Unit

- 4U
- 0-6 / Rack
- Up To 384 SFF DASD / Unit
- File System

CECs

- 2U
- 1-12 CECs/Rack
- 256 Cores
- 128 SN DIMM Slots / CEC
- 8,16, (32) GB DIMMs
- 17 PCI-e Slots
- Imbedded Switch
- Redundant DCA
- NW Fabric
- Up to:3072 cores, 24.6TB (49.2TB)

WCU

- Facility Water Input
- 100% Heat to Water
- Redundant Cooling
- CRAH Eliminated

National Petascale Computing Facility at a Glance

Partners

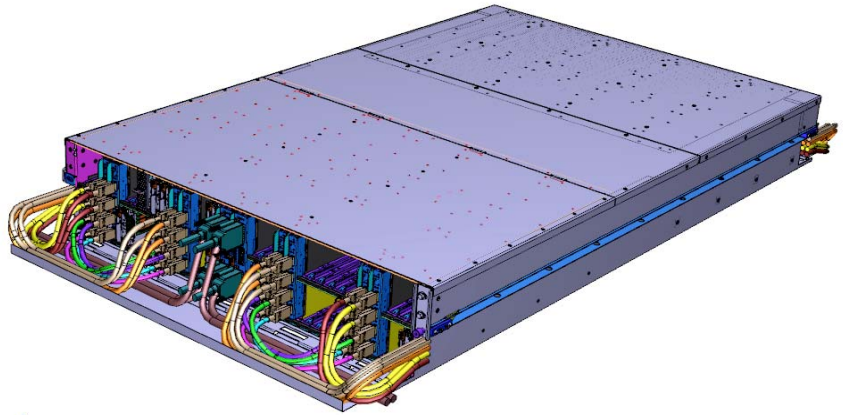
EYP MCF/
Gensler
IBM
Yahoo!

www.ncsa.illinois.edu/BlueWaters

- 88,000 GSF over two stories—45' tall
  - 30,000+ GSF of raised floor
  - 20,000+ unobstructed net for computers
  - 6' clearance of raised floor
- 24 MW initial power feeds + backup
  - Three 8 MW feeds + One 8 MW for backup
  - 13,800 volt power to the each
- 5,400 Tons of cooling
  - Full water side economization for 50%+ of the year
  - Automatic Mixing of mechanical and ambient chilled water for optimal efficiency
  - Adjacent to (new) 6.5M gallon thermal storage tank
- 480 Volt distribution to computers
- Energy Efficiency
  - PUE - ~1.02 to <1.2 (projected)
  - USGBC LEED ~~Silver~~ Gold (Platinum?) classification target

Data Capability

- >18PB of disk
- Peak IO performance in excess of 1.5TB/s.
- Note over 1PB of memory
  - ♦ Can load 100 TB database in a few minutes
  - ♦ Entire DB fits in memory (for even a more modest sized system)
  - ♦ Excellent system for data analysis, not just FLOPS



Status

- Building is ready (NPCF)
- POWER7 systems becoming available
  - Currently testing on simulators as well as hardware
- Programming models include UPC as well as MPI; all are interoperable through local data
- NSF providing allocations through PRAC process
  - Applications are already tuning for BW
  - Third round closed last March
  - Next round closes March 17, 2011
- Blue Waters will begin running applications in 2011

Where can we get with a Homogeneous Cluster?

- What's commodity about Blue Waters?
  - Power7, SMP nodes
  - I/O (but rare to have this much capability in an HPC system)
- What's not commodity?
  - Network (though it could/should be)
- What are the limits?
  - Power consumption in 10's of MW
    - A TGV is about 8MW
    - Water cooling (both to remove heat and do it more efficiently than air cooling)
  - Exascale will need 100-1000x power efficiency; 100-1000x space efficiency
  - Just how bad is this?

Exascale Challenges

- Exascale will be hard (see the DARPA Report [Kogge])
  - ♦ Conventional designs plateau at 100 PF (peak
    - all energy is used to move data
  - ♦ Aggressive design is at 70 MW and is very hard to use
    - 600M instruction/cycle - Concurrency
    - 0.0036 Byte moved/flop – All operations local
    - No ECC, no redundancy – Must detect/fix errors
    - No cache memory – Manual management of memory
    - HW failure every 35 minutes – Eeek!
- Waiting doesn't help
  - ♦ At the limits of CMOS technology

