Multicore Architecture and Hybrid Programming

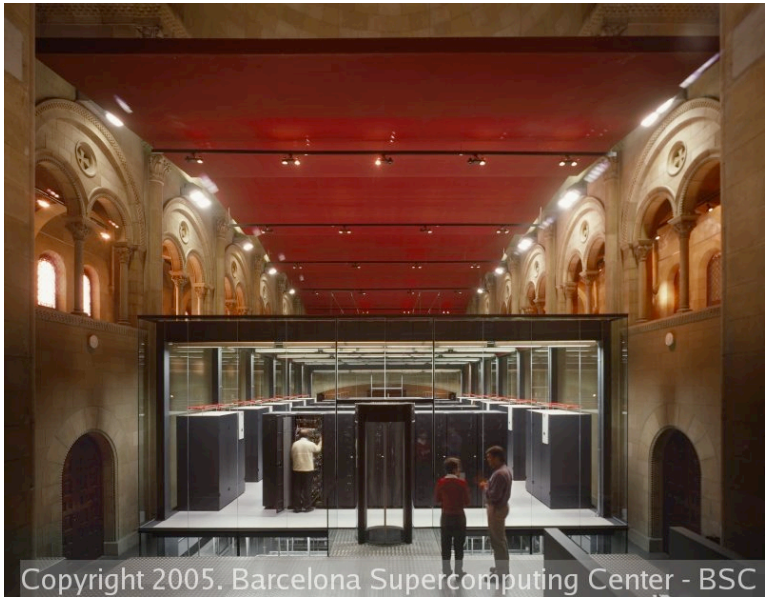
Rebecca Hartman-Baker
Oak Ridge National Laboratory
hartmanbakrj@ornl.gov

© 2004-2009 Rebecca Hartman-Baker. Reproduction permitted for non-commercial, educational use only.

Outline

I. Computer Architecture

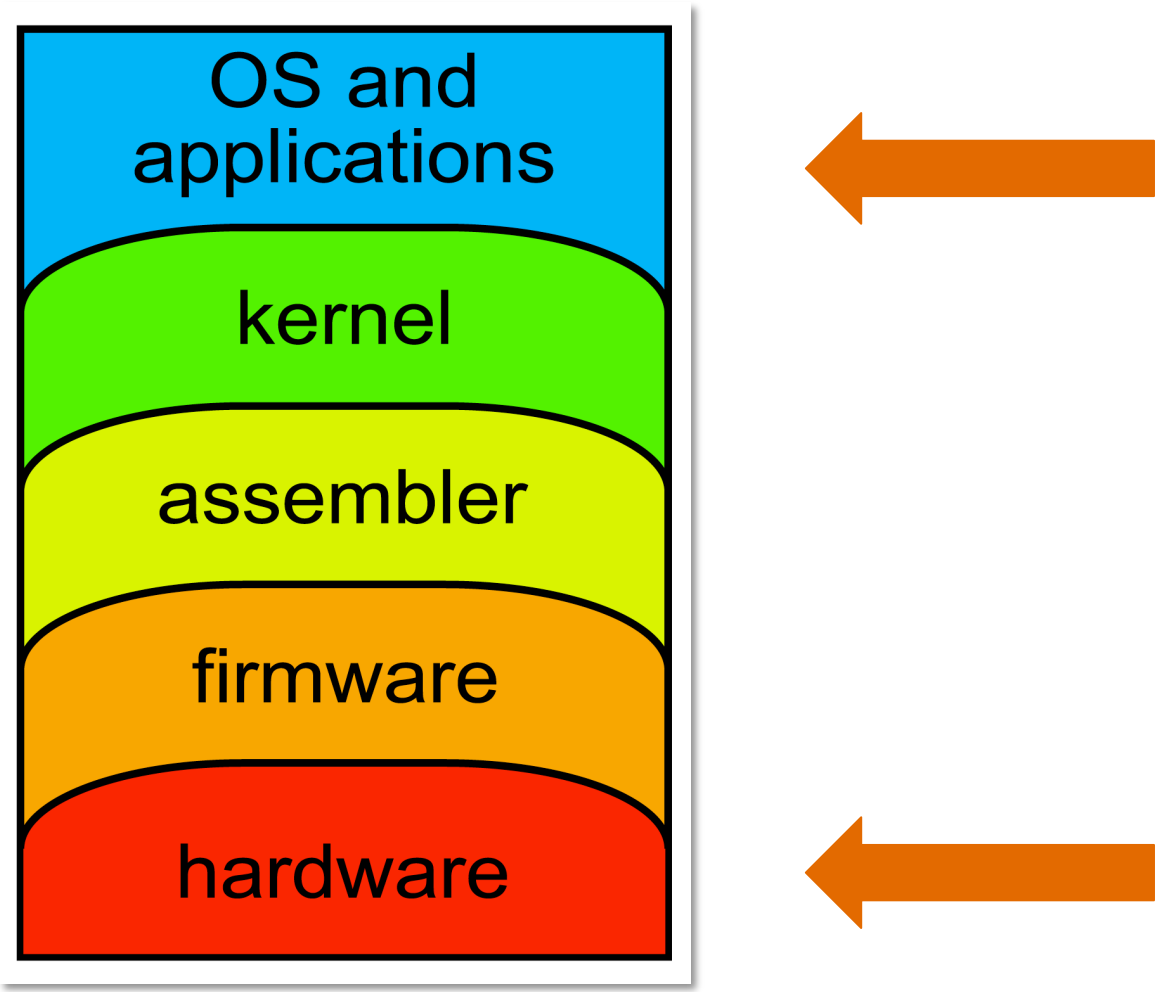
II. Hybrid Programming



I. COMPUTER ARCHITECTURE

Mare Nostrum, installed in Chapel Torre Girona, Barcelona Supercomputing Center. By courtesy of Barcelona Supercomputing Center -- http://www.bsc.es/

I. Computer Architecture 101



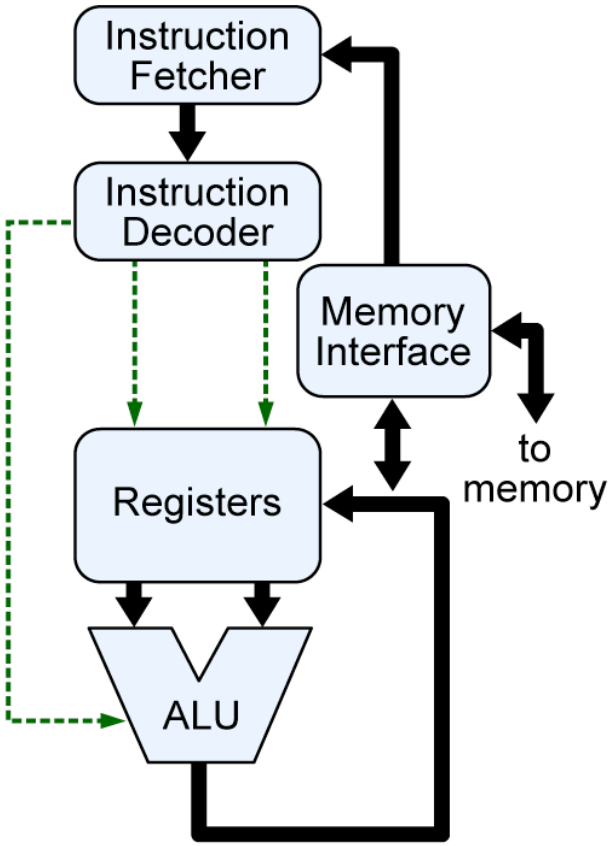
Source: http://en.wikipedia.org/wiki/Image:Computer_abstraction_layers.svg (author unknown)

Computer Architecture 101

- Processors
- Memory
  - Memory Hierarchy
  - TLB
- Interconnects
- Glossary

Computer Architecture 101: Processors

- CPU performs 4 basic operations:
  - Fetch
  - Decode
  - Execute
  - Writeback



