CRAY
THE SUPERCOMPUTER COMPANY

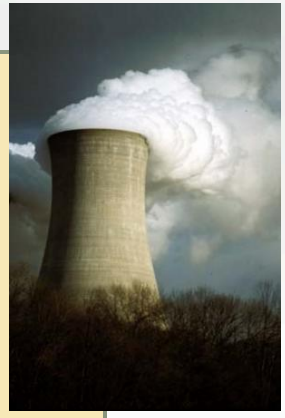
Hybrid Multi-core Programming for Exascale Computing

John Levesque
Cray CTO Office
Director of Cray's Supercomputing Center of Excellence

Titan Workshop 2012
January 23rd

Key Challenges to Get to an Exascale

Power



- Traditional voltage scaling is over
- Power now a major design constraint
- Cost of ownership
- Driving significant changes in architecture

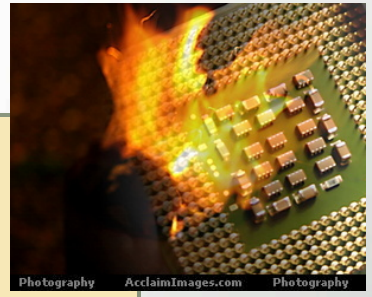
Concurrency

- A billion operations per clock
- Billions of refs in flight at all times
- Will require *huge* problems
- Need to exploit *all* available parallelism

Programming Difficulty

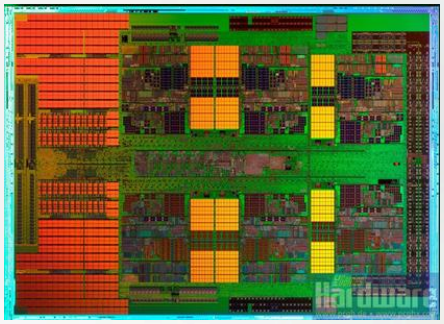
- Concurrency and new micro-architectures will significantly complicate software
- Need to hide this complexity from the users

Resiliency



- Many more components
- Components getting less reliable
- Checkpoint bandwidth not scaling

Improving Processor Efficiency

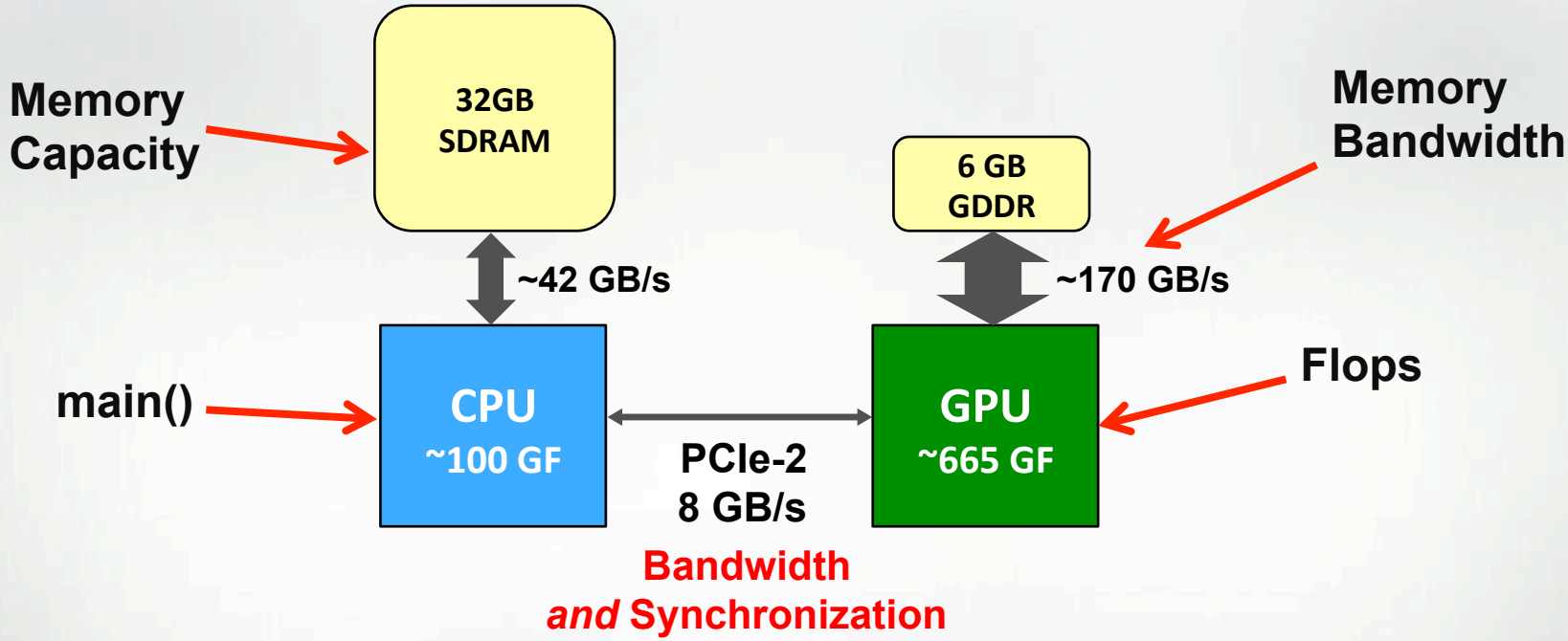


- Multi-core was a good first response to power issues
  - Performance through parallelism
  - Modest clock rate
  - Exploit on-chip locality
- However, conventional processor architectures are optimized for single thread performance rather than energy efficiency
  - Fast clock rate with latency(performance)-optimized memory structures
  - Wide superscalar instruction issue with dynamic conflict detection
  - Heavy use of speculative execution and replay traps
  - Large structures supporting various types of predictions
  - Relatively little energy spent on actual ALU operations
- Could be much more energy efficient with multiple simple processors, exploiting vector/SIMD parallelism and a slower clock rate
- But serial thread performance is really important (Amdahl's Law):
  - If you get great parallel speedup, but hurt serial performance, then you end up with a niche processor (less generally applicable, harder to program)

Conclusion: Heterogeneous Computing

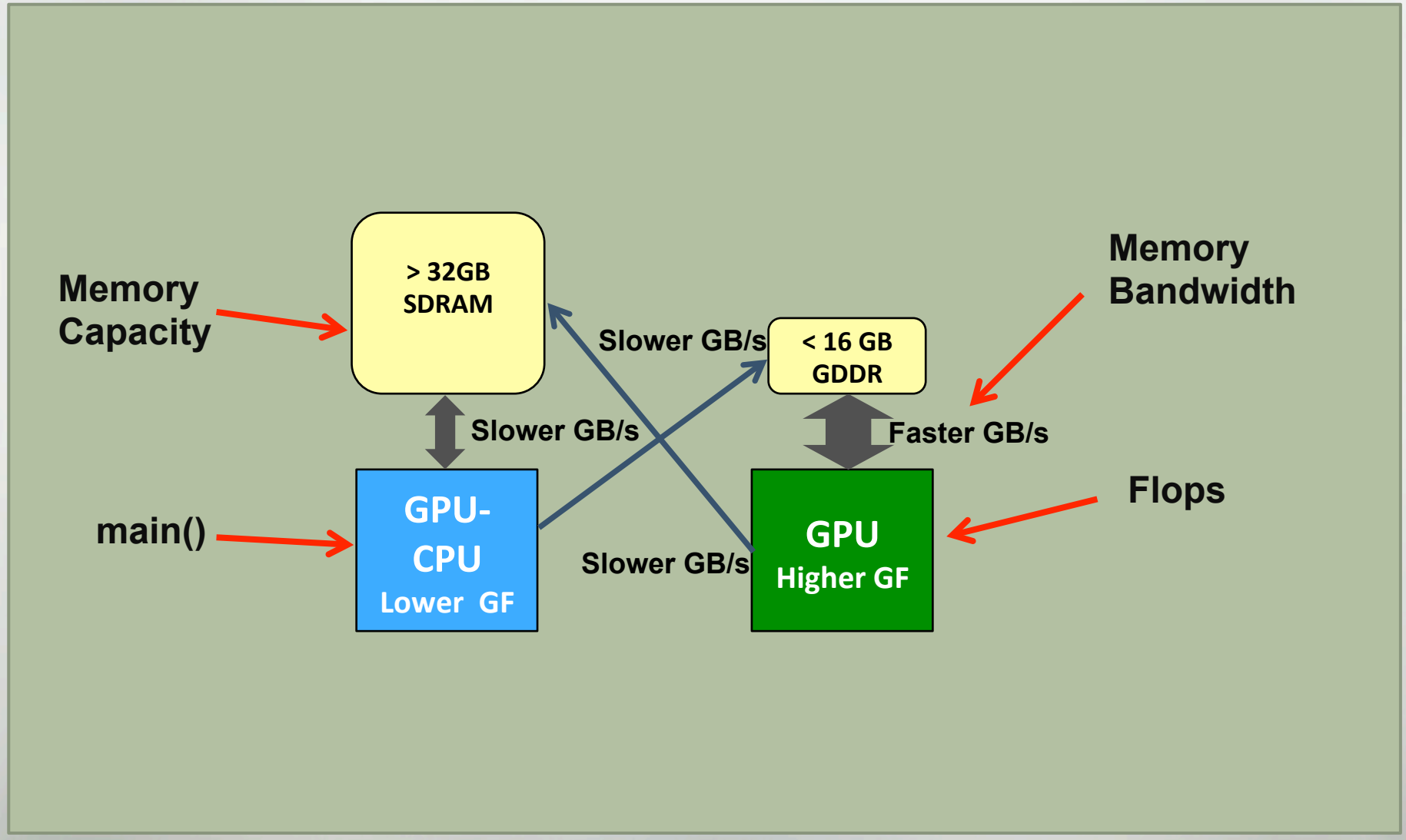
- To achieve scale and sustained performance per {$,watt}, must adopt:
  - …a *heterogeneous* node architecture
    - fast cores for serial code
    - many power-efficient cores for parallel code
  - …a deep, explicitly managed memory hierarchy
    - to better exploit locality, improve predictability, and reduce overhead
  - …a microarchitecture to exploit parallelism at all levels of a code
    - distributed memory, shared memory, vector/SIMD, multithreaded
    - (related to the "concurrency" challenge—leave no parallelism untapped)
- Sounds a lot like GPU accelerators…
- NVIDIA Fermi™ has made GPUs feasible for HPC
  - Robust error protection and strong DP FP, plus programming enhancements
- Expect GPUs to make continued and significant inroads into HPC
  - Compelling technical reasons
  - High volume market
  - It looks like they can credibly support both masters (graphics and compute)
- *Two issues w/ GPU acceleration: STRUCTURAL and PROGRAMMING*

Structural Issues with Accelerated Computing



- This is a short-lived situation
  - NVIDIA Denver and AMD Fusion
- Try to keep kernel data structures resident in GPU memory
  - Avoids copying b/w CPU and GPU; work on GPU-network communication
- May limit breadth of applicability over next 2-3 years

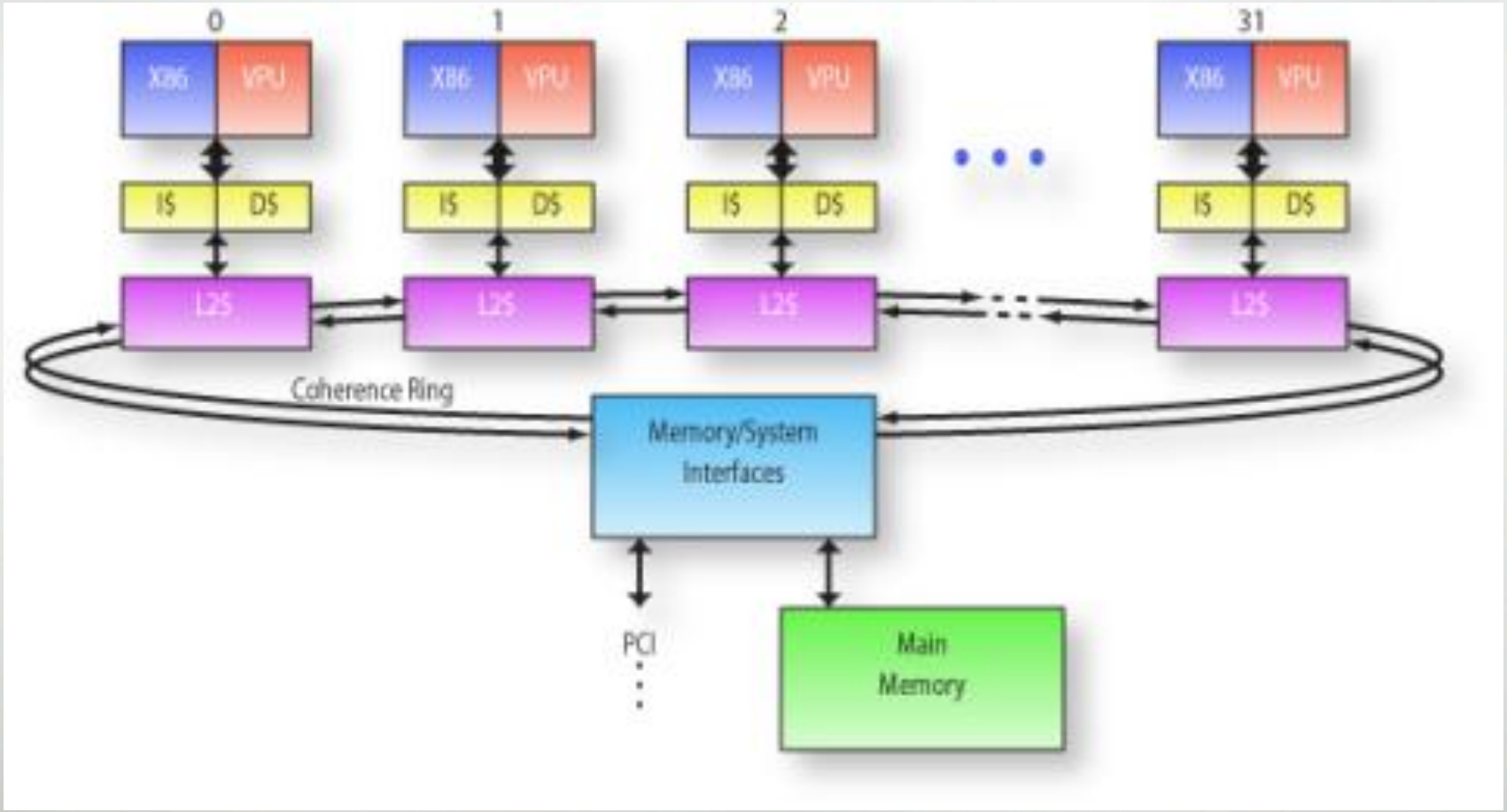
Structural Issues with Accelerated Computing
Even with fused products