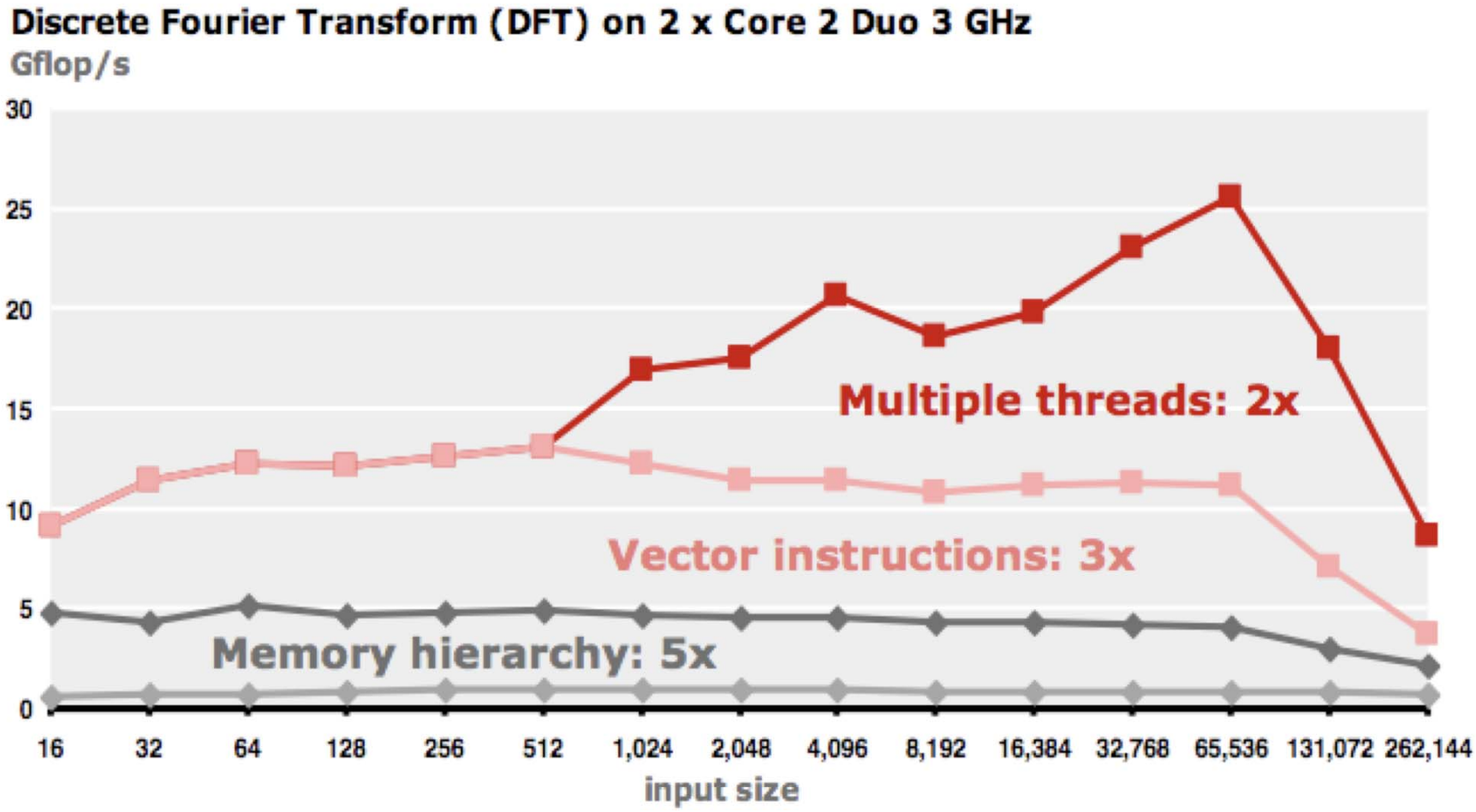
What can we do?

- Better use of our existing systems
  - ♦ Blue Waters will provide a sustained PF, but that requires ~10PF peak
- Improve node performance
  - ♦ Make the compiler better
  - ♦ Give better code to the compiler
  - ♦ Get realistic with algorithms/data structures
- Improve parallel performance/scalability
- Improve productivity of applications
- Improve algorithms

Make the Compiler Better

- It remains the case that most compilers cannot compete with hand-tuned or autotuned code on simple code
  - Just look at dense matrix-matrix multiplication or matrix transpose
  - Try it yourself!
    - Matrix multiply on my laptop:
    - N=100 (in cache): 1818 MF (1.1ms)
    - N=1000 (not): 335 MF (6s)

Compilers versus Libraries in DFT



Source: Markus Püschel. Spring 2008.