Source: http://en.wikipedia.org/wiki/Image:CPU_block_diagram.svg

CPU Operations

- **Fetch**
  - Retrieve instruction from program memory
  - Location in memory tracked by program counter (PC)
  - Instruction retrieval sped up by caching and pipelining
- **Decode**
  - Interpret instruction by breaking into meaningful parts, e.g., opcode, operands
- **Execute**
  - Connect to portions of CPU to perform operation, e.g., connect to arithmetic logic unit (ALU) to perform addition
- **Writeback**
  - Write result of execution to memory

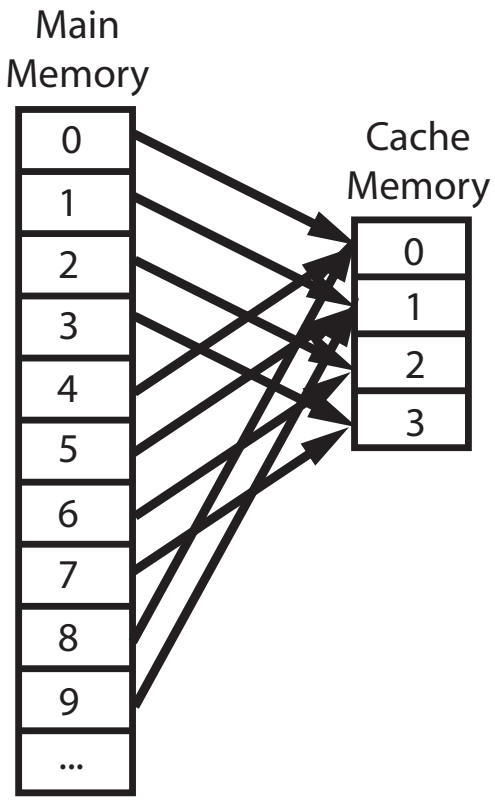
Computer Architecture 101: Memory

- Hierarchy of memory
  - Fast-access memory: small (expensive)
  - Slower-access memory: large (less expensive)
- Cache: fast-access memory where frequently used data stored
  - Reduces average access time
  - Works because typically, applications have locality of reference
  - Cache in XT4/5 also hierarchical
- TLB: Translation lookaside buffer
  - Used by memory management hardware to improve speed of virtual address translation

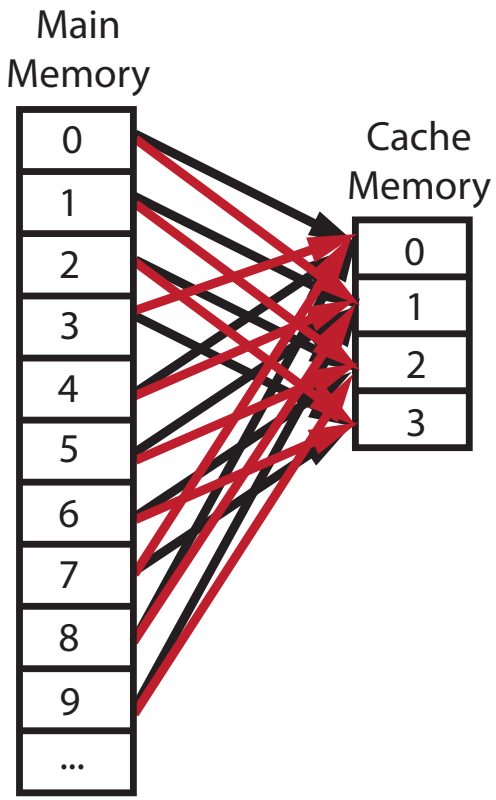
Cache Associativity

- Where to look in cache memory for copy of main memory location?
  - Direct-Mapped/ 1-way Associative: only one location in cache for each main memory location
  - Fully Associative: can be stored anywhere in cache
  - 2-way Associative: two possible locations in cache
  - *N*-way Associative: *N* possible locations in cache
- Doubling associativity (1 → 2, 2 → 4) has same effect on hit rate as doubling cache size
- Increasing beyond 4 does not substantially improve hit rate; higher associativity done for other reasons

Cache Associativity: Illustration



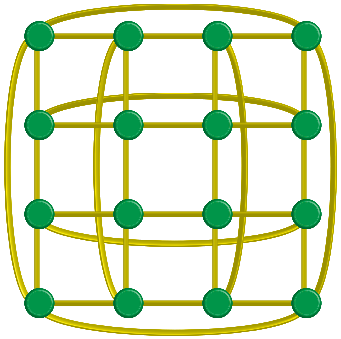
Direct-Mapped Cache



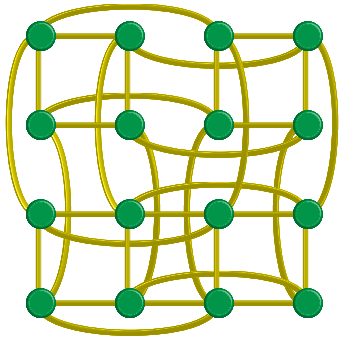
2-Way Associative Cache

Computer Architecture 101: Interconnects

- Connect nodes of machine to one another
- Methods of interconnecting
  - Fiber + switches and routers
  - Directly connecting
- Topology
  - Torus
  - Hypercube
  - Butterfly
  - Tree



2-D Torus



Hypercube

Computer Architecture 101: Glossary

- *SSE (Streaming SIMD Extensions):* instruction set extension to x86 architecture, allowing CPU to work on multiple instructions in single clock cycle
- *DDR2 (Double Data Rate 2):* synchronous dynamic random access memory, operates twice as fast as DDR1
  - DDR2-*xyz*: performs *xyz* million data transfers/second
- *Dcache:* cache devoted to data storage
- *Icache:* cache devoted to instruction storage
- *STREAM:* data flow

NCCS Facts and Figures

	Jaguar	Jaguarpf	Eugene
Compute Nodes	7832	18,772	2048
Processor	2.1 GHz AMD Opteron Quad Core	2.3 GHz AMD Opteron Dual Quad-Core	850MHz IBM quad core 450d PowerPC
Memory	2 GB/core DDR2-667/ DDR2-800	2 GB/core DDR2-800	2 GB/node
Network	Cray SeaStar 2	Cray SeaStar 2	3-D torus, 5.1 Gb/s
Peak	263 TF	1.3 PF	27 TF

XT4/5 Architecture

- Hardware
  - Processors
  - Memory
    - Memory Hierarchy
    - TLB
  - System architecture
  - Interconnects
- Software
  - Operating System Integration
  - CNL vs Linux

Quad-Core Architecture

Introducing "Barcelona"…

AMD
Smarter Choice

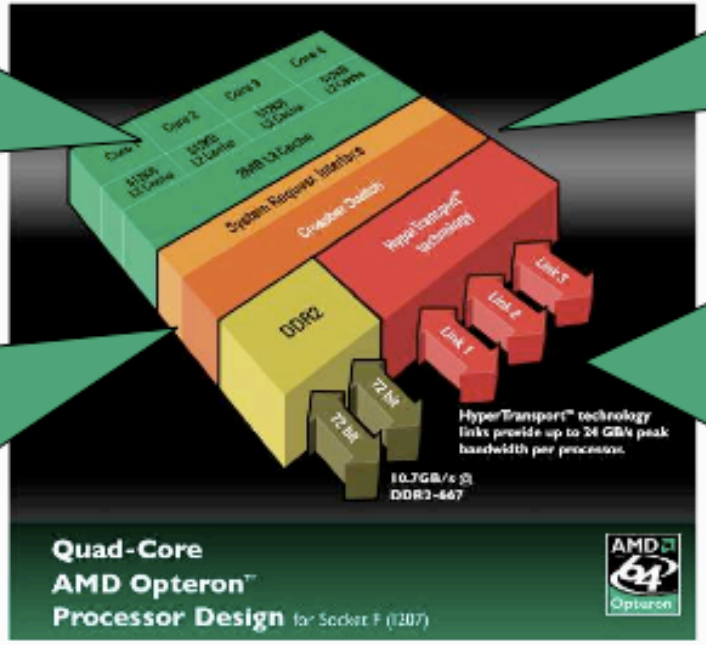
Native quad-core, 3rd Gen. AMD Opteron

Native Quad-Core Processor

To increase performance-per-watt efficiencies **using the same Thermal Design Power.**

Platform Compatibility

Socket and thermal compatible with "Socket F".