Programming Issues with Accelerated Computing

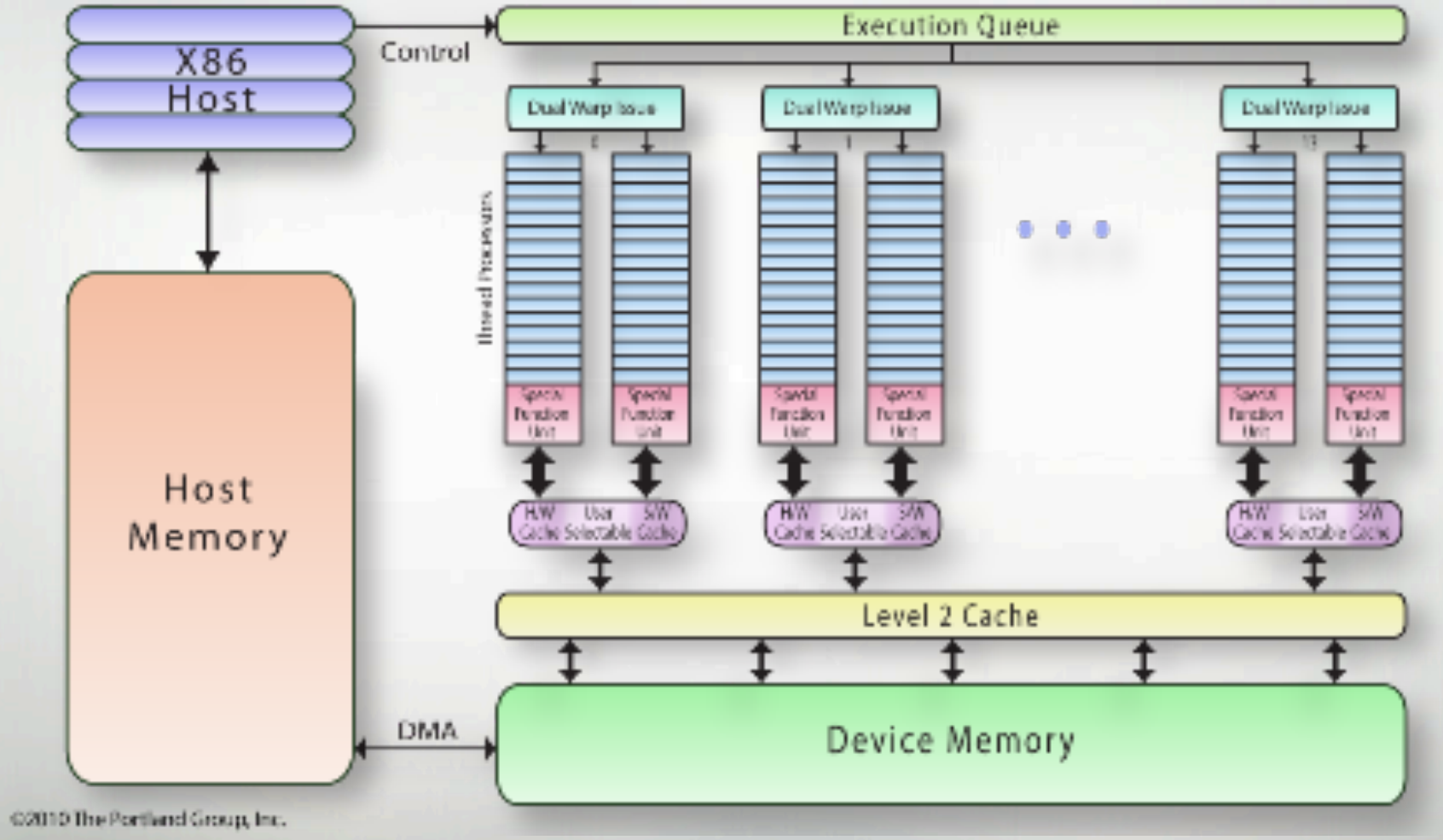
- Primary issues with programming for GPUs:
  - Learn new language/programming model
  - Maintain two code bases/lack of portability
  - Tuning for complex processor architecture (and split CPU/GPU structure)
- Need a single programming model that is portable across machine types, and also forward scalable in time
  - Portable expression of heterogeneity and multi-level parallelism
  - Programming model and optimization should not be significantly difference for "accelerated" nodes and multi-core x86 processors
  - *Allow users to maintain a single code base*
- Need to shield user from the complexity of dealing with heterogeneity
  - High level language with good complier and runtime support
  - Optimized libraries for heterogeneous multicore processors
- Directive-based approach makes sense (OpenACC)
- Getting the division of labor right:
  - User should focus on identifying parallelism (we can help with good tools)
  - Compiler and runtime can deal with mapping it onto the hardware

Intel ‘s Knight's Ferry



From Michael Wolfe’s HPC Article

Nvidia Fermi



From Michael Wolfe's HPC Article

	Intel MIC	NVIDIA Fermi
MIMD Parallelism	32	32
SIMD Parallelism	16	16
Instruction-Level Parallelism	2	1
Thread Granularity	coarse	fine
Multithreading	4	24
Clock	1.2GHz	1.1GHz
L1 cache/processor	32KB	64KB
L2 cache/processor	256KB	24KB
programming model	posix threads/ Directives	CUDA kernels/Directives
virtual memory	yes	no
memory shared with host	no	no
hardware parallelism support	no	yes
mature tools	yes	yes

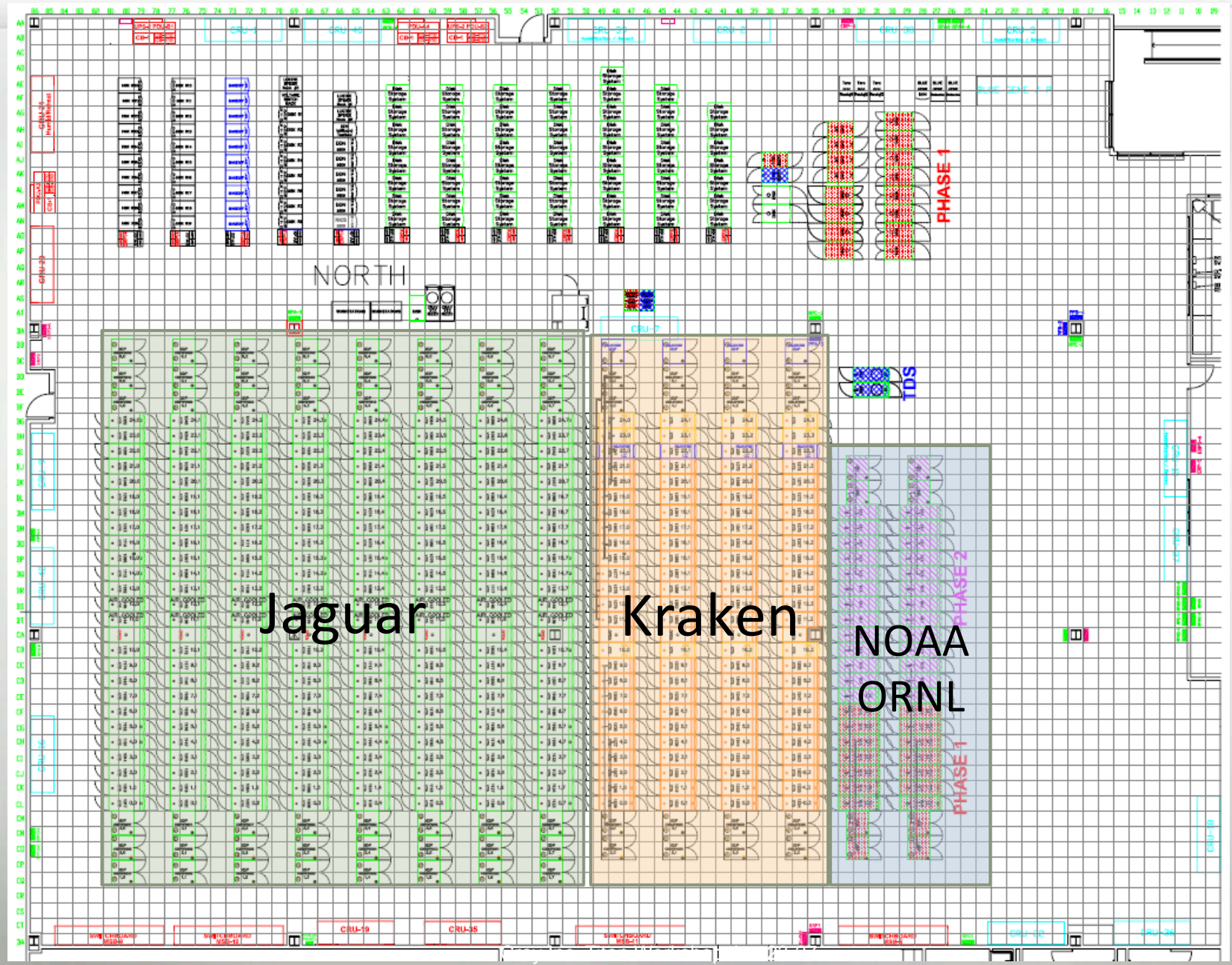
From Michael Wolfe's HPC Article

Short Term Petascale Systems – Node Architecture

	Cores on the node	Total threading	Vector Length	Programming Model
Blue Waters	16	32	8	OpenMP/MPI/ Vector
Blue Gene Q	16	32	8	OpenMP/MPI/ Vector
Magna-Cours	24	24	4	OpenMP/MPI/ Vector
Titan	16	32 (768*)	16	Threads/ Cuda/Vector
Intel MIC	32	128	8	OpenMP/MPI/ Vector
Interlagos	32	64	8	OpenMP/MPI/ Vector

* Nvidia allows oversubscription to SIMT units

One Room – 3 Petascale Systems



Hybrid Multi-core Architecture

- Massively Parallel System with high powered nodes that exhibit
  - Multiple levels of parallelism
    - Shared Memory parallelism on the node
    - SIMD vector units on each core or thread
  - Potentially disparate processing units
    - Host with conventional X86 architecture
    - Accelerator with highly parallel – SIMD units
  - Potentially disparate memories
    - Host with conventional DDR memory
    - Accelerator with high bandwidth memory

Hybrid Multi-core Architecture

- All MPI may not be best approach
  - Memory per core will decrease
  - Injection bandwidth/core will decrease
  - Memory bandwidth/core will decrease
- Hybrid MPI + threading on node may be able to
  - Save Memory
  - Reduce amount of off node communication required
  - Reduce amount of memory bandwidth required

Cray XK6 -- Hybrid Innovation

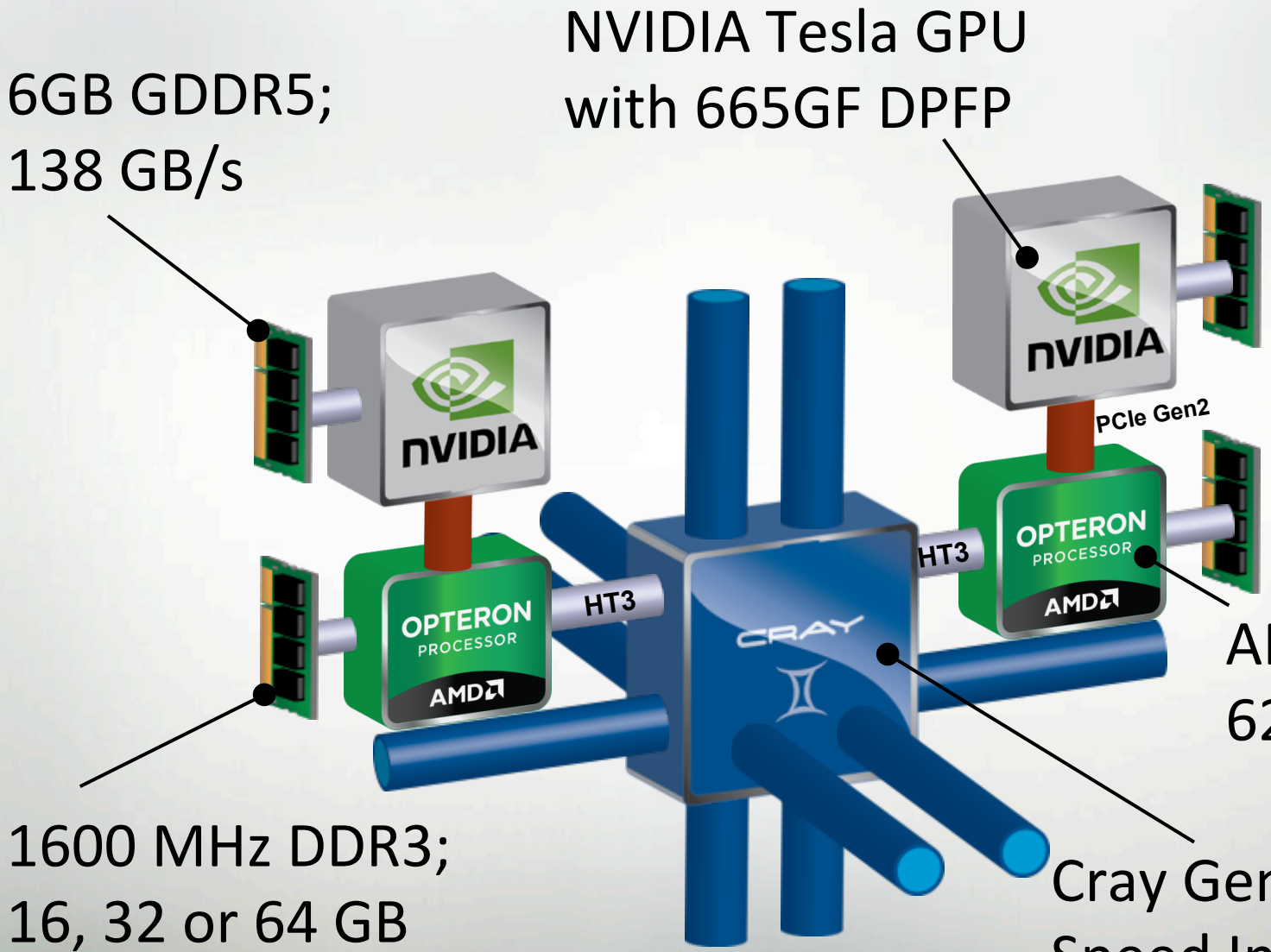
Blending the best-of-the-best into a true hybrid supercomputer

- AMD Series 6200 16-core Interlagos processors
- NVIDIA® Tesla™ 20-series many-core processors
- Cray Gemini High-Performance Interconnect
- CLE and CPE, Cray's scalable software environment

All this in order to create a

Production, scalable, adaptive supercomputer — putting our customers on the road to productive exascale computing

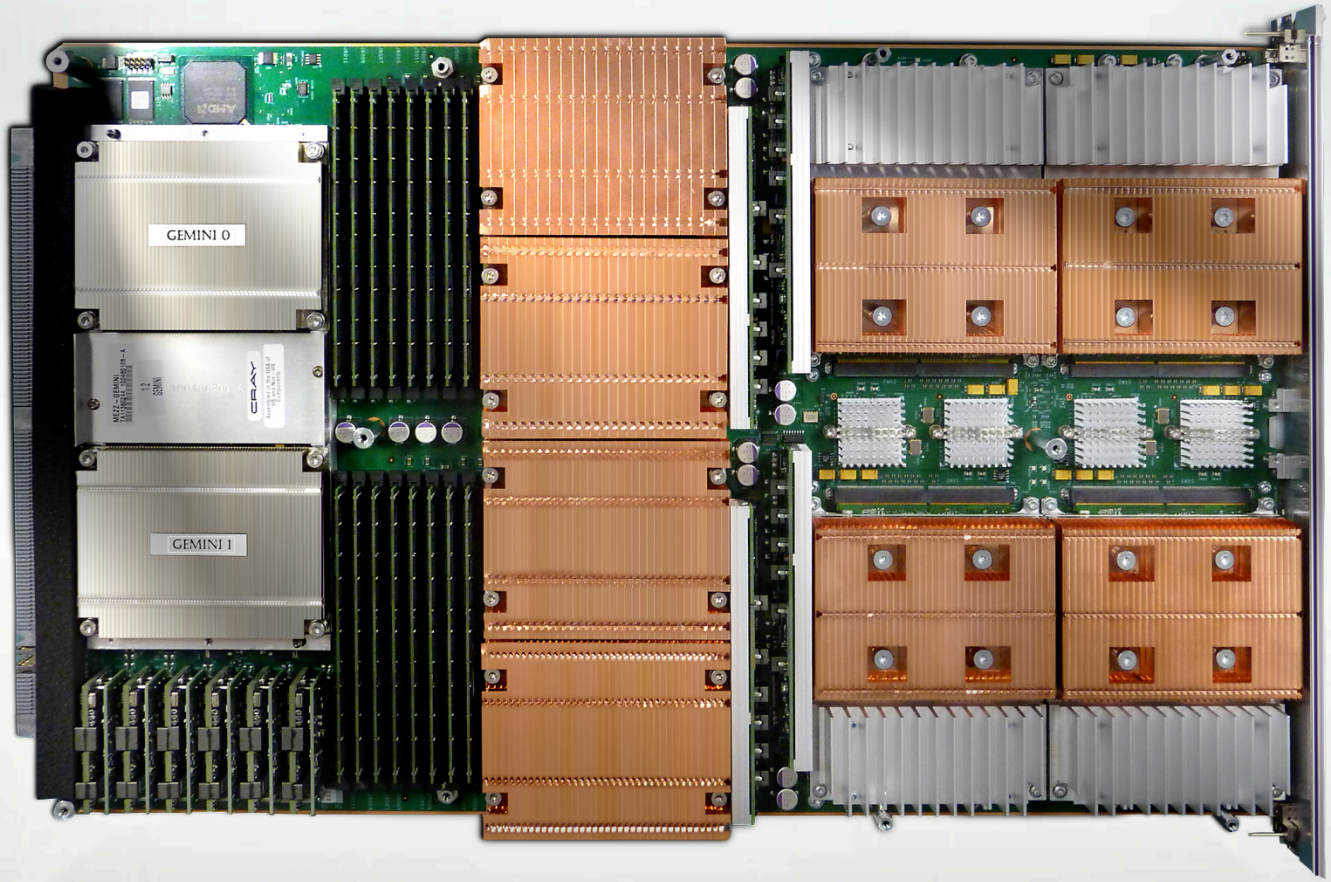
Accelerated Performance through Integration