How Do We Change This?

- Test compiler against “equivalent” code (e.g., best hand-tuned or autotuned code that performs the same computation, under some interpretation or “same”)
  - In a perfect world, the compiler would provide the same, excellent performance for all equivalent versions
- As part of the Blue Waters project, Padua, Garzaran, Maleki are developing a test suite that evaluates how the compiler does with such equivalent code
  - Working with vendors to improve the compiler
  - Identify necessary transformations
  - Identify opportunities for better interaction with the programmer to facilitate manual intervention.
  - Main focus has been on code generation for vector extensions
  - Result is a compiler whose realized performance is less sensitive to different expression of code and therefore closer to that of the best hand-tuned code.
  - Just by improving automatic vectorization, loop speedups of more than 5 have been observed on the Power 7.
- But this is a long-term project
  - What can we do in the meantime?

Give “Better” Code to the Compiler

- Augmenting current programming models and languages to exploit advanced techniques for performance optimization (i.e., *autotuning*)
- Not a new idea, and some tools already do this.
- But how can these approaches become part of the mainstream development?

How Can Autotuning Tools Fit Into Application Development?

- In the short run, just need effective mechanisms to replace user code with tuned code
  - ♦ Manual extraction of code, specification of specific collections of code transformations
- But this produces at least two versions of the code (tuned (for a particular architecture and problem choice) and untuned). And there are other issues.
- What does an application *want* (what is the Dream)?

Application Needs Include

- Code must be portable
- Code must be persistent
- Code must permit (and encourage) experimentation
- Code must be maintainable
- Code must be correct
- Code must be faster

Implications of These Requirements

- Portable - augment existing language. Either use pragmas/ comments or extremely portable precompiler
  - Best if the tool that performs all of these steps looks like just like the compiler, for integration with build process
- Persistent
  - Keep original and transformed code around
- Maintainable
  - Let use work with original code *and* ensure changes automatically update tuned code
- Correct
  - Do whatever the app developer needs to believe that the tuned code is correct
    - In the end, this will require running some comparison tests
- Faster
  - Must be able to interchange tuning tools - pick the best tool for *each* part of the code
  - No captive interfaces
  - Extensibility - a clean way to add new tools, transformations, properties, …

Application-Relevant Abstractions

- Language for interfacing with autotuning must convey concepts that are meaningful to the application programmer
- Wrong: unroll by 5
  - Though could be ok for performance expert, and some compilers already provide pragmas for specific transformations
- Right (maybe): Performance precious, typical loop count between 100 and 10000, even, not power of 2
- We need work at developing higher-level, performance-oriented languages or language extensions

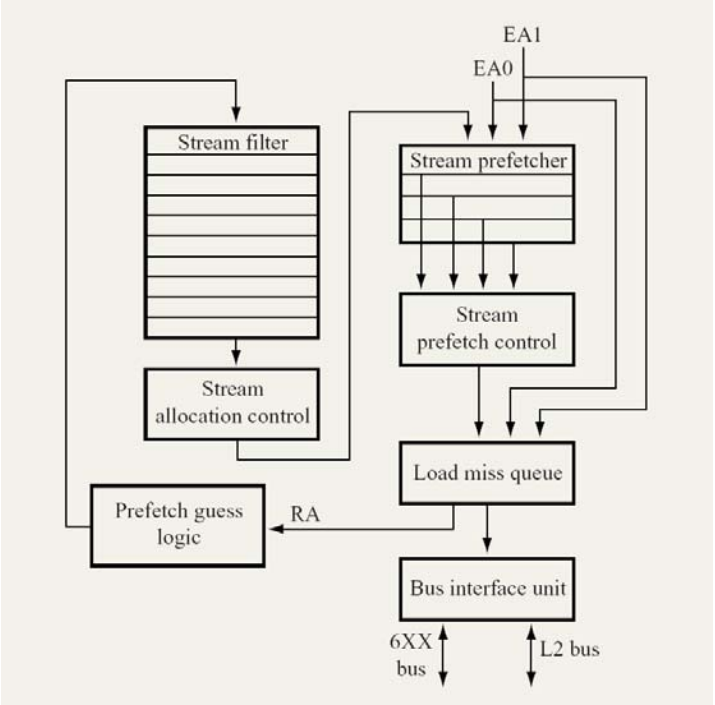
Better Algorithms and Data Structures

- Autotuning only offers the best performance with the given data structure and algorithm
  - That's a big constraint
- Processors include hardware to address performance challenges
  - "Vector" function units
  - Memory latency hiding/prefetch
  - Atomic update features for shared memory
  - Etc.