Advanced Process Technology

65nm Silicon-on Insulator Process

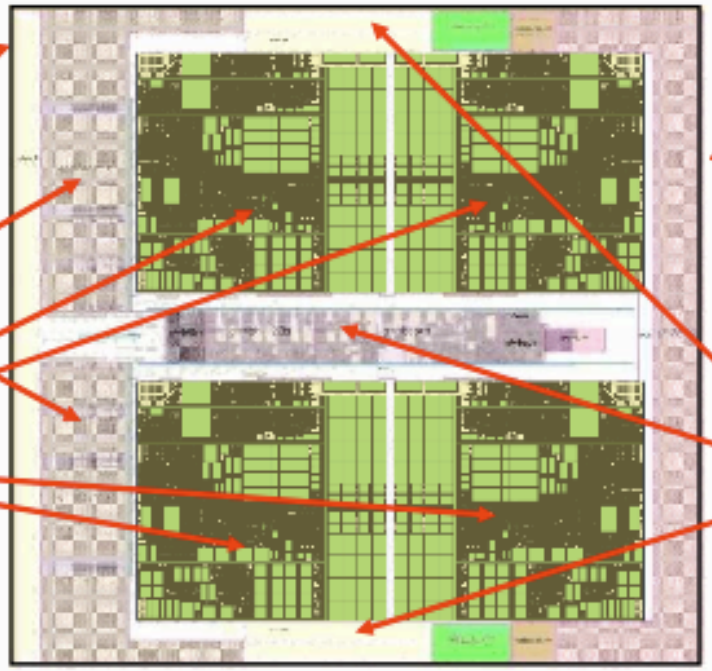
Fast transistors with low power leakage to **reduce power and heat.**

Direct Connect Architecture

- Integrated memory controller designed for **reduced memory latency and increased performance**
  - Memory directly connected
- Provides **fast CPU-to-CPU communication**
  - CPUs directly connected
- **Glueless SMP** up to 8 sockets

AMD's Next Generation Processor Technology

AMD
Smarter Choice



Native quad core die

Optimized for 65nm SOI and beyond

Expandable shared L3 cache

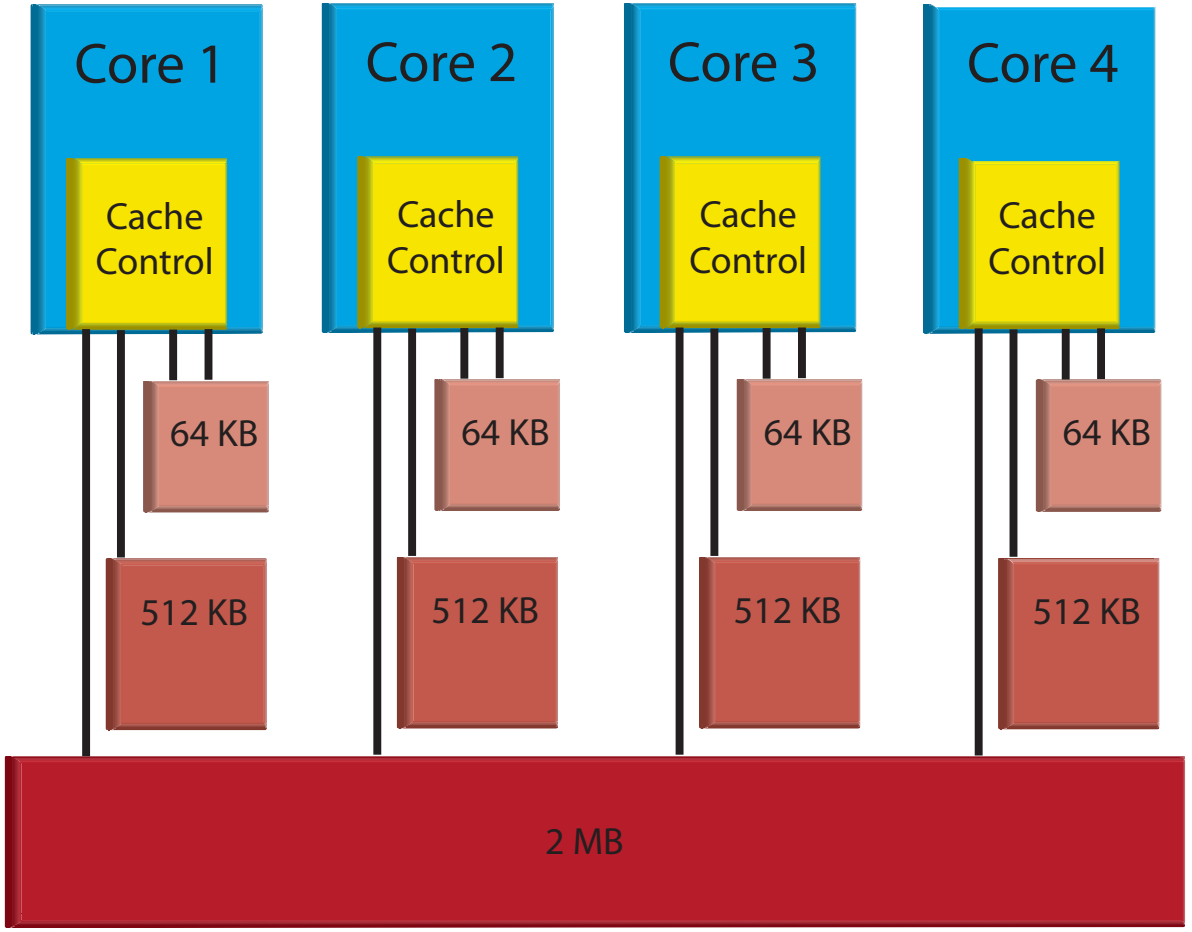
IPC enhanced CPU cores

- 32B instruction fetch
- Improved branch prediction
- Out-of-order load execution
- Up to 4 DP FLOPS/cycle
- Dual 128-bit SSE dataflow
- Dual 128-bit loads per cycle
- Instruction Extensions
- Power Management Extensions

Enhanced Direct Connect Architecture and Northbridge

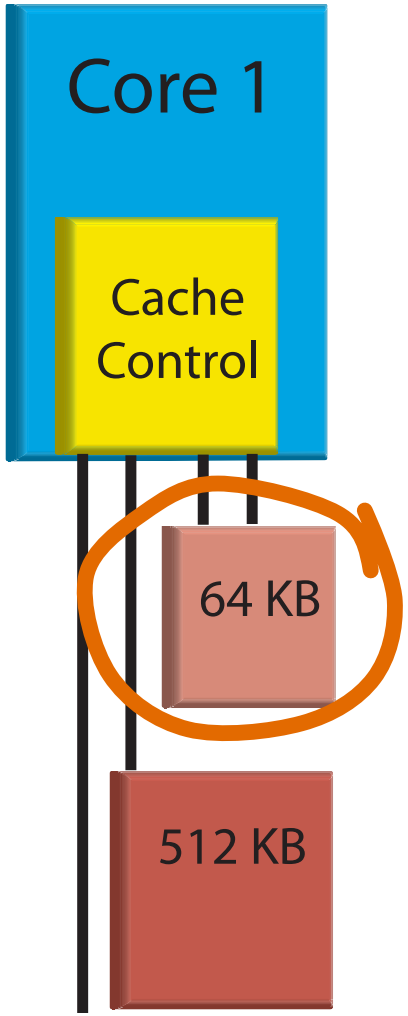
- Enhanced HyperTransport™ links
- Enhanced crossbar
- DDR2 with migration path to DDR3
- Enhanced power management and RAS