And then there is Intel's MIC processor

- Current MICs have 32 Intel processors moving to 50 processors, both of these systems have vector length of 512 bits (8 – 64 bit words of 16-32 bit words)
- While Intel is saying that codes can be compiled directly for the MIC (Including MPI), one has to be concerned about
  - The scalar performance of one of those cores
  - The amount of memory on the MIC
- If there is too much scalar code and/or too much memory required, then the MIC will necessarily be treated like the other accelerators
  - Two disparate memories
  - Two disparate computational engines

Greater than 3TF per blade

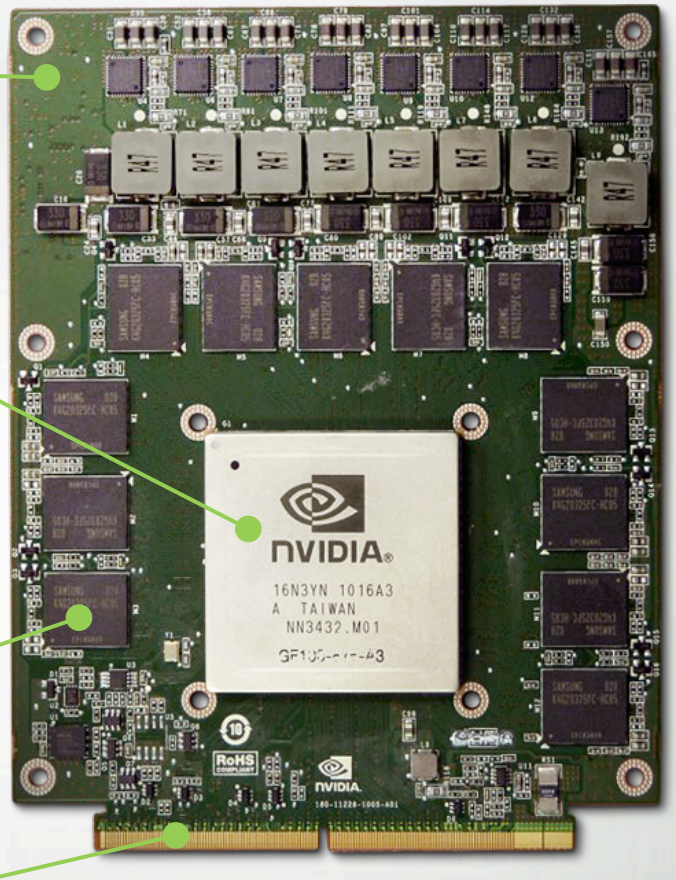


NVIDIA SXM is Feature Rich for HPC

High density form factor at less than 225 W

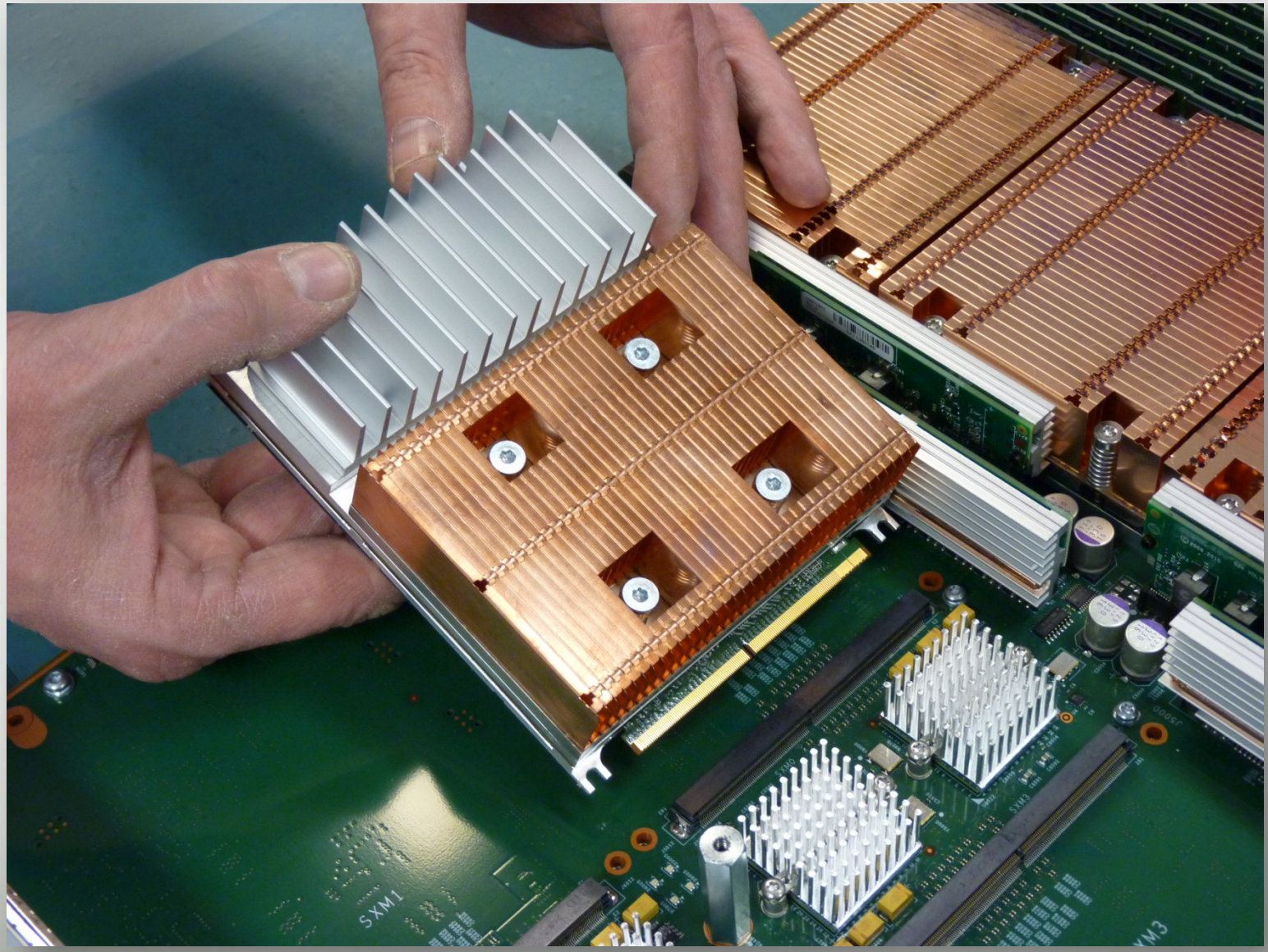
665 GF double precision floating point with ECC protection

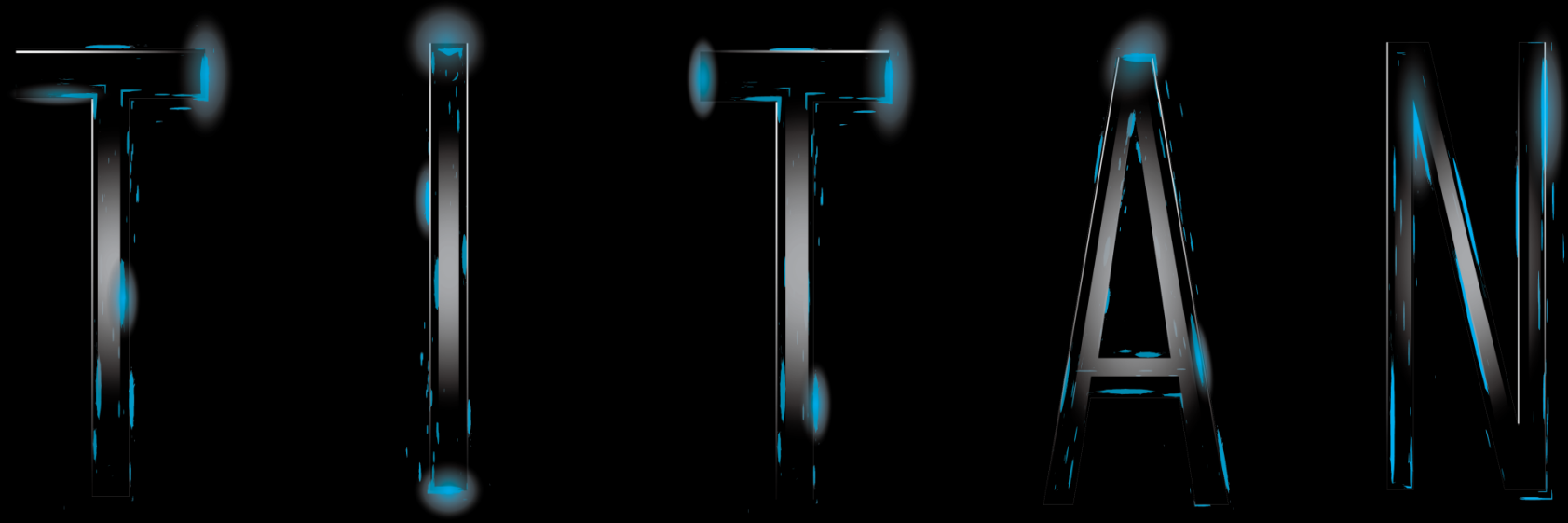
6 GB of GDDR5 memory available at 138 GB/s



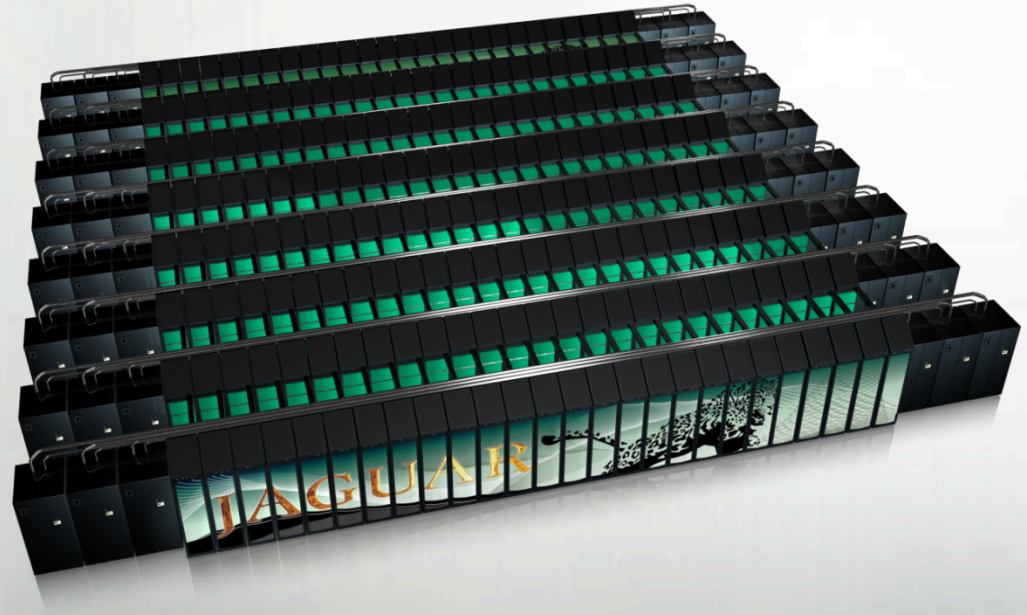
Field upgradeable to Kepler in 2012 for over 1 TF of peak double precision performance

Easily Upgrade to Future Accelerators




TITAN

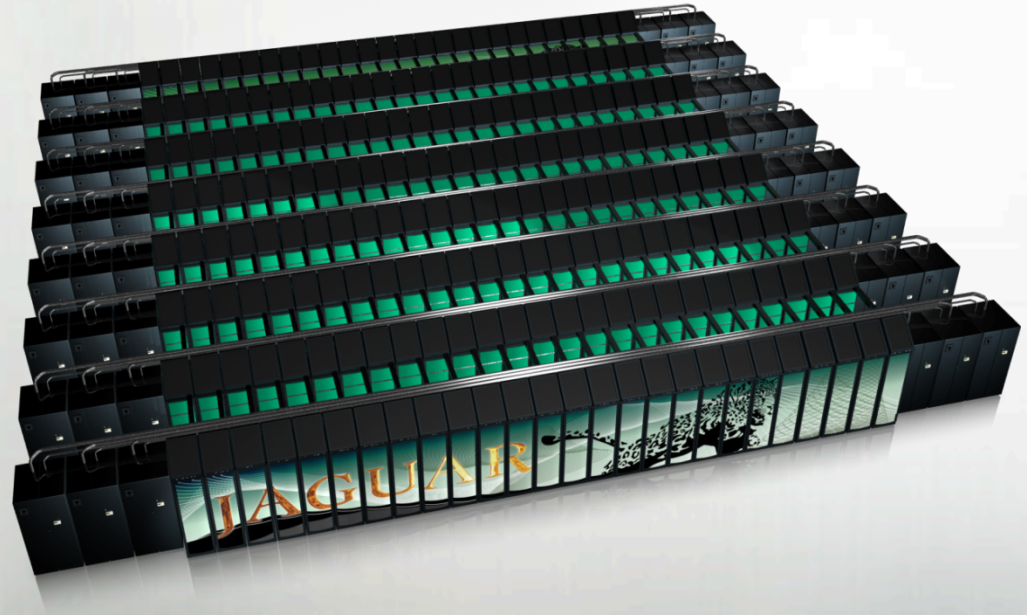
OLCF
OAK RIDGE LEADERSHIP COMPUTING FACILITY



Pre Upgrade Configuration

Name	Jaguar
Architecture	XT5
Processor	6-Core AMD
Cabinets	200
Nodes	18,688
Cores/node	12
Total Cores	224,256
Memory/Node	16 GB
Memory/Core	1.3 GB
Interconnect	SeaStar2+
GPUs	0

OLCF
OAK RIDGE LEADERSHIP COMPUTING FACILITY

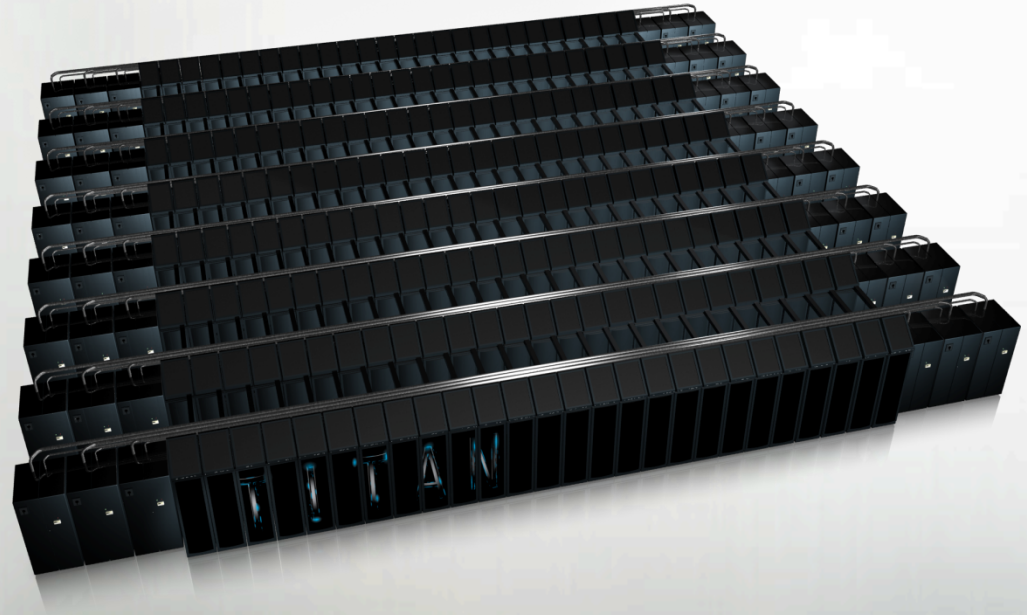


2011 Configuration

Name	Jaguar
Architecture	**XK6**
Processor	**16-Core AMD**
Cabinets	200
Nodes	18,688
Cores/node	**16**
Total Cores	**299,008**
Memory/Node	**32 GB**
Memory/Core	**2 GB**
Interconnect	**Gemini**
GPUs	**960**

And Why is it call an XK6?