Table 2: Data Stream and Cache information on IBM power chips

Prefetch Engine on IBM Power Microprocessors

- Beginning with the Power 3 chip, IBM provided a hardware component called a prefetch engine to monitor cache misses, guess the data pattern ("data stream") and prefetch data in anticipation of their use.
- Power 4, 5 and 6 processors enhanced this functionality.



The Prefetch Engine on Power3

	Data Streams	L2 Cache (MB)	L3 Cache(MB)
Power 4	8	~1.5	32
Power 5	8	1.875	36
Power 6	16	4	32

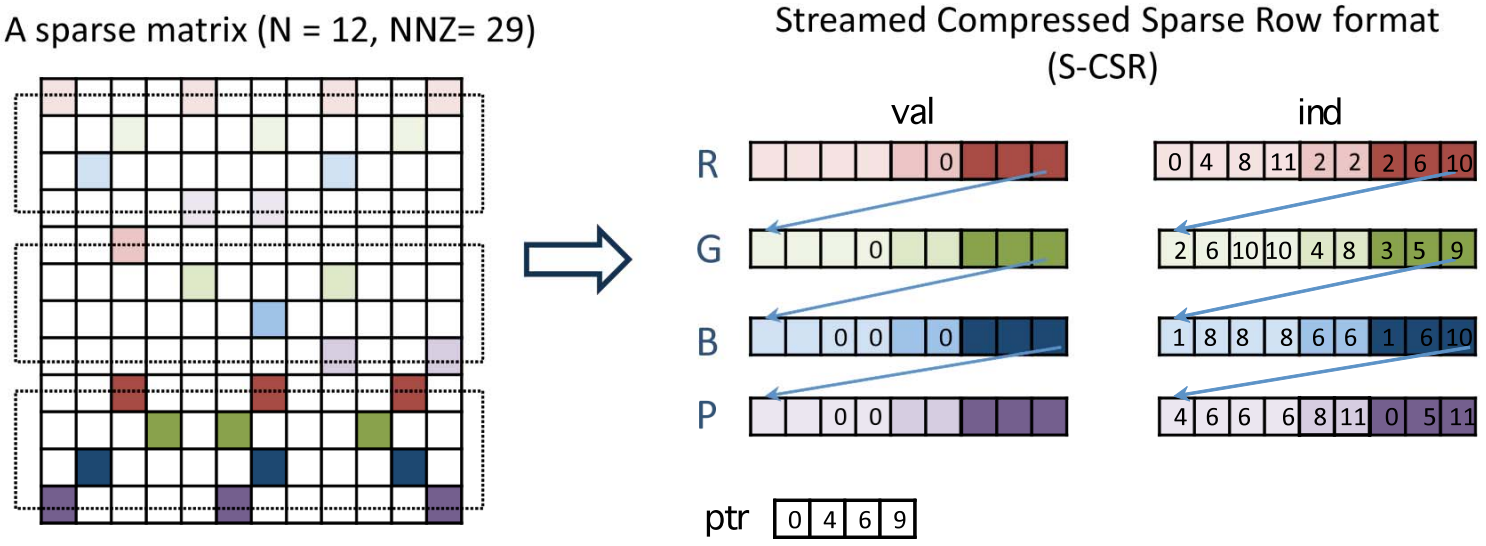
Data Stream and Cache Information

Inefficiency of CSR and BCSR formats

- The traditional CSR and Blocked CSR are hard to reorganize for data streams (esp > 2 streams) to enable prefetch, since the number of non-zero elements or blocks for every row may be different.
- Blocked CSR (BCSR) format can improve performance for some sparse matrices that are locally dense, even if a few zeros are added to the matrix.
  - If the matrix is too sparse (or structure requires too many added zeros), BCSR can hurt performance