Quad Core Cache Hierarchy



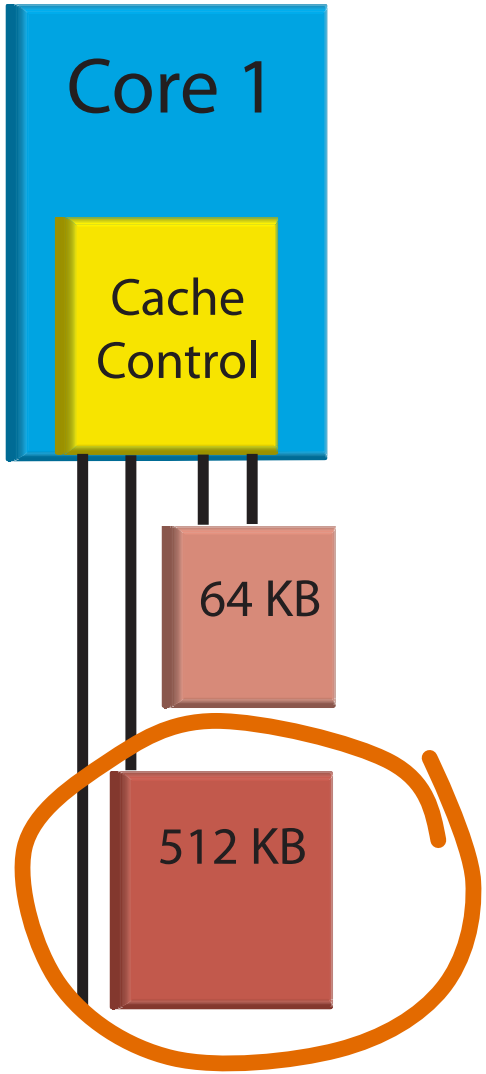
L1 Cache

- Dedicated
- 2-way associativity
- 8 banks
- 2 x 128-bit loads/cycle



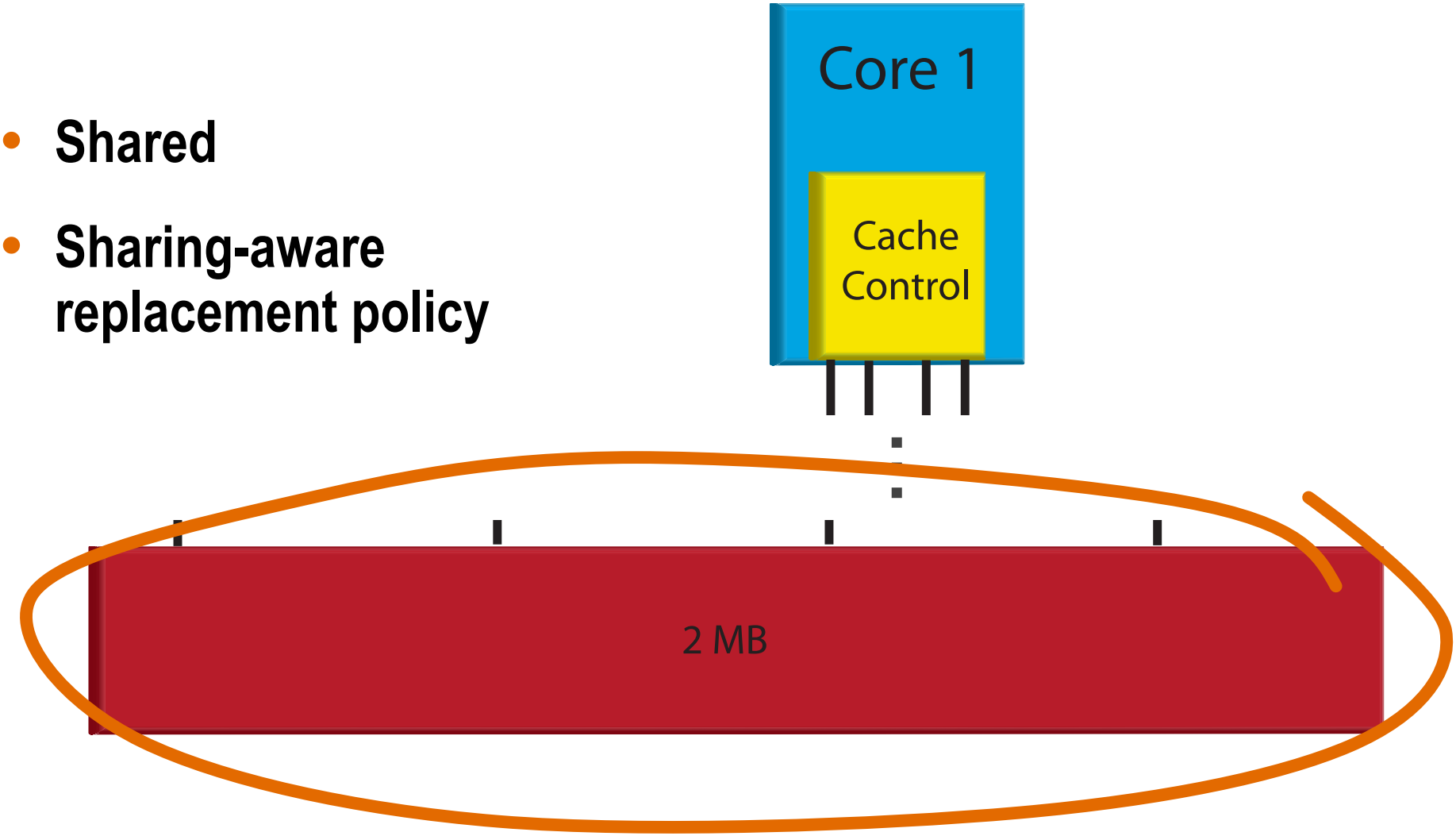
L2 Cache

- Dedicated
- 16-way associativity

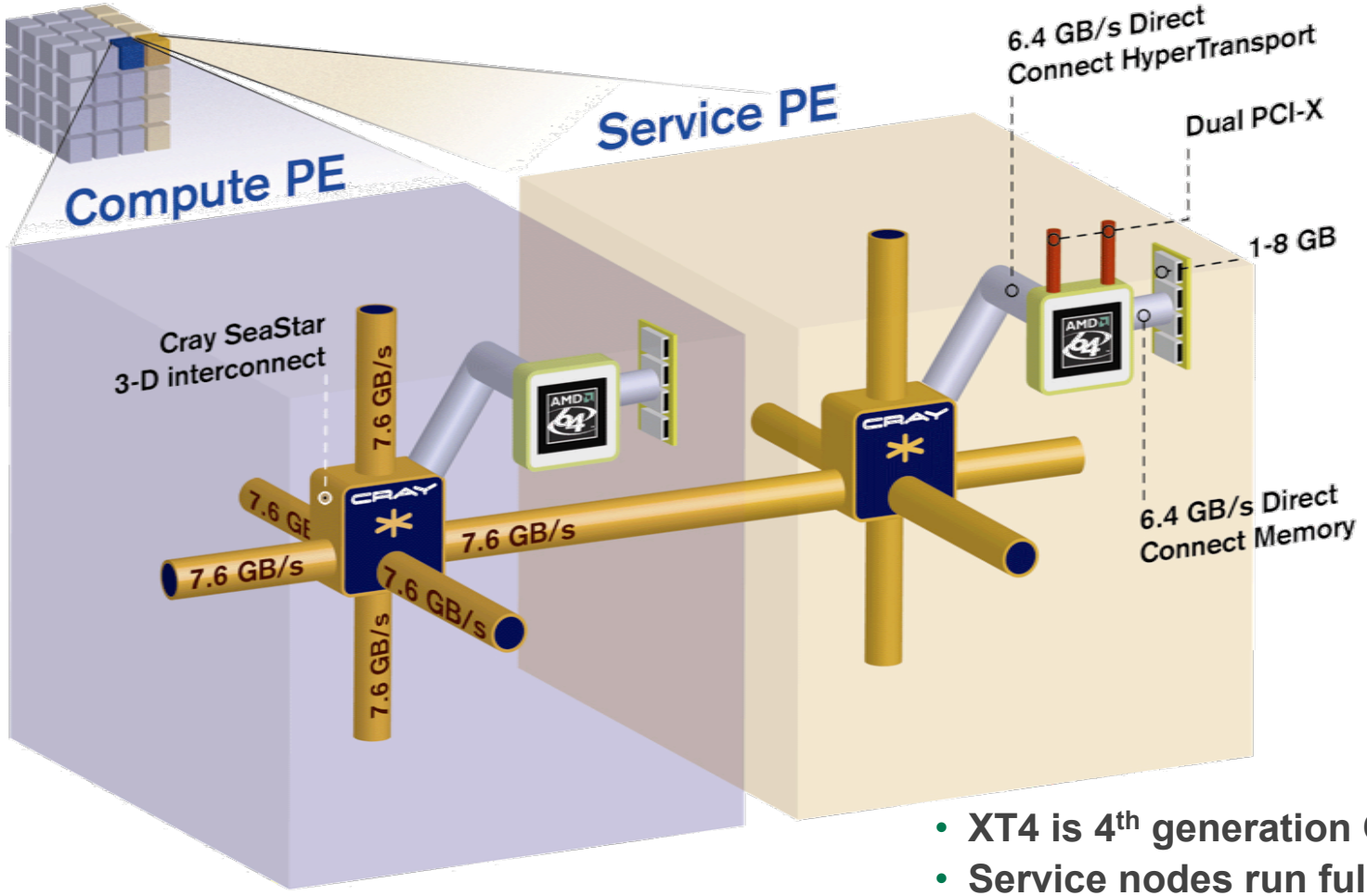


L3 Cache

- **Shared**
- **Sharing-aware replacement policy**