OLCF
OAK RIDGE LEADERSHIP COMPUTING FACILITY



Final Configuration

Name	**Titan**
Architecture	XK6
Processor	16-Core AMD
Cabinets	200
Nodes	18,688
Cores/node	16
Total Cores	299,008
Memory/Node	32 GB
Memory/Core	2 GB
Interconnect	Gemini
GPUs	**TBD**

Interlagos

Customer Documentation and Training

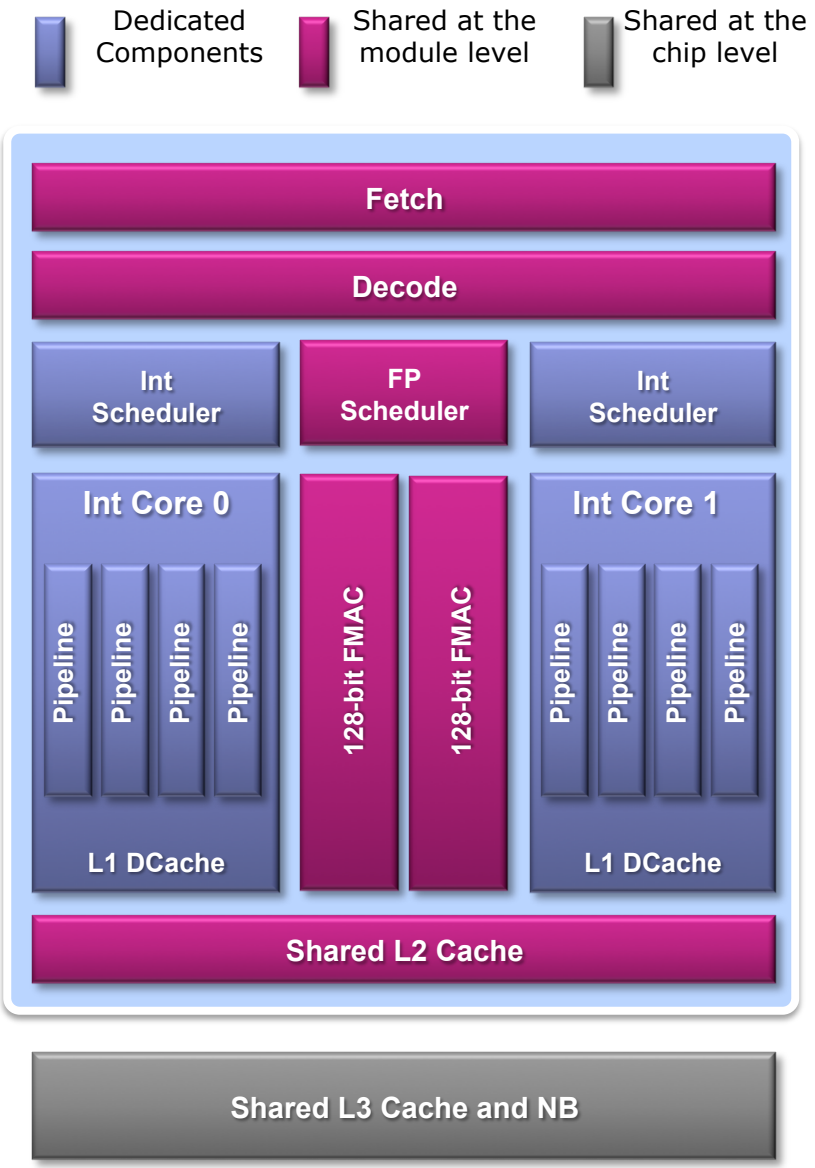
Interlagos Core Definition

- In order to optimize the utilization of the shared and dedicated resources on the chip for different types of applications, modern x86 processors offer flexible options for running applications. As a result, the definition of a *core* has become ambiguous.

- Definition of a Core for Blue Waters:
  - Equivalent to an AMD "Interlagos" Compute Unit, which is an AMD Interlagos "Bulldozer module" consisting of: one instruction fetch/decode unit, one floating point scheduler with two FMAC execution units, two integer schedulers with multiple pipelines and L1 Dcache, and a L2 cache. This is sometimes also called a "Core Module." A "core" = "compute unit" = "core module."

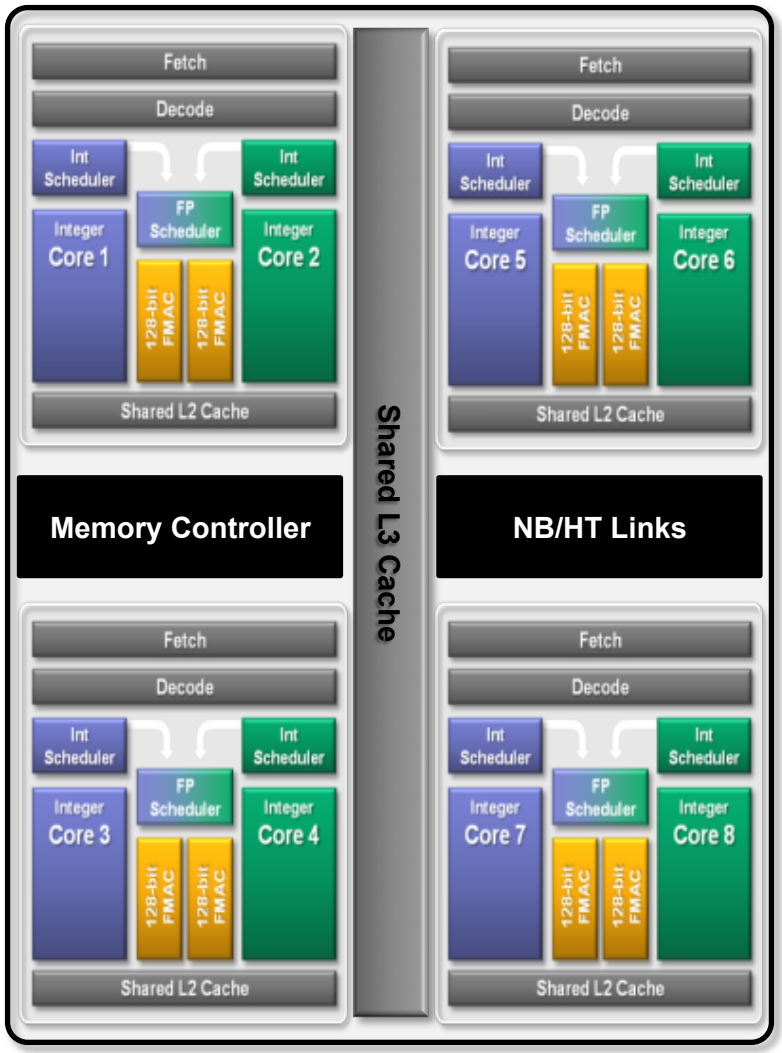
Interlagos Processor Architecture

- Interlagos is composed of a number of “Bulldozer modules” or “Compute Unit”
  - A compute unit has shared and dedicated components
    - There are two independent integer units; shared L2 cache, instruction fetch, Icache; and a *shared*, 256-bit Floating Point resource
  - A single Integer unit can make use of the entire Floating Point resource with 256-bit AVX instructions
    - Vector Length
      - 32 bit operands, VL = 8
      - 64 bit operands, VL = 4

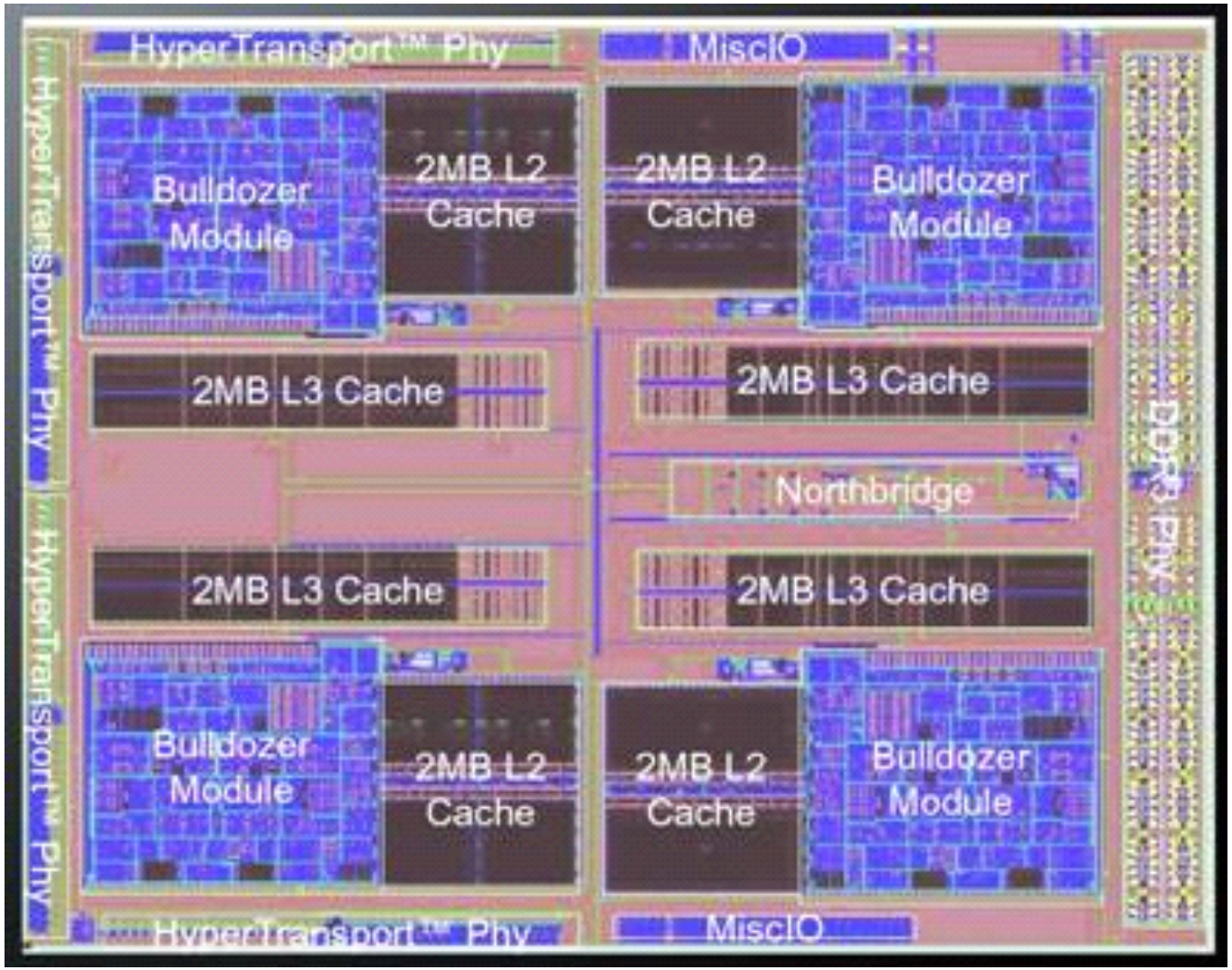


Building an Interlagos Processor

- Each processor die is composed of 4 compute units
  - The 4 compute units share a memory controller and 8MB L3 data cache
    - Each processor die is configured with two DDR3 memory channels and multiple HT3 links

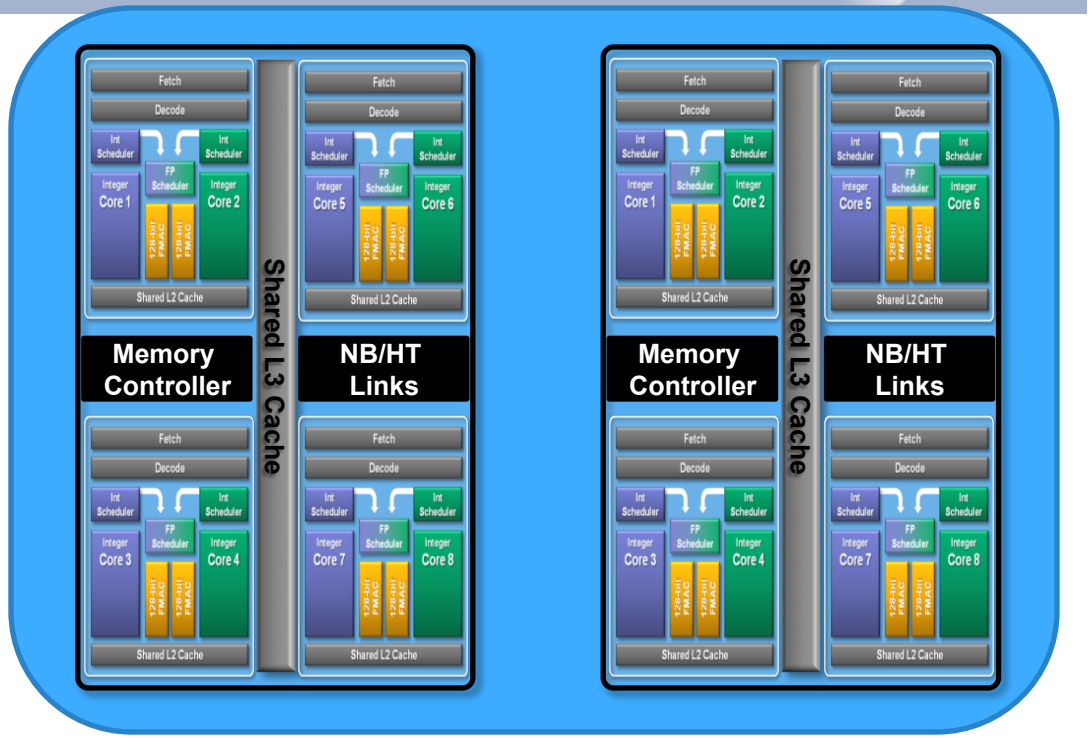


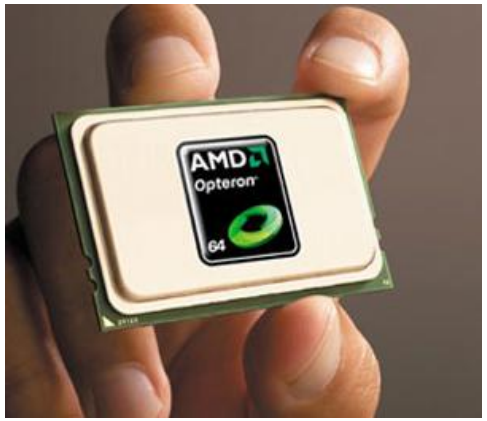
Interlagos Die Floorplan



Interlagos Processor

- Two die are packaged on a multi-chip module to form an Interlagos processor
  - Processor socket is called G34 and is compatible with Magny Cours
  - Package contains
    - 8 compute units
    - 16 MB L3 Cache
    - 4 DDR3 1333 or 1600 memory channels



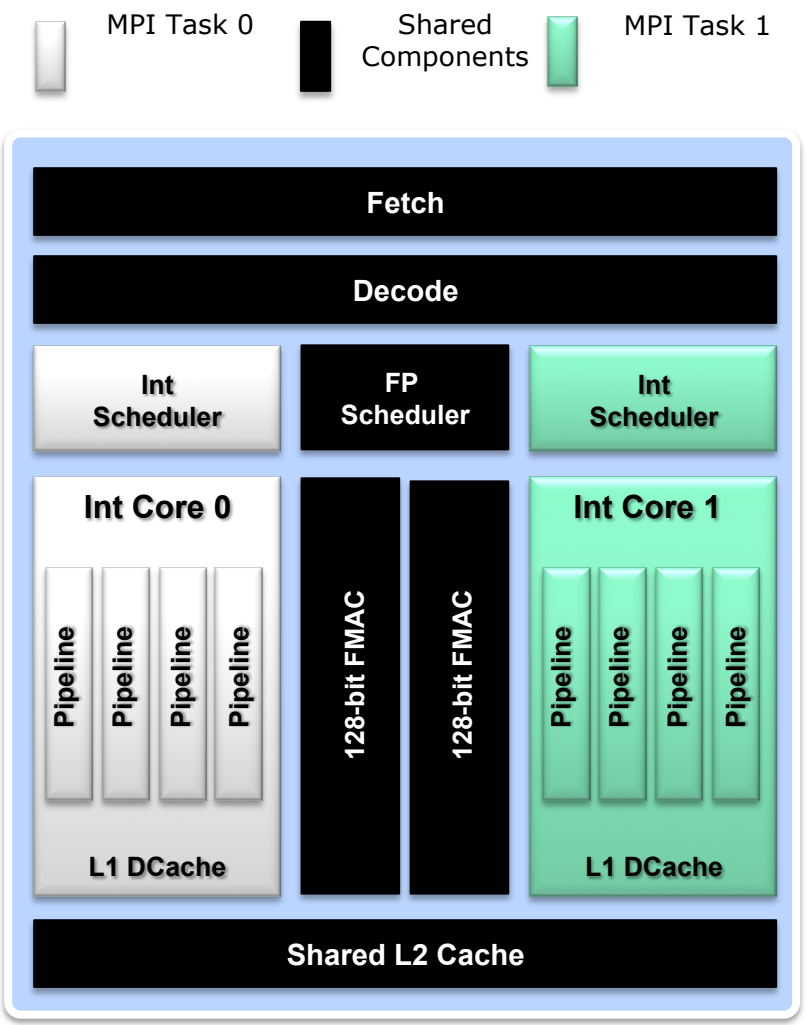


Interlagos Caches and Memory

- L1 Cache
  - 16 KB, 4-way predicted, parity protected
  - Write-through and inclusive with respect to L2
  - 4 cycle load to use latency
- L2 Cache
  - 2MB, Shared within core-module
  - 18-20 cycle load to use latency
- L3 Cache
  - 8 MB, non-inclusive victim cache (mostly exclusive)
    - Entries used by multiple core-modules will remain in cache
    - 1 to 2 MB used by probe filter (snoop bus)
    - 4 sub-caches, one close to each compute module
    - Minimum Load to latency of 55-60 cycles
- Minimum latency to memory is 90-100 cycles