Streamed Compressed Sparse Row (S-CSR) format

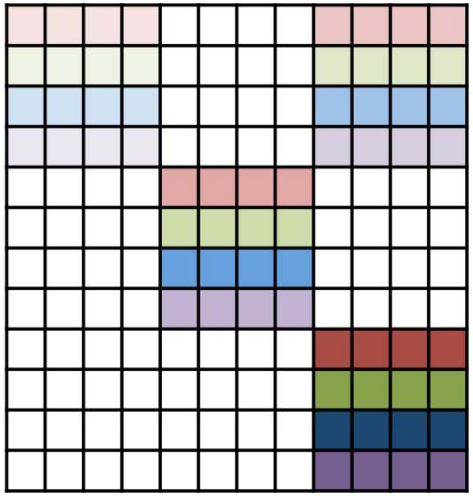
- S-CSR format partitions the sparse matrix into blocks along rows with size of bs. Zeros are added in to keep the number of elements the same in each row of a block. The column indices for ZEROs in each row are set to the index of the last non-zero element in the row. The first rows of all blocks are stored first, then second, third … and bs-th rows.
- For the sample matrix in the following Figure, NNZ = 29. Using a block size of bs = 4, it generates four equal length streams R, G, B and P. This new design only adds 7 zeros every 4 rows.



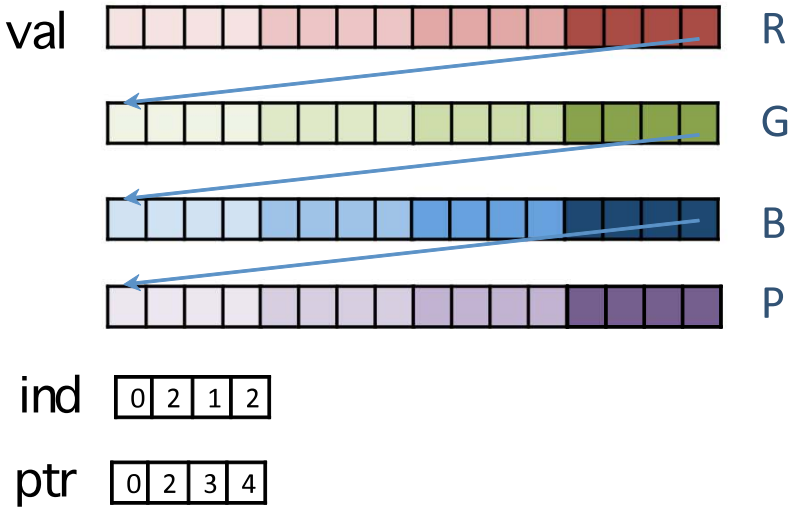
Streamed Blocked Compressed Sparse Row (S-BCSR) format

- When the matrix is locally dense and can be blocked efficiently with a few ZEROs added in, we can restore the blocked matrix using the similar idea as S-CSR format. The first rows of all blocks are stored first, then second, third … and last rows. Using 4x4 block for example, it will generate R, G, B and P four equal length streams. We call this the Streamed Blocked Compressed Row storage format (S-BCSR).