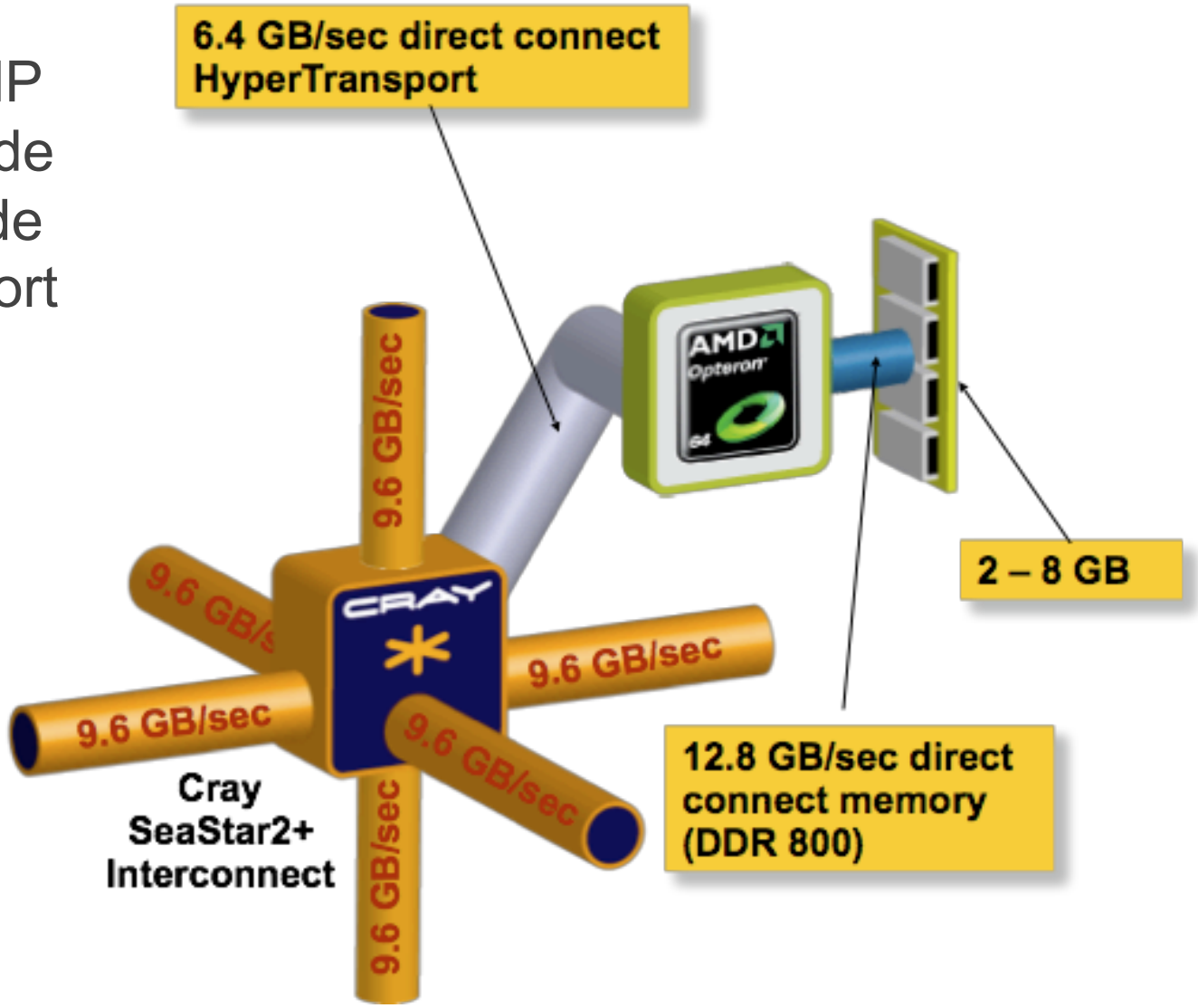
Cray XT4 Architecture



- XT4 is 4th generation Cray MPP
- Service nodes run full Linux
- Compute nodes run Compute Node Linux (CNL)

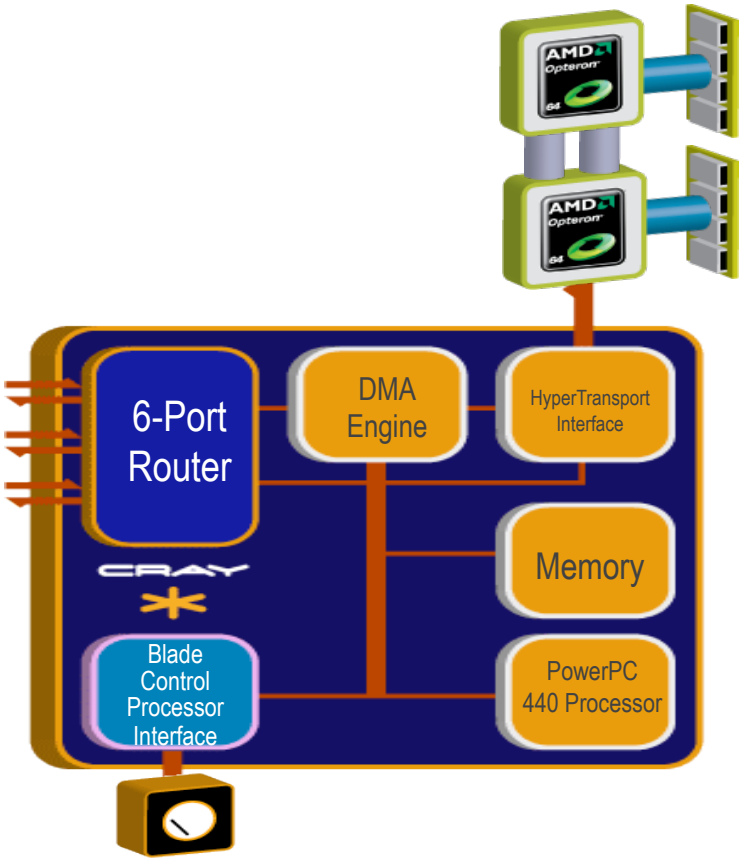
Cray XT4 Architecture

- 2- or 4-way SMP
- > 35 Gflops/node
- Up to 8 GB/node
- OpenMP Support within Socket