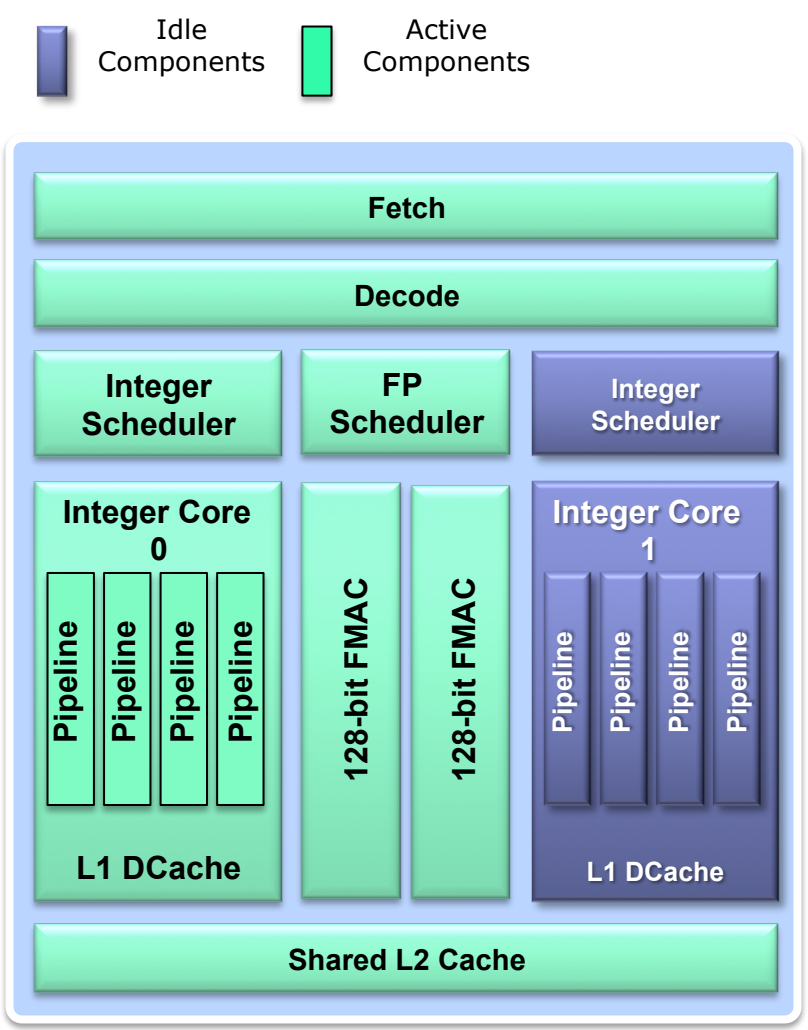
Two MPI Tasks on a Compute Unit ("Dual-Stream Mode")

- An MPI task is pinned to each integer unit
  - Each integer unit has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit, instruction fetch, and the L2 Cache are shared between the two integer units
    - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code is highly scalable to a large number of MPI ranks
  - Code can run with a 2GB per task memory footprint
  - Code is not well vectorized



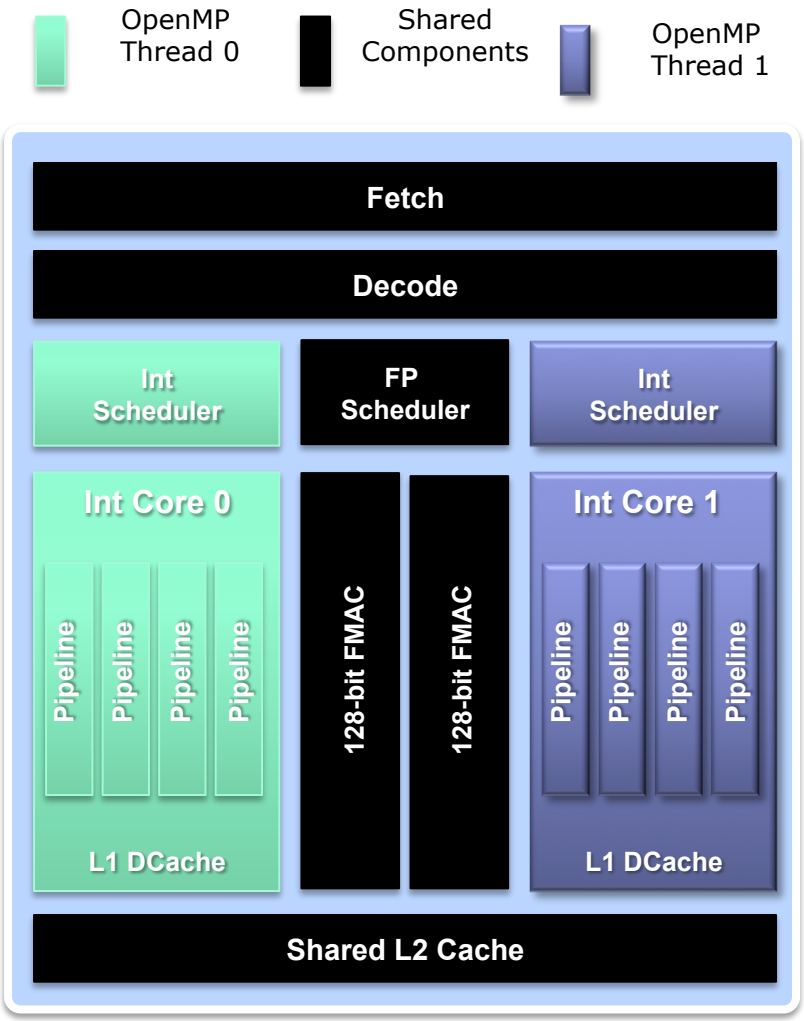
One MPI Task on a Compute Unit ("Single Stream Mode")

- **Only one integer unit is used per compute unit**
  - **This unit has exclusive access to the 256-bit FP unit and is capable of 8 FP results per clock cycle**
  - **The unit has twice the memory capacity and memory bandwidth in this mode**
  - **The L2 cache is effectively twice as large**
  - **The peak of the chip is not reduced**
- **When to use**
  - **Code is highly vectorized and makes use of AVX instructions**
  - **Code benefits from higher per task memory size and bandwidth**



One MPI Task per compute unit with Two OpenMP Threads ("Dual-Stream Mode")

- An MPI task is pinned to a compute unit
- OpenMP is used to run a thread on each integer unit
  - Each OpenMP thread has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two threads
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code needs a large amount of memory per MPI rank
  - Code has OpenMP parallelism at each MPI rank



AVX (Advanced Vector Extensions)

- Max Vector length doubled to 256 bit
- Much cleaner instruction set
  - Result register is unique from the source registers
  - Old SSE instruction set always destroyed a source register
- Floating point multiple-accumulate
  - A(1:4) = B(1:4)*C(1:4) + D(1:4) ! Now one instruction
- Next gen of both AMD and Intel will have AVX

- Vectors are becoming more important, not less

Running in Dual-Stream mode

- Dual-Stream mode is the current default mode on the Cray XE6 systems. General use does not require any options. CPU affinity is set automatically by ALPS.
- Use the `aprun -d` option to set the number of OpenMP threads per process. If OpenMP is not used, no `-d` option is required. The `aprun -N` option is used to specify the number of MPI processes to assign per compute node. This is generally needed if OpenMP threads are used and more than one node is used.

Running in Single-Stream mode

- Single-Stream mode is simple to specify on the Cray XE6 systems if no OpenMP threads are used. The `aprun -d` option is set to a value of 2, and CPU affinity is set automatically by ALPS. (Make sure `$OMP_NUM_THREADS` is not set, or is set to a value of 1.)
- When OpenMP threads are used, careful setting of the `aprun -cc cpu_list` option is required to get the desired CPU affinity. A `cpu_list` is map of CPUs to threads. Each process is assigned a list of CPUs, with one CPU per thread. See the `aprun(1)` man page for more details. The `aprun -N` option is used to specify the number of MPI processes to assign per compute node. This is generally needed if more than one node is used in Single-Stream mode. Also, the environment variable `$OMP_NUM_THREADS` needs to be set to the correct number of threads per process.

aprun Examples

- No OpenMP or 1 OpenMP thread per process, 16 processes per compute node

  `-d 2`

- 2 OpenMP threads per MPI process, 8 processes per compute node

  `-N 8 -cc 0,2:4,6:8,10:12,14:16,18:20,22:24,26:28,30`

- 4 OpenMP threads per MPI process, 4 processes per compute node

  `-N 4 -cc 0,2,4,6:8,10,12,14:16,18,20,22:24,26,28,30`

- 8 OpenMP threads per MPI process, 2 processes per compute node

  `-N 2 -cc 0,2,4,6,8,10,12,14:16,18,20,22,24,26,28,30`

- 16 OpenMP threads per MPI process, 1 process per compute node

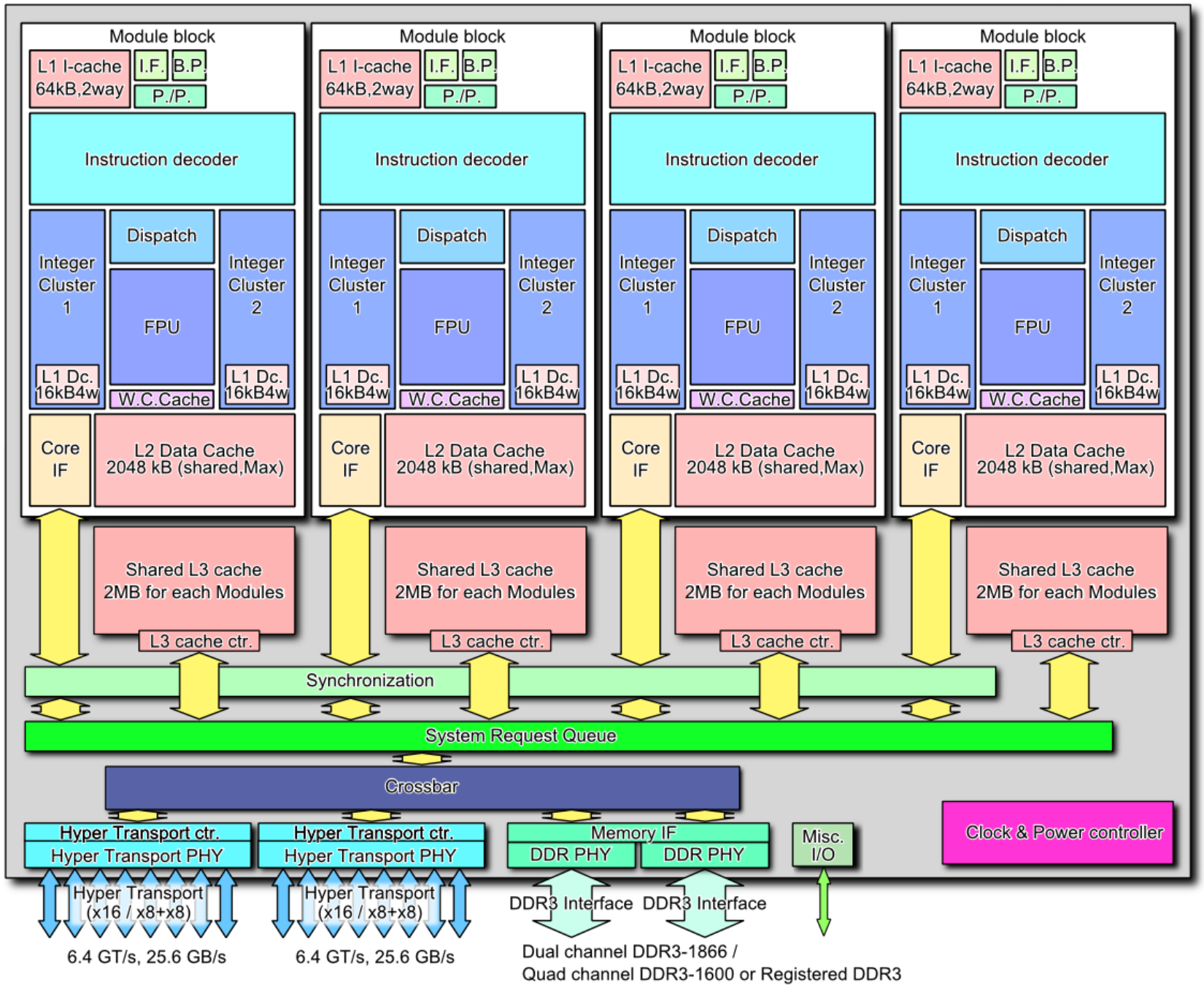
  `-N 1 -cc 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30`

NUMA Considerations

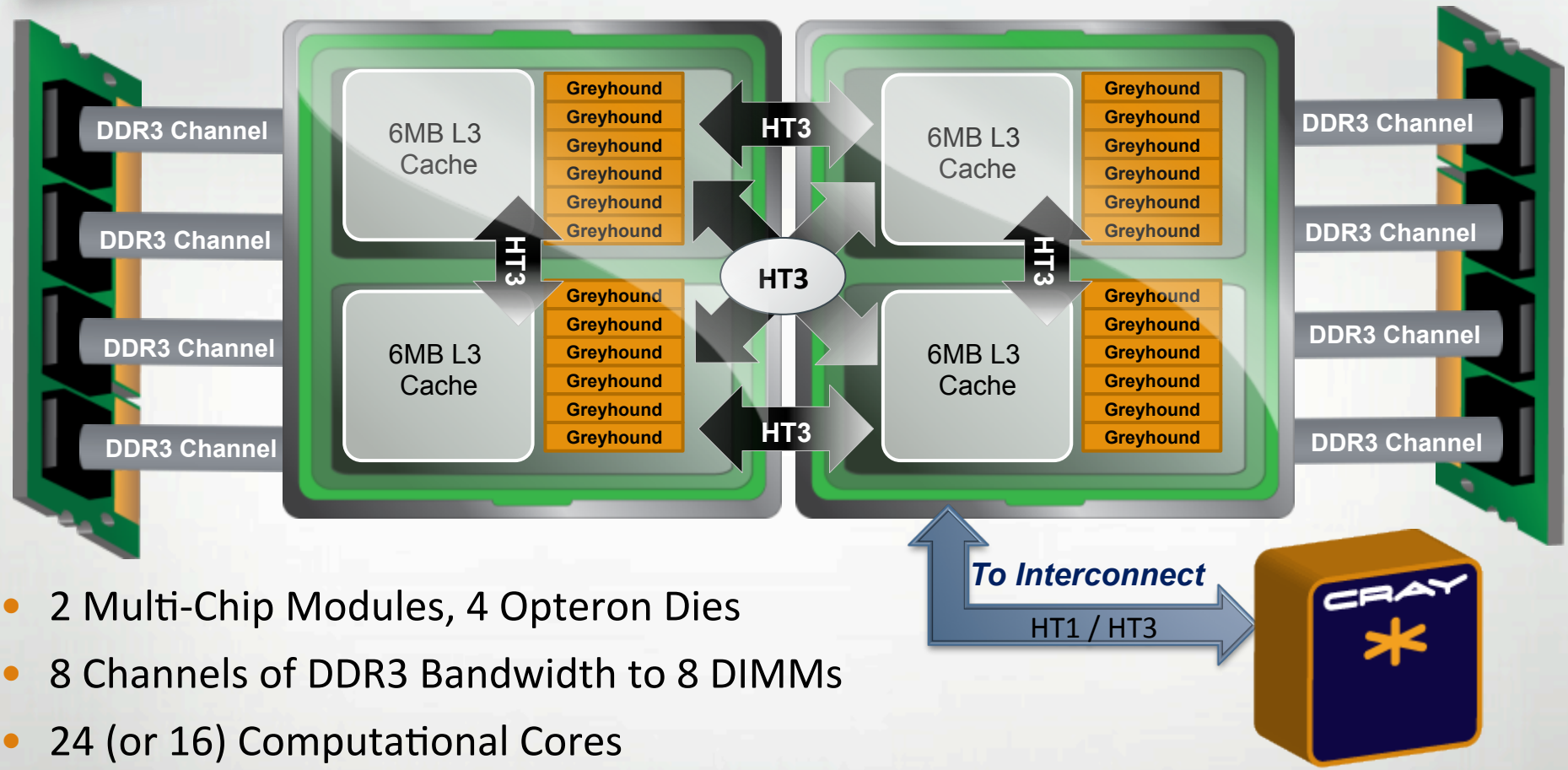
- An XE6 compute node with 2 Interlagos processors has 4 NUMA memory domains, each with 4 Bulldozer Modules. Access to memory located in a remote NUMA domain is slower than access to local memory. Bandwidth is lower, and latency is higher.
- OpenMP performance is usually better when all threads in a process execute in the same NUMA domain. For the Dual-Stream case, 8 CPUs share a NUMA domain, while in Single-Stream mode 4 CPUs share a NUMA domain. Using a larger number of OpenMP threads per MPI process than these values may result in lower performance due to cross-domain memory access.