A sparse matrix with 4X4 blocks

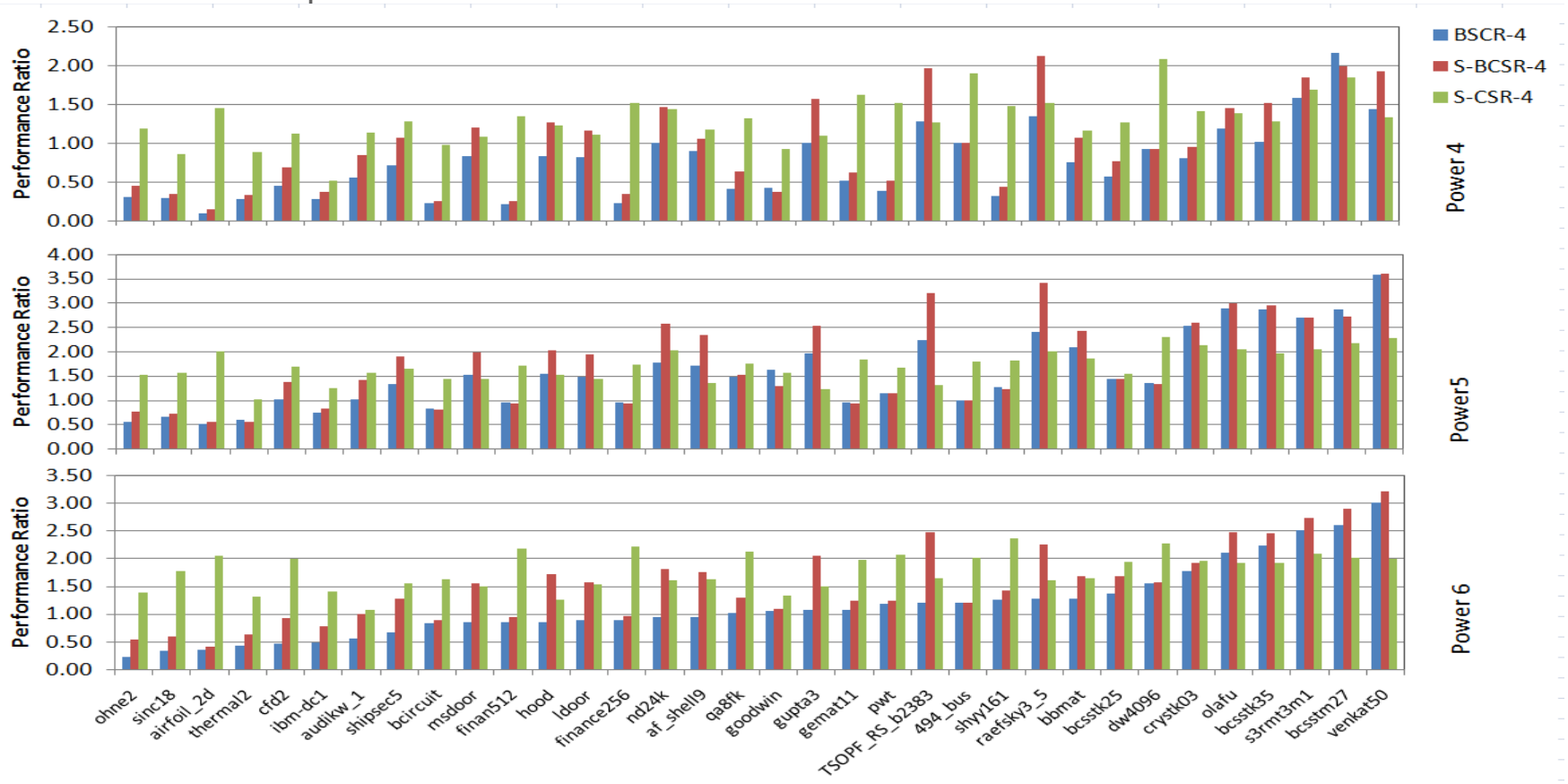


Streamed Blocked Compressed Sparse Row format (S-BCSR)



Performance Ratio Compared to CSR Format

- S-CSR format is better than CSR format for all (on Power 5 and 6) or Most (on Power 4) matrices
- S-BCSR format is better than BCSR format for all (on Power 6) or Most (on Power 4 and 5) matrices
- Blocked format performance from ½ to 3x CSR.



What Does This Mean For You?

- It is time to rethink data structures and algorithms to match the realities of memory architecture
  - We have results for x86 where the benefit is smaller but still significant
  - Better match of algorithms to prefetch hardware is necessary to overcome memory performance barriers
- Similar issues come up with heterogeneous processing elements (someone needs to *design* for memory motion and concurrent and nonblocking data motion)

Performance on a Node

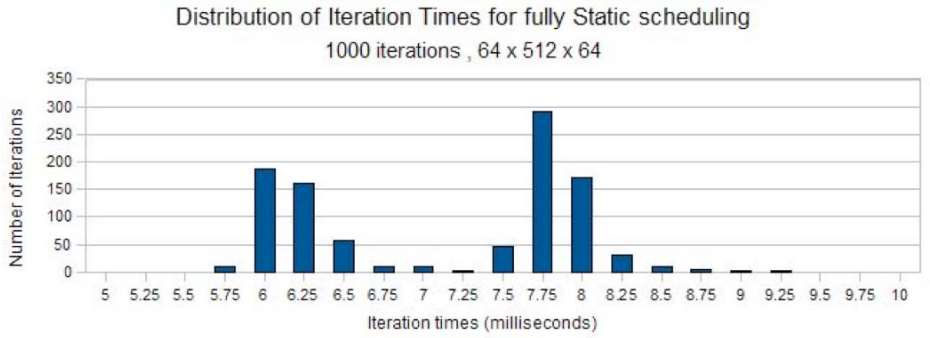
- Nodes are SMPs
  - You have this problem on anything (even laptops)
- Tuning issues include the usual
  - Getting good performance out of the compiler (often means adapting to the memory hierarchy)
- New (SMP) issues include
  - Sharing the SMP with other processes
  - Sharing the memory system