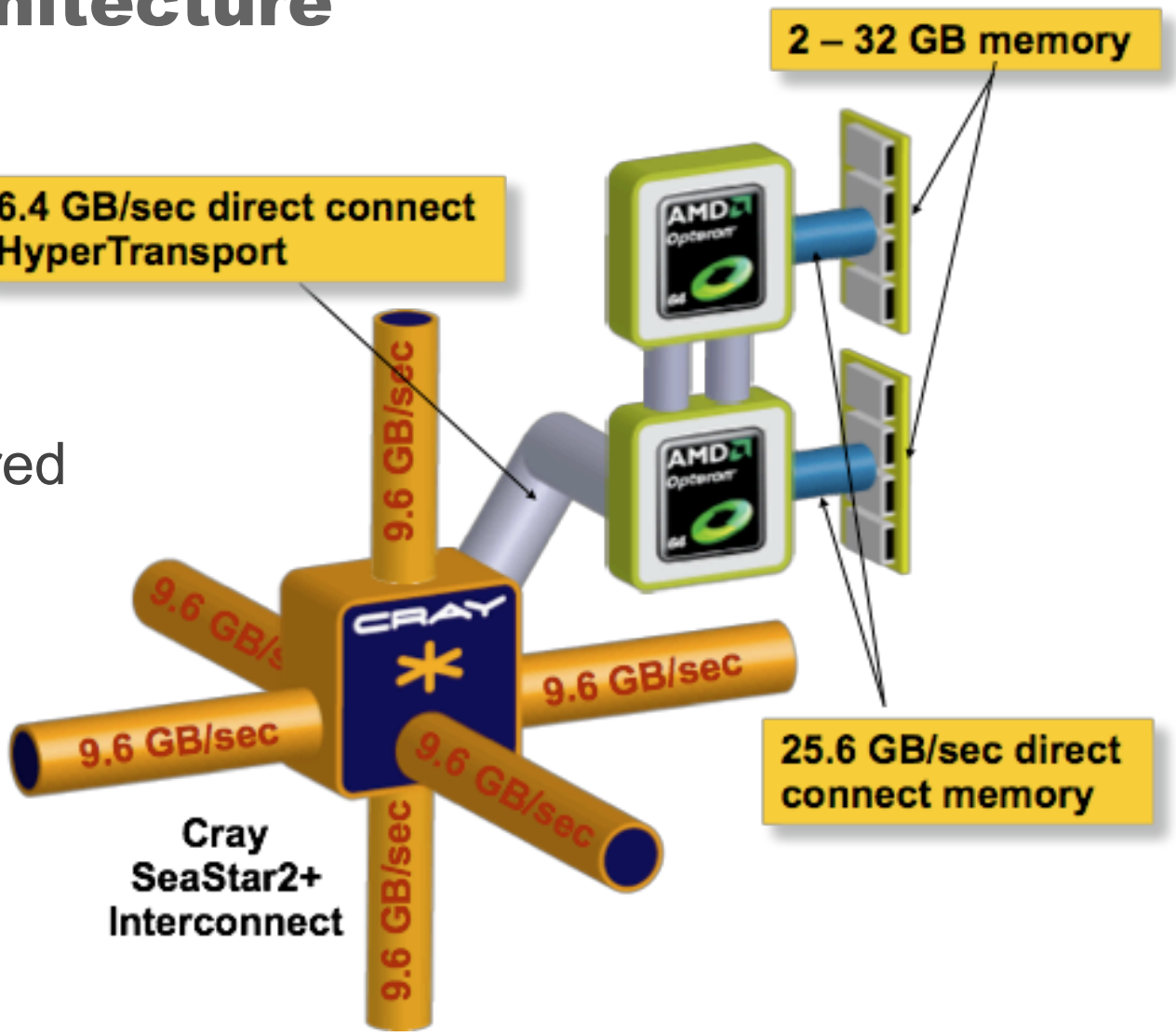
Cray SeaStar2 Architecture

- Router connects to 6 neighbors in 3-D torus
  - Peak bidirectional BW 7.6 GB/s; sustained 6 GB/s
  - Reliable link protocol with error correction and retransmission
- Communications Engine: DMA Engine + PPC 440
  - Together, perform messaging tasks so AMD processor can focus on computing
- DMA Engine and OS together minimize latency with path directly from app to communication hardware (without traps and interrupts)

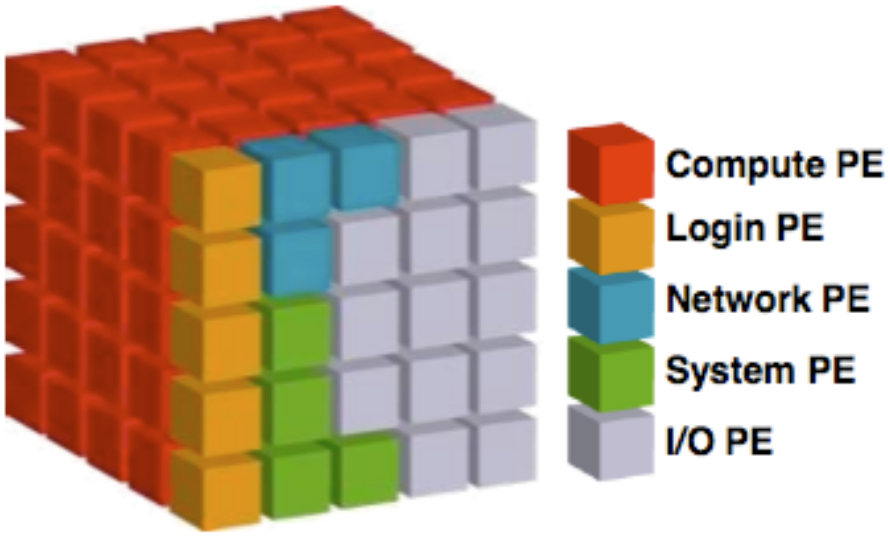


Cray XT5 Architecture

- 8-way SMP
- > 70 Gflops/node
- Up to 32 GB shared memory/node
- OpenMP support



Software Architecture



Service Partition

Specialized Linux nodes

- CNL microkernel on compute nodes
- Full-featured Linux on service nodes
- Software architecture eliminates jitter and enables reproducible runtimes
- Even large machines can reboot in < 30 mins, including filesystem

Software Architecture

- *Compute PE (processing element):* used for computation only; users cannot directly access compute nodes
- *Service PEs:* run full Linux
  - *Login:* users access these nodes to develop code and submit jobs, function like normal Linux box
  - *Network:* provide high-speed connectivity with other systems
  - *System:* run global system services such as system database
  - *I/O:* provide connectivity to GPFS (global parallel file system)