`aprun` Options Summary

Run Type	Dual-Stream	Single-Stream
No OpenMP	No option needed	-d 2 (note: $OMP_NUM_THREADS not set)
2 OpenMP threads	-N 16 -d 2	-N 8 -cc 0,2:4,6:8,10:12,14:16,18:20,22:24,26:28,30
4 OpenMP threads	-N 8 -d 4	-N 4 -cc 0,2,4,6:8,10,12,14:16,18,20,22:24,26,28,30
8 OpenMP threads	-N 4 -d 8	-N 2 -cc 0,2,4,6,8,10,12,14:16,18,20,22,24,26,28,30
16 OpenMP threads	-N 2 -d 16	-N 1 -cc 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
32 OpenMP threads	-N 1 -d 32	Not Applicable


Module block
L1 I-cache 64kB,2way
I.F.
B.P.
P./P.
Instruction decoder
Integer Cluster 1
Dispatch
FPU
Integer Cluster 2
L1 Dc. 16kB4w
W.C.Cache
L1 Dc. 16kB4w
Core IF
L2 Data Cache 2048 kB (shared,Max)
Shared L3 cache 2MB for each Modules
L3 cache ctr.
Synchronization
System Request Queue
Crossbar
Hyper Transport ctr.
Hyper Transport PHY
Hyper Transport (x16 / x8+x8)
6.4 GT/s, 25.6 GB/s
Memory IF
DDR PHY
DDR3 Interface
Dual channel DDR3-1866 /
Quad channel DDR3-1600 or Registered DDR3
Misc. I/O
Clock & Power controller

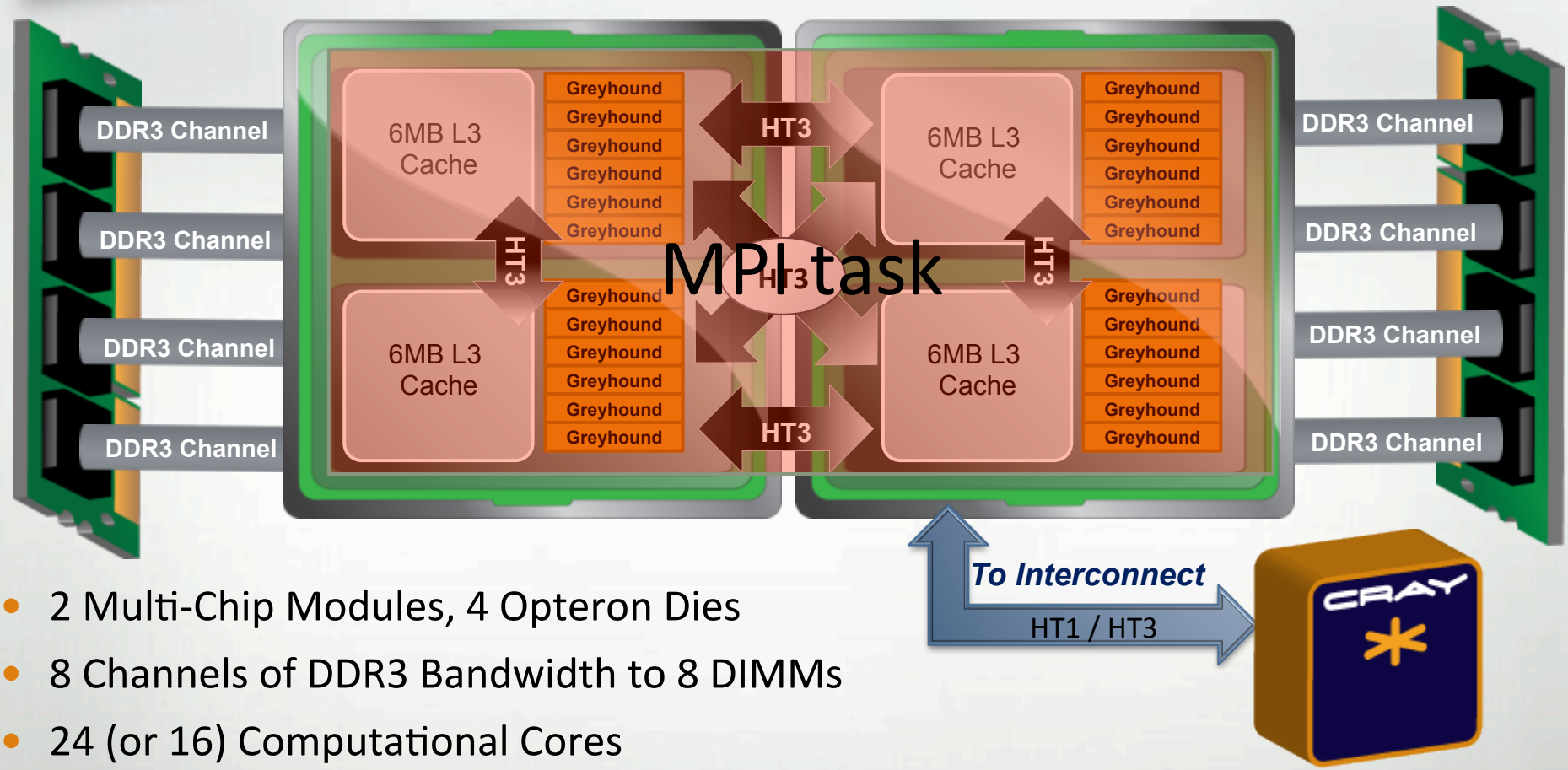
XE6 Node Details: 24-core Magny Cours



- 2 Multi-Chip Modules, 4 Opteron Dies
- 8 Channels of DDR3 Bandwidth to 8 DIMMs
- 24 (or 16) Computational Cores
  - 64 KB L1 and 512 KB L2 caches for each core
  - 6 MB of shared L3 cache on each die
- Dies are fully connected with HT3
- Snoop Filter Feature Allows 4 Die SMP to scale well

XE6 Node Details: 24-core Magny Cours

Run using 1 MPI task on the node

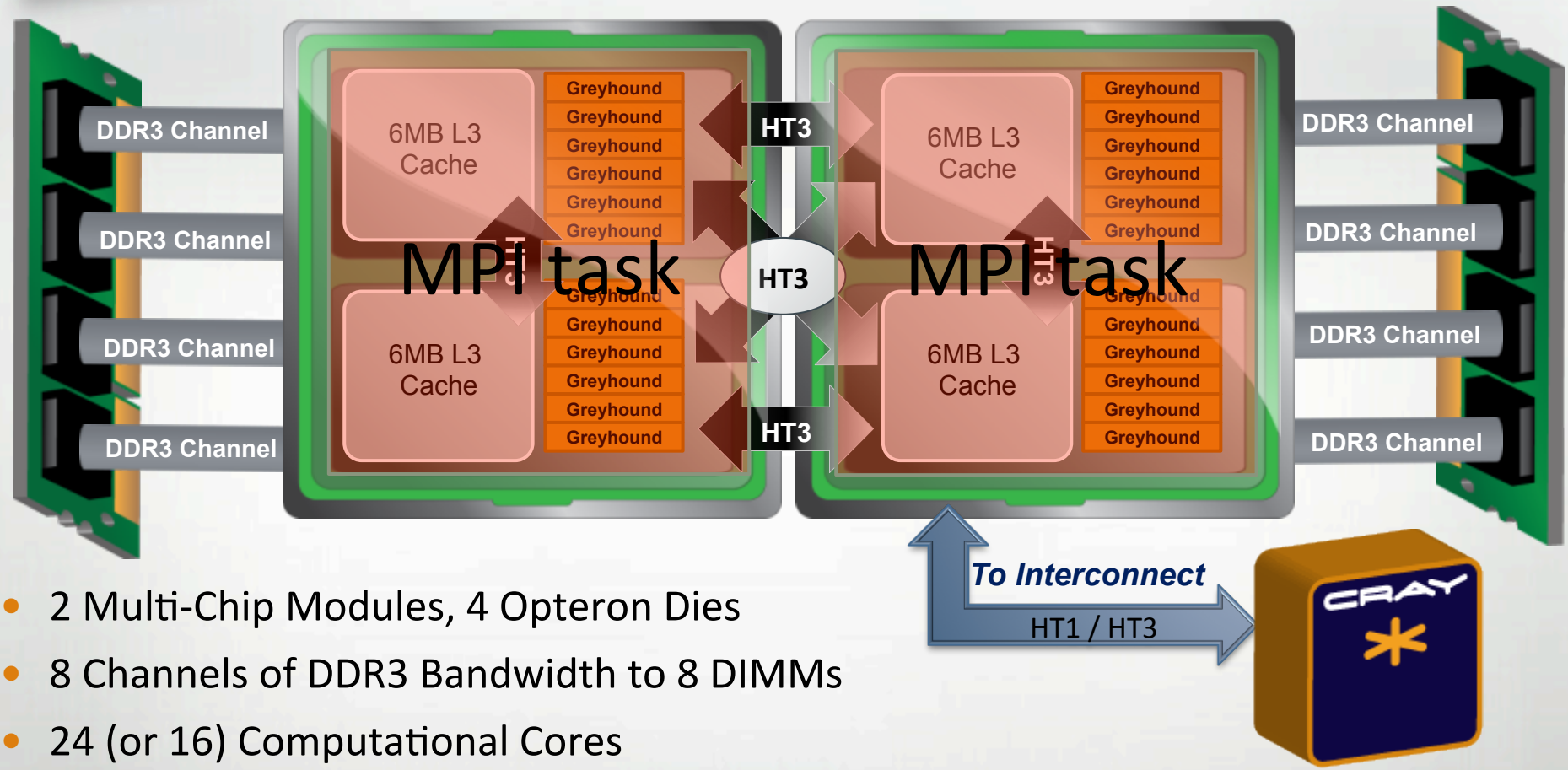


- 2 Multi-Chip Modules, 4 Opteron Dies
- 8 Channels of DDR3 Bandwidth to 8 DIMMs
- 24 (or 16) Computational Cores
  - 64 KB L1 and 512 KB L2 caches for each core
  - 6 MB of shared L3 cache on each die
- Dies are fully connected with HT3
- Snoop Filter Feature Allows 4 Die SMP to scale well

Use OpenMP across all 24 cores

XE6 Node Details: 24-core Magny Cours

Run using 2 MPI tasks on the node
One on Each Die

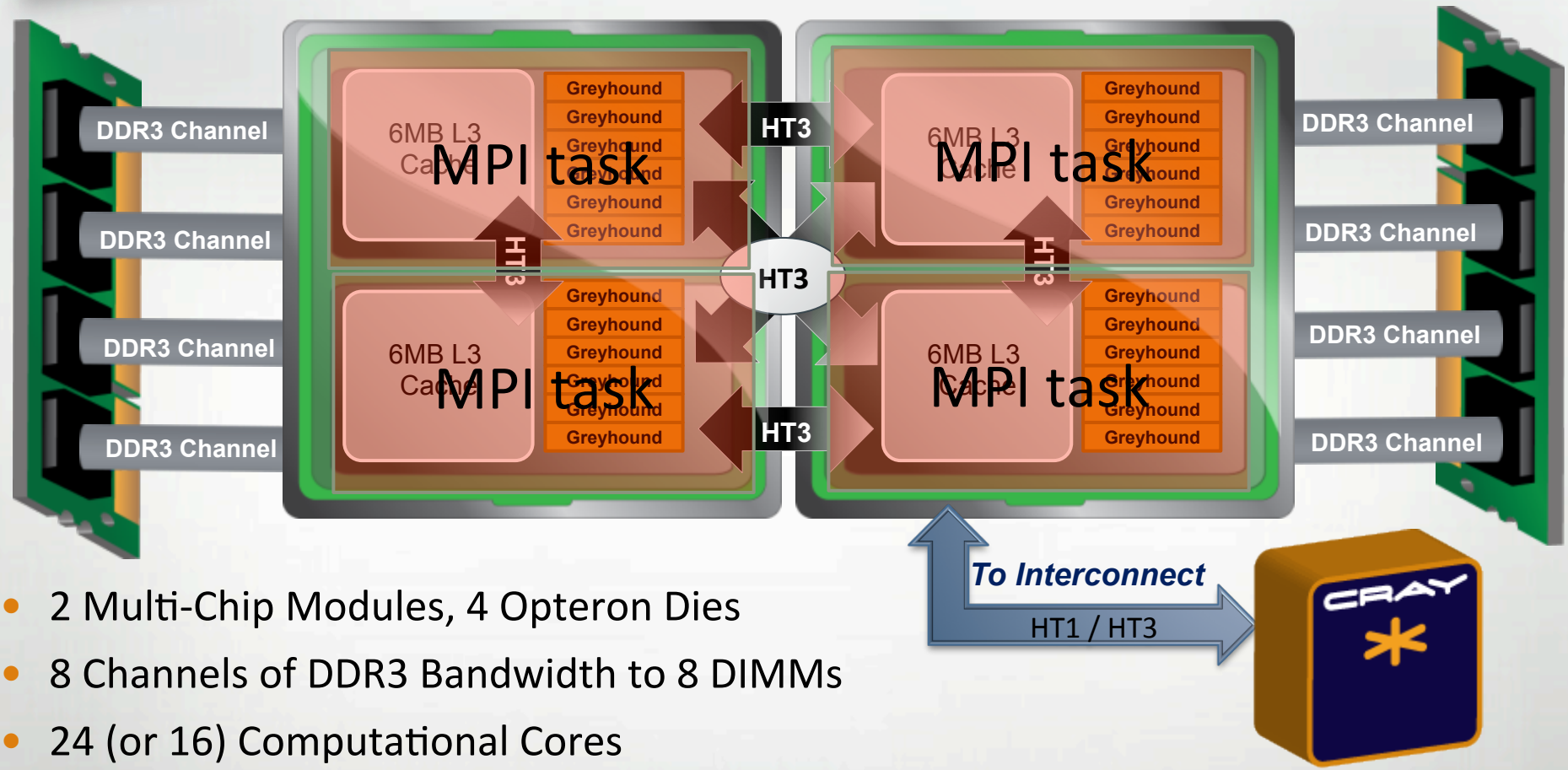


- 2 Multi-Chip Modules, 4 Opteron Dies
- 8 Channels of DDR3 Bandwidth to 8 DIMMs
- 24 (or 16) Computational Cores
  - 64 KB L1 and 512 KB L2 caches for each core
  - 6 MB of shared L3 cache on each die
- Dies are fully connected with HT3
- Snoop Filter Feature Allows 4 Die SMP to scale well

Use OpenMP across all 12 cores in the Die

XE6 Node Details: 24-core Magny Cours

Run using 4 MPI tasks on the node
One on Each Socket



- 2 Multi-Chip Modules, 4 Opteron Dies
- 8 Channels of DDR3 Bandwidth to 8 DIMMs
- 24 (or 16) Computational Cores
  - 64 KB L1 and 512 KB L2 caches for each core
  - 6 MB of shared L3 cache on each die
- Dies are fully connected with HT3
- Snoop Filter Feature Allows 4 Die SMP to scale well

Use OpenMP across all 6 cores in the Socket

Proposed Programming Paradigm for Hybrid Multi-core

- MPI or PGAS between nodes and/or sockets
- OpenMP, Pthreads or some other shared memory parallelism across a portion of the cores on the node
- Vectorization to utilize the SSE# or SIMD units on the cores

CRAY
THE SUPERCOMPUTER COMPANY

Gemini Interconnect

Cray Network Evolution

SeaStar

- Built for scalability to 250K+ cores
- Very effective routing and low contention switch