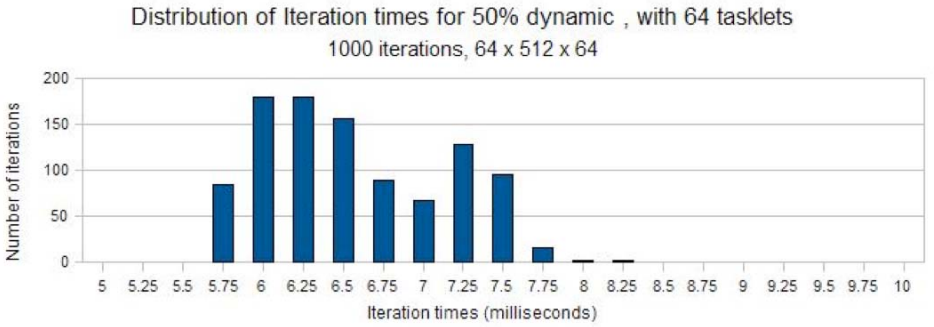
New (?) Wrinkle – Avoiding Jitter

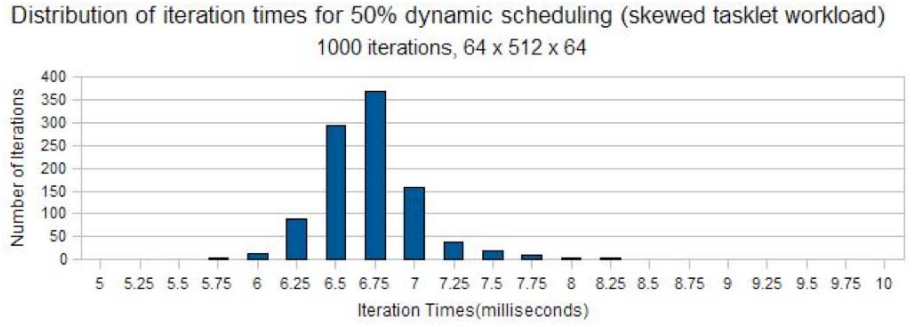
- Jitter here means the variation in time measured when running identical computations
  - Caused by other computations, e.g., an OS interrupt to handle a network event or runtime library servicing a communication or I/O request
- This problem is in some ways *less* serious on HPC platform, as the OS and runtime services are tuned to minimize impact
  - However, cannot be eliminated entirely

Sharing an SMP

- Having many cores available makes everyone think that they can use them to solve other problems ("no one would use all of them all of the time")
- However, compute-bound scientific calculations are often *written* as if all compute resources are owned by the application
- Such *static* scheduling leads to performance loss
- Pure dynamic scheduling adds overhead, but is better
- Careful mixed strategies are even better
- Thanks to Vivek Kale







Expressing Parallelism

- Programming Model Libraries
  - OpenMP; threads
  - MPI (MPI-1, MPI-2, MPI-3)
  - (Open)SHMEM, GA
- Parallel Programming Languages
  - UPC, CAF in Fortran 2008
  - HPCS (Chapel, X10, Fortress)
- Hybrid Models
  - MPI + Threads
- Libraries/Frameworks
  - Math libraries
  - I/O libraries
  - Parallel programming frameworks (e.g., Charm++)

The PGAS Languages

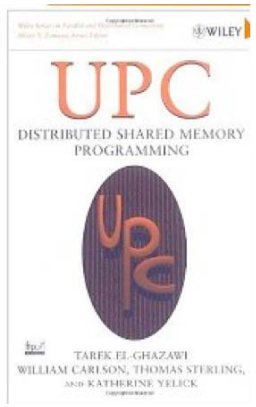
- PGAS (Partitioned Global Address Space) languages attempt to combine the convenience of the global view of data with awareness of data locality, for performance
  - Co-Array Fortran (CAF), an extension to Fortran 90
  - UPC (Unified Parallel C), an extension to C

Co-Array Fortran (CAF)

- SPMD – Single program, multiple data
  - Replicated to a number of images
  - Images have indices 1,2, …
  - Number of images fixed during execution
  - Each image has its own set of local variables
  - Images execute asynchronously except when explicitly synchronized
- Variables declared as co-arrays are accessible by another image through a set of array subscripts, delimited by [ ] and mapped to image indices by the usual rule
- Multiple versions of CAF
  - Classic CAF
  - CAF in Fortran 2008
    - Like Classic, but: No collectives; no teams.