CNL vs Linux

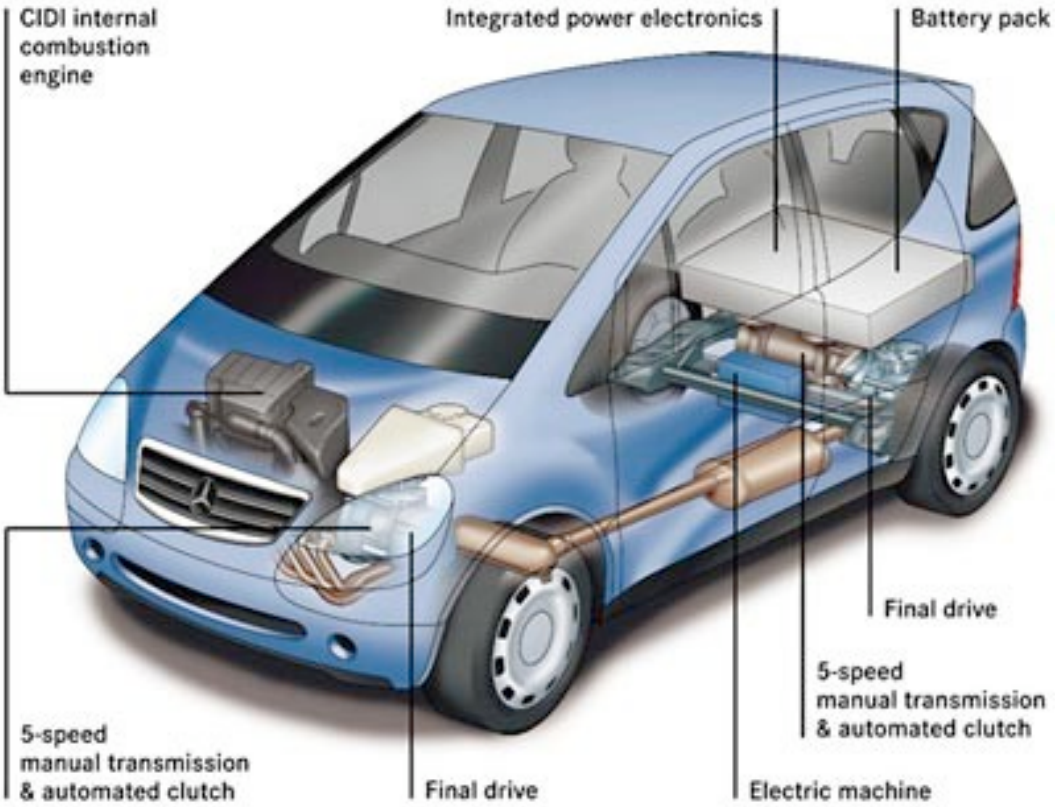
- CNL (Compute-Node Linux) contains subset of Linux features
- Minimizes system overhead because little between application and bare hardware

Resources: Computer Architecture 101

- Wikipedia articles on computer architecture: http://en.wikipedia.org/wiki/Computer_architecture , http://en.wikipedia.org/wiki/CPU , http://en.wikipedia.org/wiki/CPU_cache , http://en.wikipedia.org/wiki/DDR2_SDRAM , http://en.wikipedia.org/wiki/Microarchitecture , http://en.wikipedia.org/wiki/SSE2 , http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions
- Heath, Michael T. (2007) *Notes for CS 554, Parallel Numerical Algorithms*, http://www.cse.uiuc.edu/courses/cs554/notes/index.html

Resources: Cray XT4 Architecture

- NCCS machines
  - Jaguar: http://www.nccs.gov/computing-resources/jaguar/
  - Eugene: http://www.nccs.gov/computing-resources/eugene/
  - Jaguarpf: http://www.nccs.gov/jaguar/
- AMD architecture
  - Waldecker, Brian (2008) *Quad Core AMD Opteron Processor Overview*, available at http://www.nccs.gov/wp-content/uploads/2008/04/amd_craywkshp_apr2008.pdf
  - Larkin, Jeff (2008) *Optimizations for the AMD Core*, available at http://www.nccs.gov/wp-content/uploads/2008/04/optimization1.pdf
- XT4 Architecture
  - Hartman-Baker, Rebecca (2008*) XT4 Architecture and Software*, available at http://www.nccs.gov/wp-content/training/2008_users_meeting/4-17-08/using-xt44-17-08.pdf



II. HYBRID PROGRAMMING

Hybrid Car. Source: http://static.howstuffworks.com/gif/hybrid-car-hyper.jpg

II. Hybrid Programming

- Motivation
- Considerations
- MPI threading support
- Designing hybrid algorithms
- Examples

Motivation

- Multicore architectures are here to stay
- Macro scale: distributed memory architecture, suitable for MPI
- Micro scale: each node contains multiple cores and shared memory, suitable for OpenMP
- Obvious solution: use MPI between nodes, and OpenMP within nodes
- Hybrid programming model

Considerations

- Sounds great, Rebecca, but is hybrid programming always better?
  - No, not always
  - Especially if poorly programmed ☺
  - Depends also on suitability of architecture
- Think of accelerator model
  - in omp parallel region, use power of multicores; in serial region, use only 1 processor
  - If your code can exploit threaded parallelism “a lot”, then try hybrid programming

Considerations

- Hybrid parallel programming model
  - Are communication and computation discrete phases of algorithm?
  - Can/do communication and computation overlap?
- Communication between threads
  - Communicate only outside of parallel regions
  - Assign a manager thread responsible for inter-process communication
  - Let some threads perform inter-process communication
  - Let all threads communicate with other processes

MPI Threading Support

- MPI-2 standard defines four threading support levels
  - (0) `MPI_THREAD_SINGLE` only one thread allowed
  - (1) `MPI_THREAD_FUNNELED` master thread is only thread permitted to make MPI calls
  - (2) `MPI_THREAD_SERIALIZED` all threads can make MPI calls, but only one at a time
  - (3) `MPI_THREAD_MULTIPLE` no restrictions
  - (0.5) MPI calls not permitted inside parallel regions (returns `MPI_THREAD_SINGLE`) – this is MPI-1

What Threading Model Does My Machine Support?

```
#include <mpi.h>
#include <stdio.h>
int main(int *argc, char **argv) {

MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE,
  &provided);

printf("Supports level %d of %d %d %d %d\n",
provided,
MPI_THREAD_SINGLE,
MPI_THREAD_FUNNELED,
MPI_THREAD_SERIALIZED,
MPI_THREAD_MULTIPLE);

MPI_Finalize();
return 0;
}
```

MPI_Init_Thread

- **`MPI_Init_thread(int required, int *supported)`**
  - Use this instead of **`MPI_Init(…)`**
  - **`required`**: the level of thread support you want
  - **`supported`**: the level of thread support provided by implementation (hopefully **`= required`**, but if not available, returns lowest level **`> required`**; failing that, largest level **`< required`**)
  - Using **`MPI_Init(…)`** is equivalent to **`required = MPI_THREAD_SINGLE`**
- **`MPI_Finalize()`** should be called by same thread that called **`MPI_Init_thread(…)`**

Other Useful MPI Functions

- `MPI_Is_thread_main(int *flag)`
  - Thread calls this to determine whether it is main thread
- `MPI_Query_thread(int *provided)`
  - Thread calls to query level of thread support

Supported Threading Models: Single

- Use single pragma

```
#pragma omp parallel
{
#pragma omp barrier
#pragma omp single
{
  MPI_Xyz(…)
}
#pragma omp barrier
}
```

```
!$OMP parallel
!$OMP barrier
!$OMP single
call MPI_Xyz(…,ierr)
!$OMP end single
!$OMP barrier
!$OMP end parallel
```

Supported Threading Models: Funneling

- XT4 supports funneling (probably Ranger too?)
- Use master pragma

```
#pragma omp parallel
{
#pragma omp barrier
#pragma omp master
{
  MPI_Xyz(…)
}
#pragma omp barrier
}
```

```
!$OMP parallel
!$OMP barrier
!$OMP master
call MPI_Xyz(…,ierr)
!$OMP end master
!$OMP barrier
!$OMP end parallel
```

What Threading Model Should I Use?