Gemini

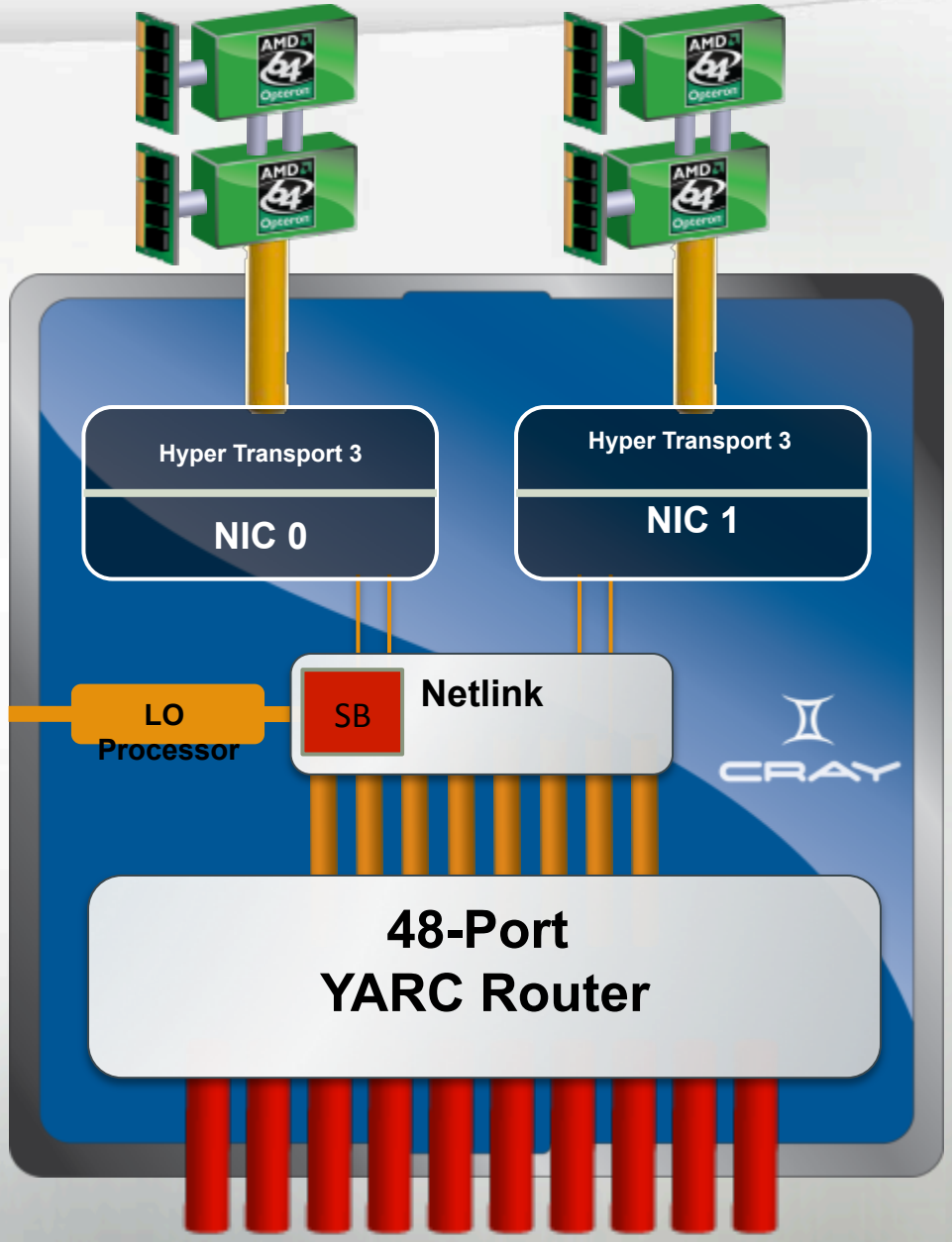
- 100x improvement in message throughput
- 3x improvement in latency
- PGAS Support, Global Address Space
- Scalability to 1M+ cores

Aries

- Ask me about it

Cray Gemini

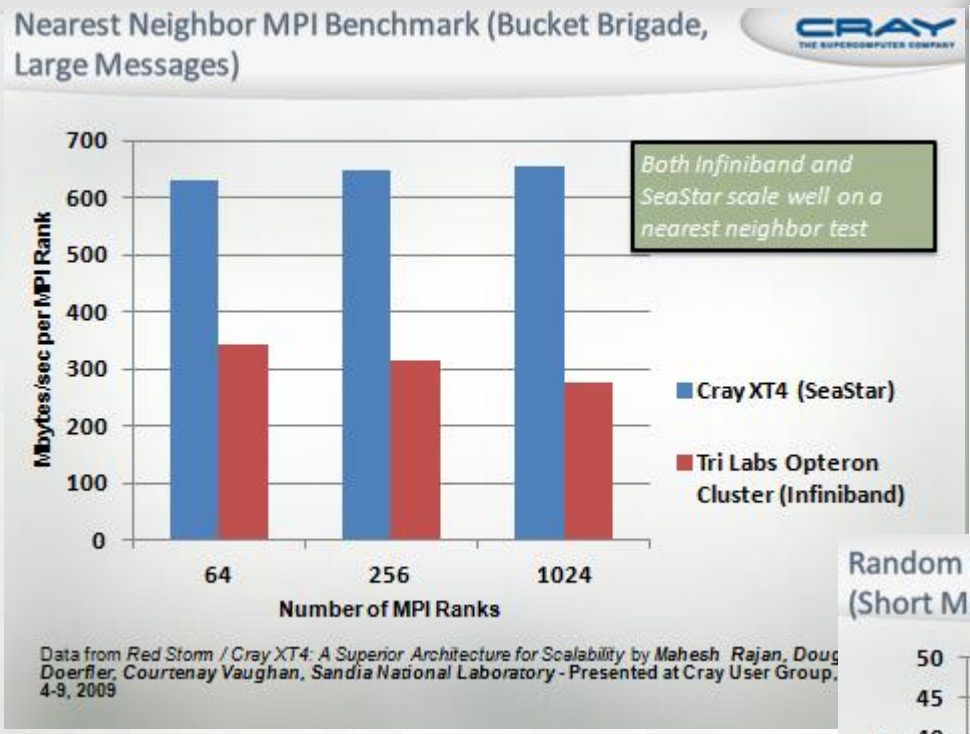
- 3D Torus network
- Supports 2 Nodes per ASIC
- 168 GB/sec routing capacity
- Scales to over 100,000 network endpoints
  - Link Level Reliability and Adaptive Routing
  - Advanced Resiliency Features
- Provides global address space
- Advanced NIC designed to efficiently support
  - MPI
    - Millions of messages/second
  - One-sided MPI
  - UPC, FORTRAN 2008 with coarrays, shmem
  - Global Atomics

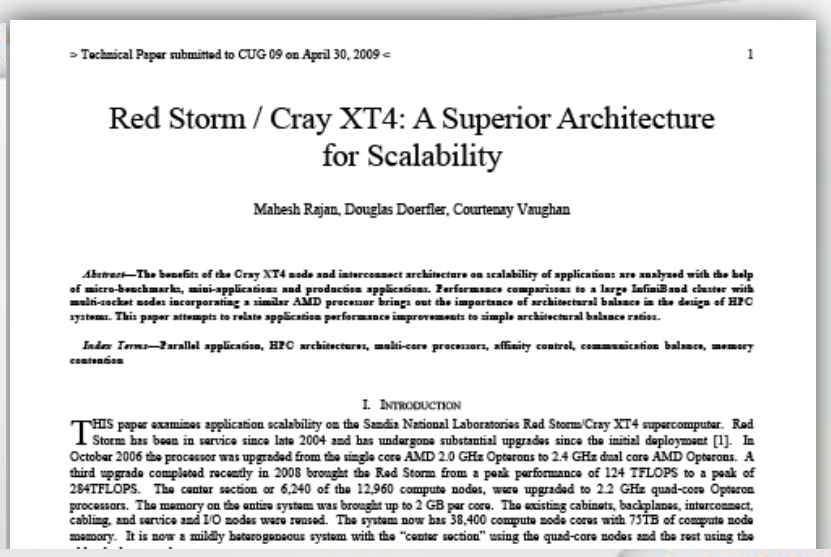


Gemini Advanced Features

- Globally addressable memory provides efficient support for UPC, Co-array FORTRAN, Shmem and Global Arrays
  - Cray Programming Environment will target this capability directly
- Pipelined global loads and stores
  - Allows for fast irregular communication patterns
- Atomic memory operations
  - Provides fast synchronization needed for one-sided communication models

Why Custom Interconnects?




> Technical Paper submitted to CUG 09 on April 30, 2009 < 1

Red Storm / Cray XT4: A Superior Architecture for Scalability

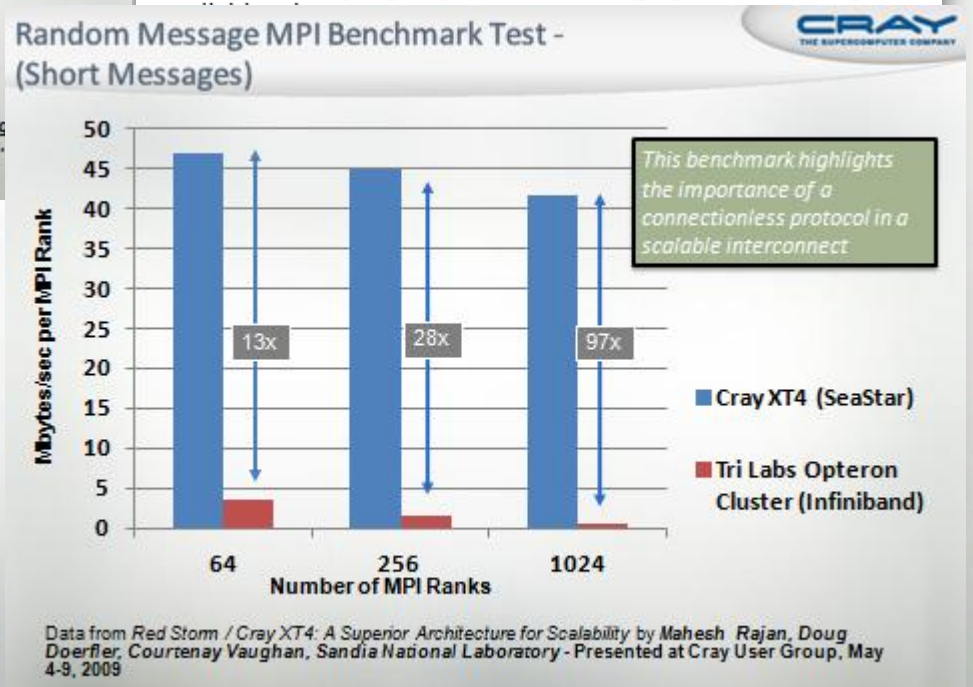
Mahesh Rajan, Douglas Doerfler, Courtenay Vaughan

Abstract—The benefits of the Cray XT4 node and interconnect architecture on scalability of applications are analyzed with the help of micro-benchmarks, mini-applications and production applications. Performance comparisons to a large InfiniBand cluster with multi-socket nodes incorporating a similar AMD processor brings out the importance of architectural balance in the design of HPC systems. This paper attempts to relate application performance improvements to simple architectural balance ratios.

Index Terms—Parallel application, HPC architectures, multi-core processors, affinity control, communication balance, memory contention

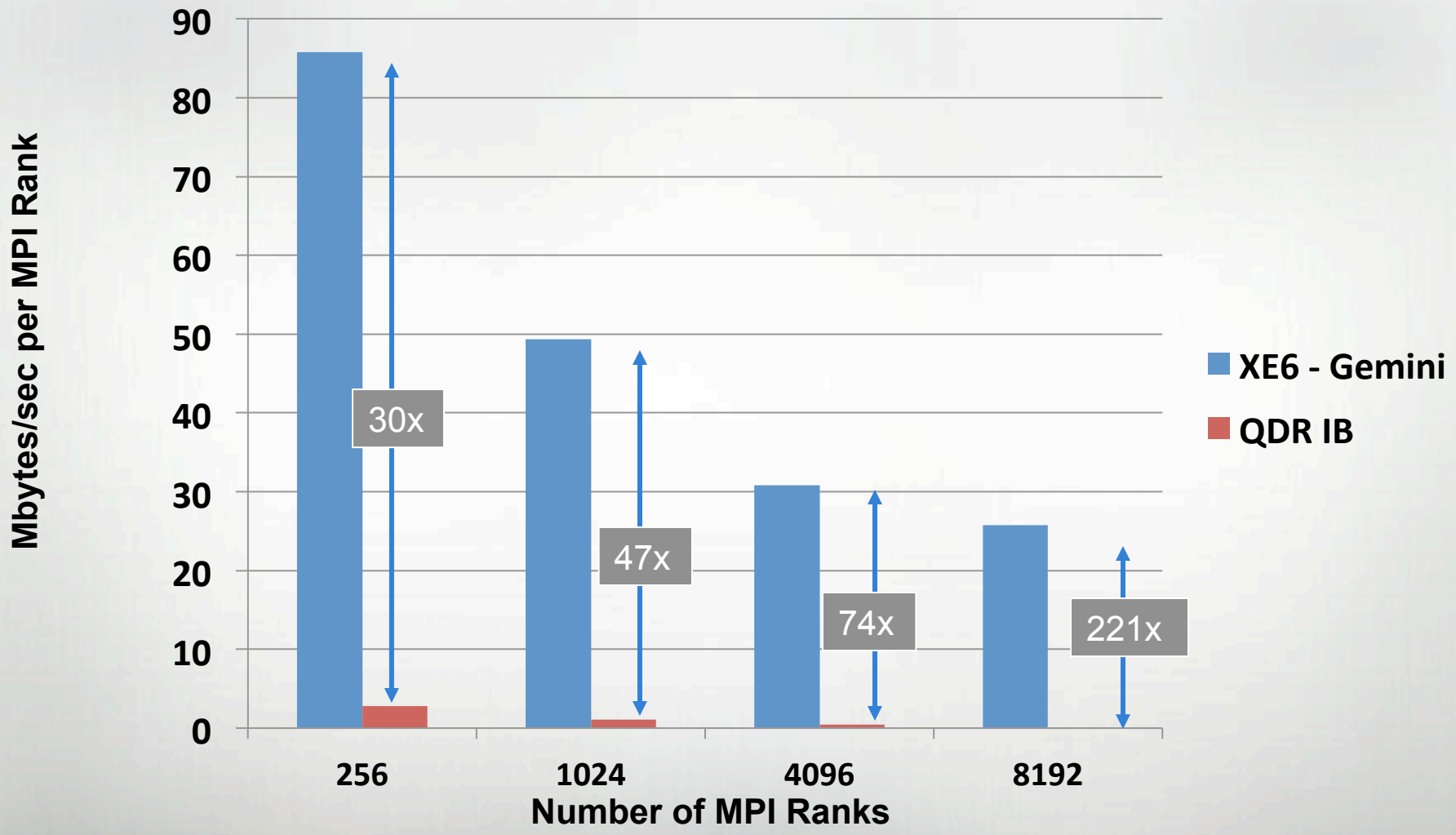
I. Introduction

THIS paper examines application scalability on the Sandia National Laboratories Red Storm/Cray XT4 supercomputer. Red Storm has been in service since late 2004 and has undergone substantial upgrades since the initial deployment [1]. In October 2006 the processor was upgraded from the single core AMD 2.0 GHz Opterons to 2.4 GHz dual core AMD Opterons. A third upgrade completed recently in 2008 brought the Red Storm from a peak performance of 124 TFLOPS to a peak of 284TFLOPS. The center section or 6,240 of the 12,960 compute nodes, were upgraded to 2.2 GHz quad-core Opteron processors. The memory on the entire system was brought up to 2 GB per core. The existing cabinets, backplanes, interconnect, cabling, and service and I/O nodes were reused. The system now has 38,400 compute node cores with 75TB of compute node memory. It is now a mildly heterogeneous system with the "center section" using the quad-core nodes and the rest using the



- What's changed since 2009?
  1. DDR has moved to QDR
  2. SeaStar has moved to Gemini
  3. Cores per node have grown by a factor of 4 to 8
- How has this picture changed?