UPC

- UPC is an extension of C (not C++) with shared and local addresses
- Shared keyword in type declarations
- UPC defines parallelism in terms of “threads” (may be implemented as OS threads)
- Extensions include collectives and nonblocking transfers
- Several implementations exist including xlupc from IBM



Newer Languages

- HPCS Languages
  - ♦ Chapel, X10, Fortress
  - ♦ Retains locality (but in a general form)
  - ♦ Adds concurrency creation
  - ♦ More general distributed data structures
  - ♦ More general synchronization methods
- Research implementations – not ready for applications

Hybrid Programming Models

- No one programming model is best for all parts of most applications
- Combining programming models provides a powerful set of tools
  - Can give very good results
  - But relies on a clean and efficient interface between programming models – this is often missing
- On Blue Waters, MPI, UPC, CAF, and others will be interoperable
  - Can build library routines/components in most appropriate model
  - Link application together
  - Work still needs to be done to understand how best to coordinate the models
    - On BW, all models make use of a single lower level, simplifying that coordination. However, threads and internode support not unified

Getting Past MPI

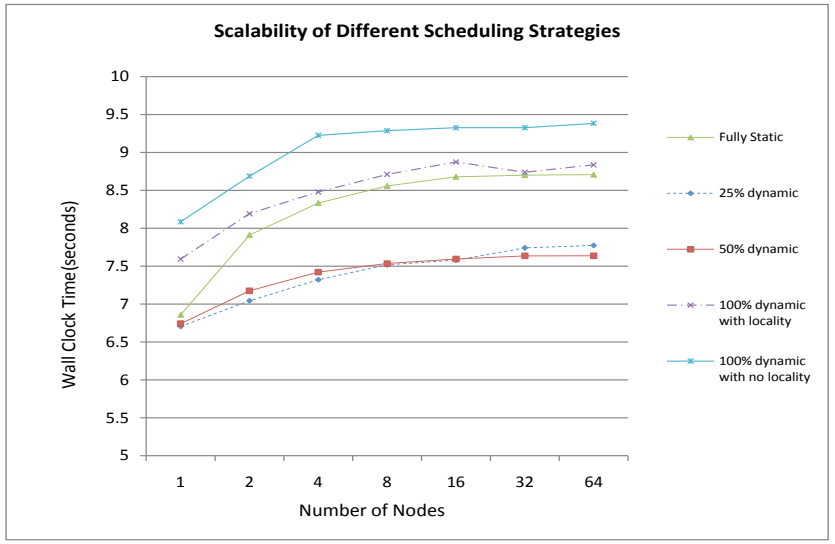
- Incremental – various hybrid models
  - Single thread language (C/C++Fortran)
  - MPI – general Data structures; low level locality control
  - Thread/OpenMP – low-level shared memory; concurrency creation
  - PGAS – support for distributed data structures; compiled communication
- Revolution
  - Is it C/Fortran/C++ with extensions (e.g., next generation PGAS)?
  - Must offer radical new capabilities
    - Concurrency creation
    - Latency hiding
    - Data motion minimization
  - But without sacrificing generality

Scalability – A Matter of Degree

- Concurrency
  - ♦ Need > 300k concurrent threads
  - ♦ Need > 10K more loosely coupled tasks
    - Typical latencies: 1-10ns to Cache; 100-1000ns to Memory; 1000-10000ns to remote memory
- Latency tolerance and communication overlap
  - ♦ Systems are hundreds to thousands of clock cycles across
- Load Balance
  - ♦ Any imbalance, from whatever cause, can cause everyone to wait
  - ♦ Synchronous (barrier) algorithms likely to scale poorly

Example of Load (Im)balance and Scaling

- Simple regular grid sweep
- Work per node should be the same (each node is a 16-way SMP); weak scaling
- All 16 cores used (typical for real life)
- Local imbalances within node create scalability problem
- Note that load imbalance will appear to slower MPI communication



What's Different at Petascale

- Performance Focus
  - Only a little – basically, the resource is expensive, so a premium placed on making good use of resource
  - Quite a bit – node is more complex, has more features that must be exploited
- Scalability
  - Solutions that work at 100-1000 way often inefficient at 100,000-way
  - Some algorithms scale well
    - Explicit time marching in 3D
  - Some don't
    - Direct implicit methods
  - Some scale well for a while
    - FFTs (communication volume in Alltoall)
  - Load balance, latency are critical issues
- Fault Tolerance becoming important
  - Now: reduce time spent in checkpoints
  - Soon: Lightweight recovery from transient errors

A Cluster Agenda

- Better use of existing resources
  - Performance-oriented programming
  - Dynamic management of resources at all levels
  - Embrace hybrid programming models (you have already)
- Focus on results
  - Network bandwidth (and latency)
  - I/O capability
- Prepare for the future
  - Fault tolerance
  - Latency Tolerant Algorithms
  - Data-driven systems