- **Depends on the application!**

Model	Advantages	Disadvantages
Single	Portable: every MPI implementation supports this	Limited flexibility
Funneled	Simpler to program	Manager thread could get overloaded
Serialized	Freedom to communicate	Risk of too much cross-communication
Multiple	Completely thread safe	Limited availability

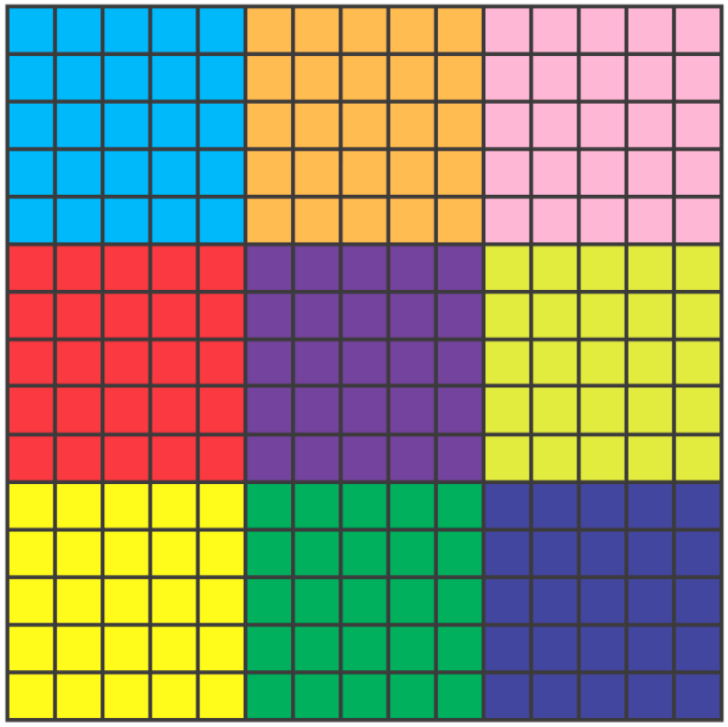
Designing Hybrid Algorithms

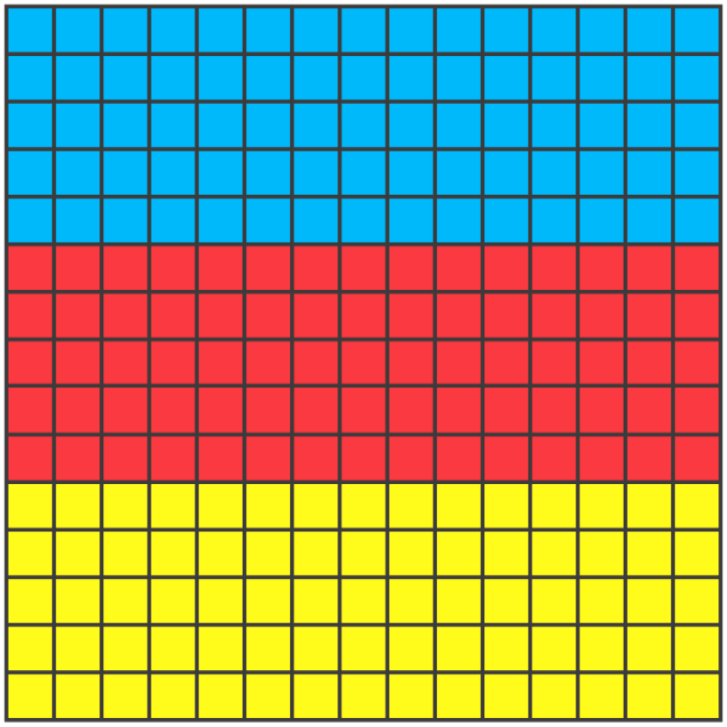
- Just because you *can* communicate thread-to-thread, doesn't mean you *should*
- Tradeoff between lumping messages together and sending individual messages
  - Lumping messages together: one big message, one overhead
  - Sending individual messages: less wait time (?)
- Programmability: performance will be great, when you finally get it working!

Example: Mesh Partitioning

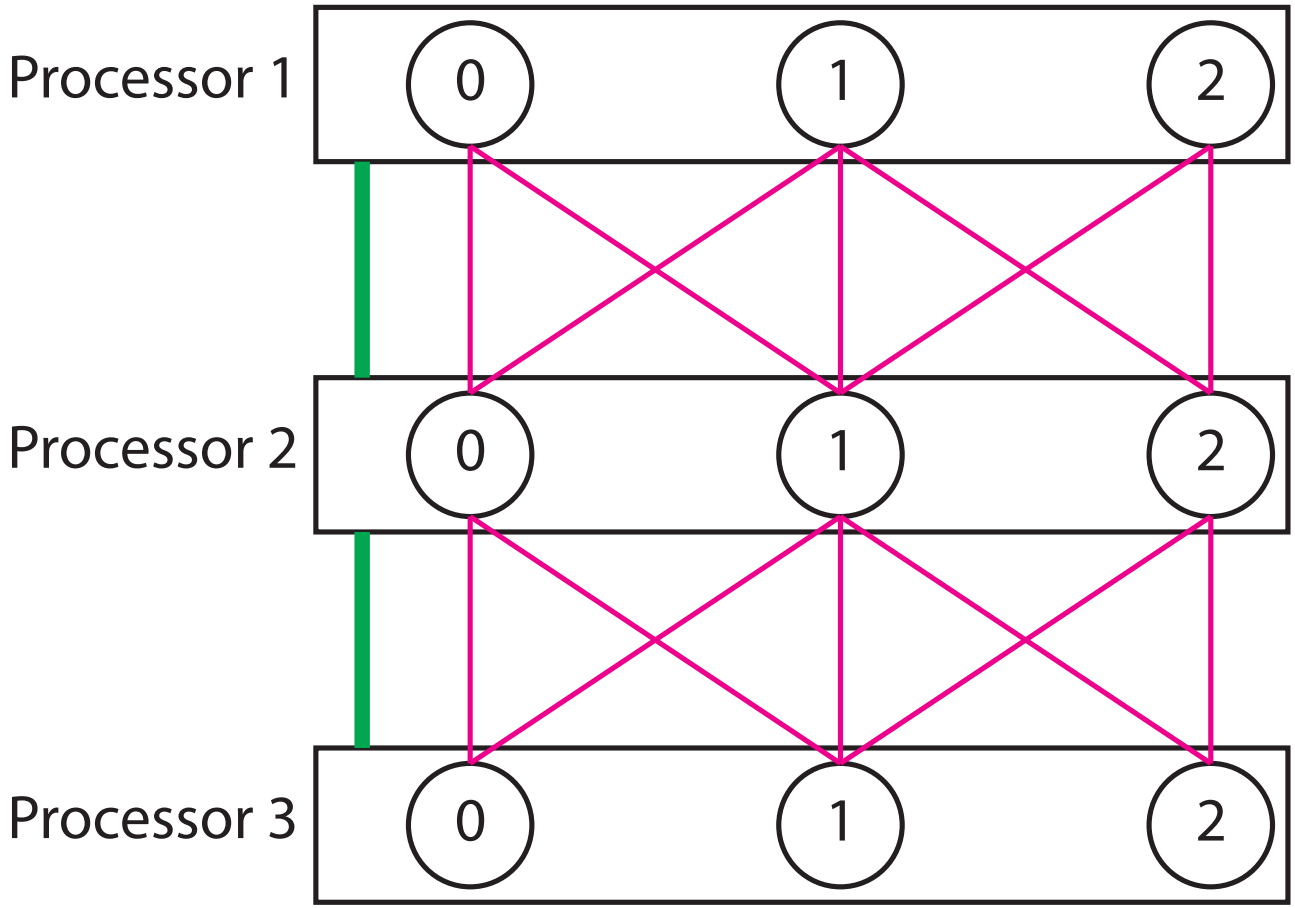
- Regular mesh of finite elements
- When we partition mesh, need to communicate information about (domain) adjacent cells to (computationally) remote neighbors

Example: Mesh Partitioning





Example: Mesh Partitioning Communication Patterns



Bibliography/Resources

- von Alfthan, Sebastian, *Introduction to Hybrid Programming*, PRACE Summer School 2008, URL:http://www.prace-project.eu/hpc-training/prace-summer-school/hybridprogramming.pdf
- Ye, Helen and Chris Ding, Hybrid OpenMP and MPI Programming and Tuning, Lawrence Berkeley National Laboratory, http://www.nersc.gov/nusers/services/training/classes/NUG/Jun04/NUG2004_yhe_hybrid.ppt
- Zollweg, John, Hybrid Programming with OpenMP and MPI, Cornell University Center for Advanced Computing, http://www.cac.cornell.edu/education/Training/Intro/Hybrid-090529.pdf
- MPI-2.0 Standard, Section 8.7, "MPI and Threads," http://www.mpi-forum.org/docs/mpi-20-html/node162.htm#Node162