Random Message MPI Benchmark Test Today Gemini vs. QDR IB



A Comparison of the Performance Characteristics of Capability and Capacity Class HPC Systems
By Douglas Doerfler, Mahesh Rajan, Marcus Epperson, Courtenay Vaughan, Kevin Pedretti, Richard Barrett, Brian Barrett , Sandia National Laboratories

A Comparison of Two Codes with Very Different Communication Patterns

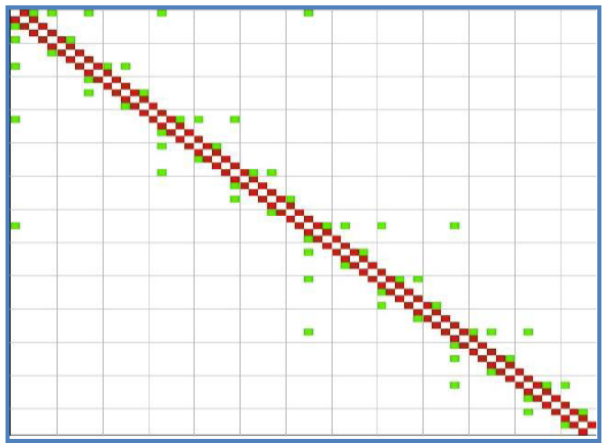


Figure 17. AMG inter-processor communication CrayPat plot

Nearest Neighbor

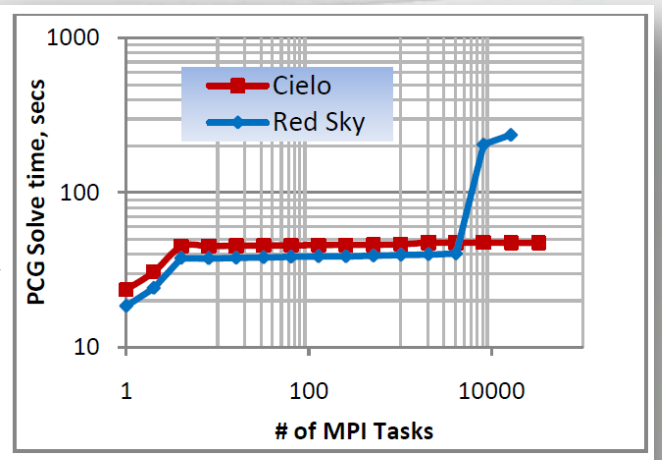


Figure 16. AMG Scaling Performance (Lower is better)

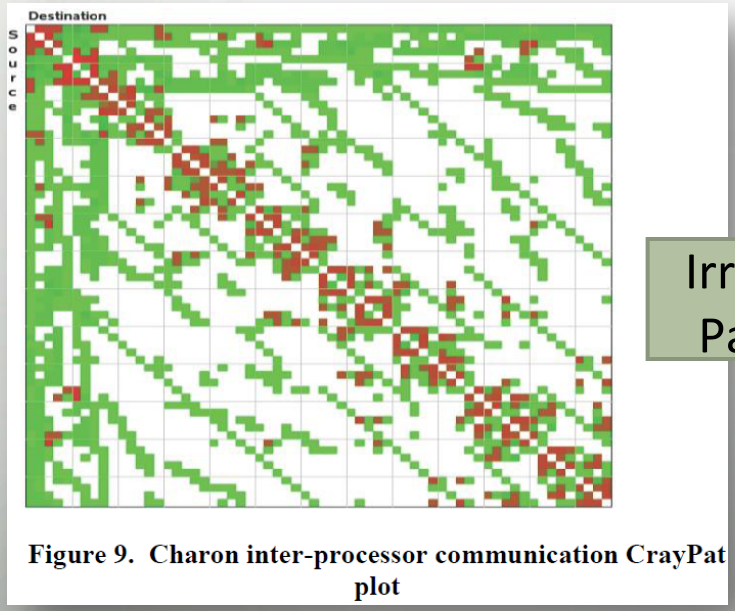


Figure 9. Charon inter-processor communication CrayPat plot

Irregular Pattern

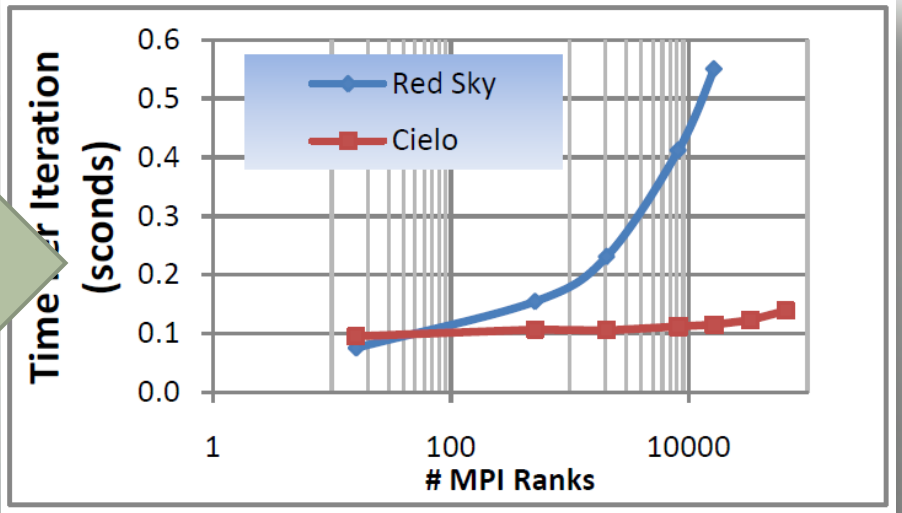
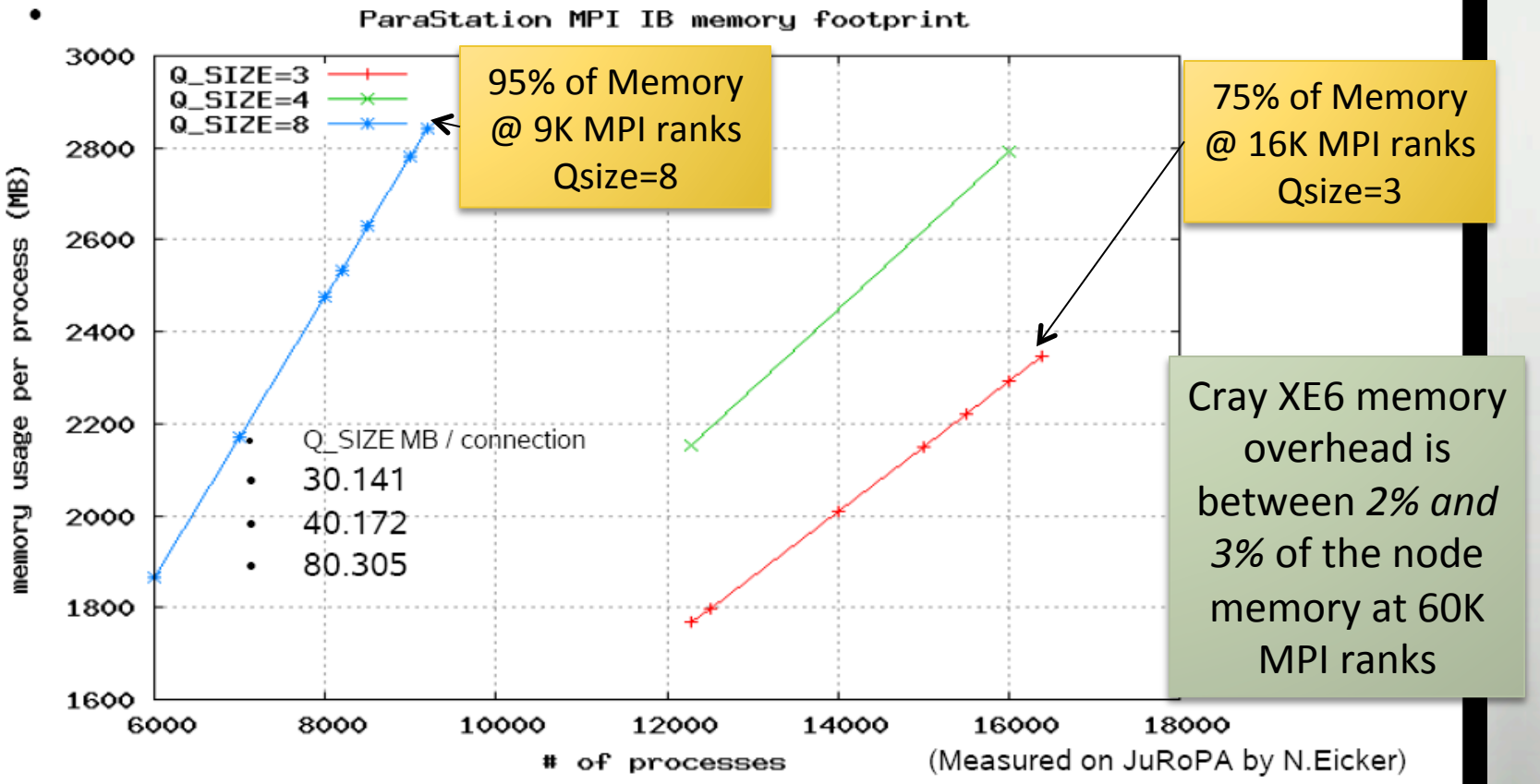


Figure 8. Charon scaling performance (Lower is better)

Bull-Oracle-Sun QDR system @ Julich

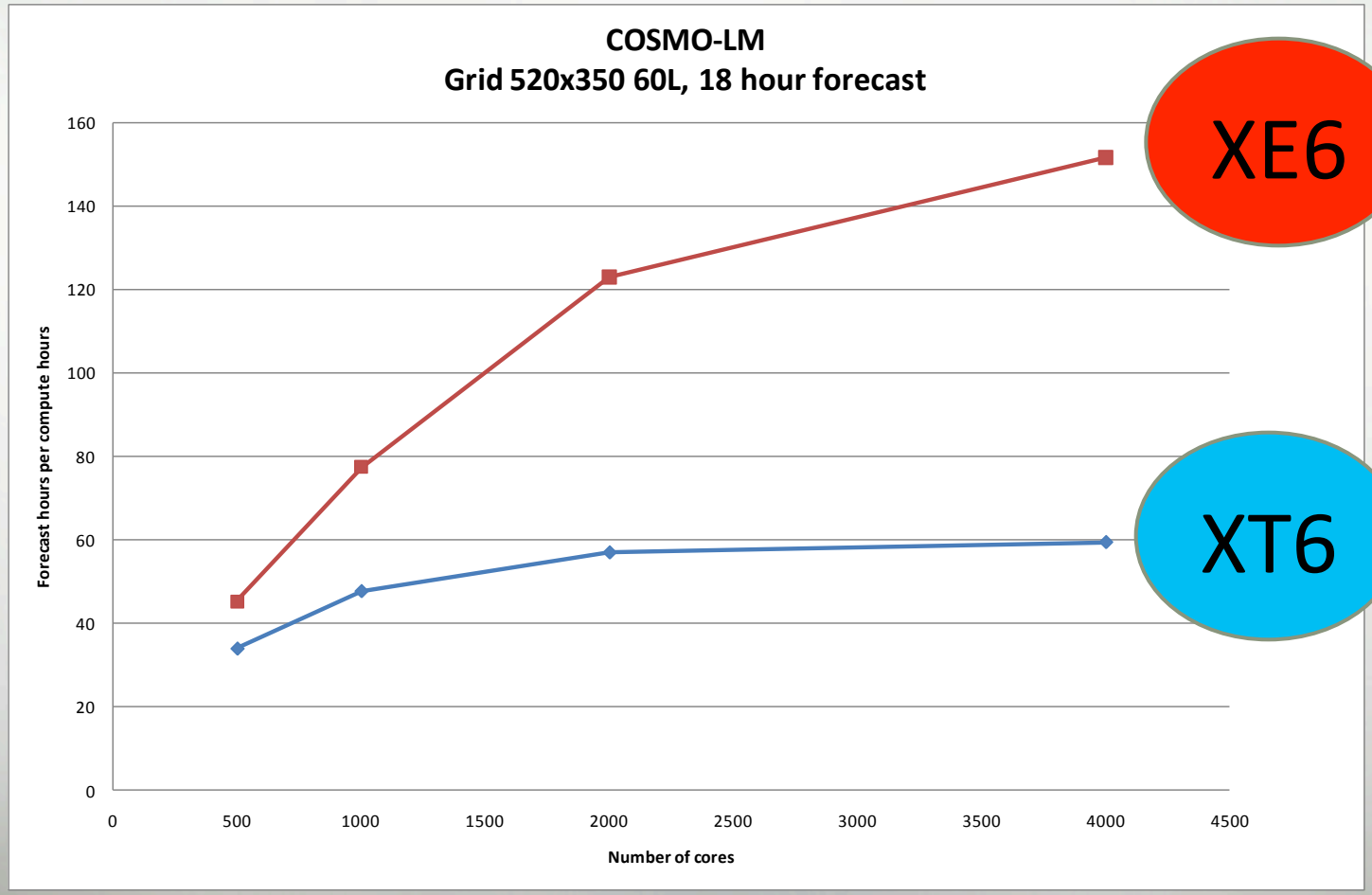
Memory footprint (all 2 all)

- With 16k processes you loose more than 75% of your memory for communication buffers and connection handling!
  (JuRoPA 24GB/node ~ 3GB/process)



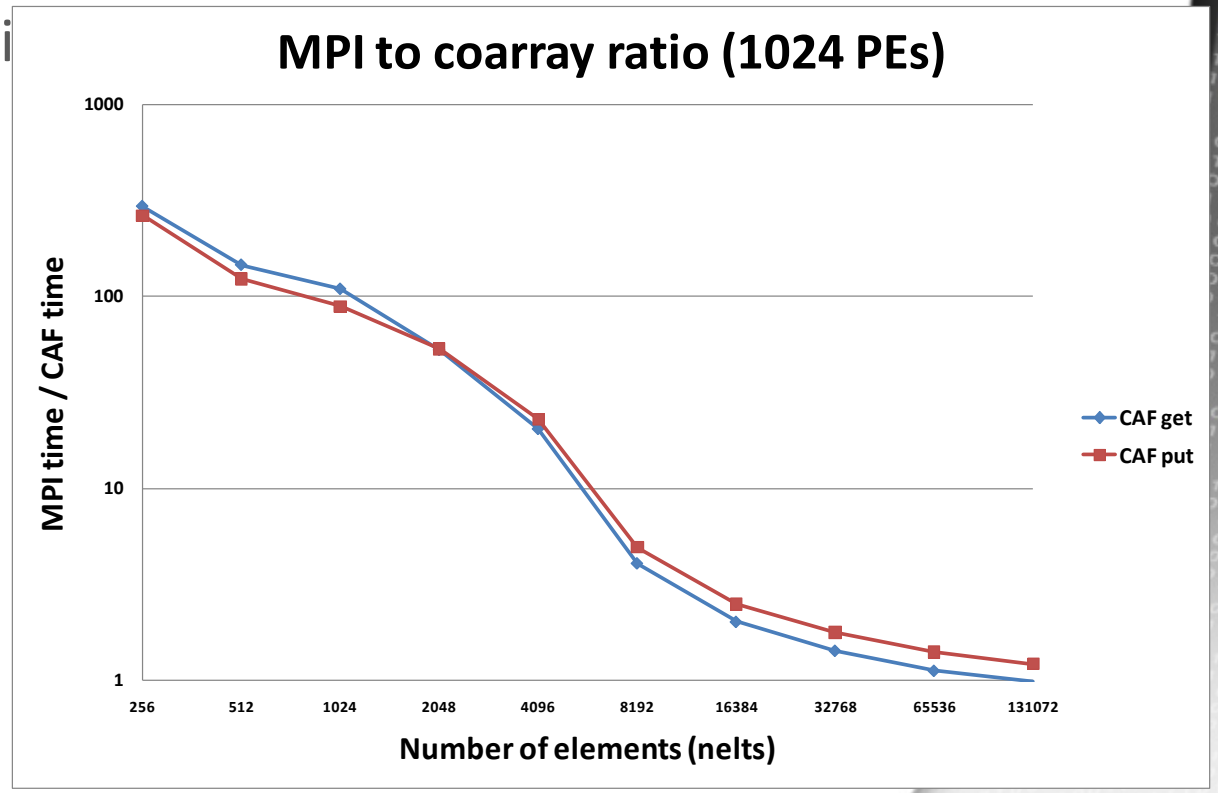
Scalability and simulation rate

- Forecast Hours per compute Hours
- Typical performance improvement



Remote gather: coarray vs MPI

- Coarray implementations are much simpler
- Coarray syntax allows the expression of remote data in a natural way – no need of complex protocols
- Coarray implementation is orders of magnitude faster for small numbers of indi



CRAY THE SUPERCOMPUTER COMPANY | epcc

Cray Software Objectives for Accelerators

1. Provide baseline accelerator environment
   - Don't preclude use of tools developers/programmers are used to
2. Integrated Programming Environment
   - Extension of PE Cray has provided on XT/XE systems
   - Provide "bundled" 3rd party commonly used or expected software (compilers, libraries, tools)
3. Cray integrated programming environment include:
   - Greatly enhance the productivity of the programming writing new applications or porting existing applications to accelerators
   - Improve performance of existing applications by exploiting greater levels of parallelism
   - Maintain source compatibility between multi-core and accelerator versions of the code

Thank you. Questions?