

# Lecture 16

## The future of High Performance Computing

# Announcements

- Abe will be down for part of the day on Monday 12/1

# CSE 260 Symposium and Report

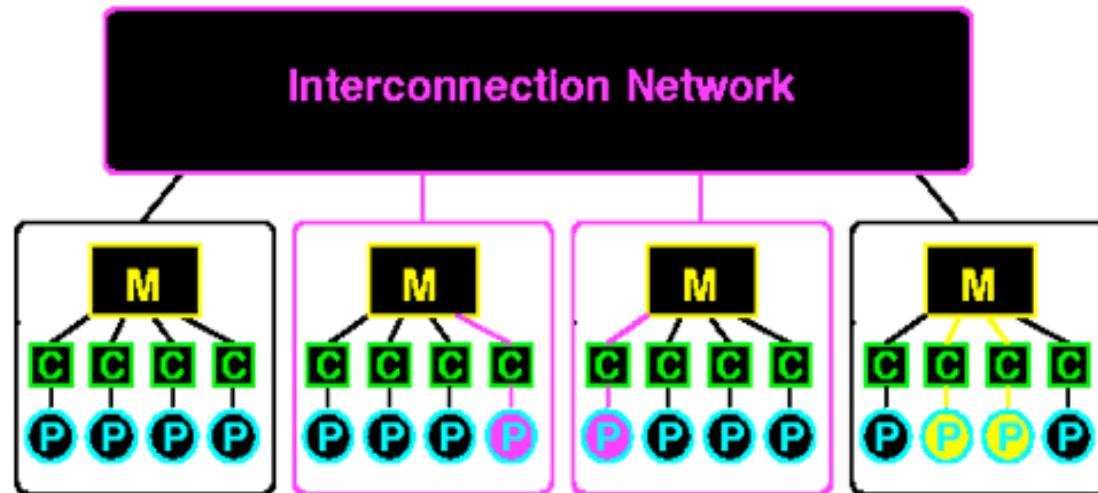
- Meet over 2 days
  - Thursday 12/4, 2:00 to 3:20 PM
  - Friday 12/5, 1:00 to 4:00 PM (Room TBA)
- Format
  - 15 minute presentation + 5 minutes for questions
- Post a link to your slides on the web board
- Report
  - Due at 5pm
  - Email me your electronic turnin with code and experimental data

# Today's lecture

- Programming: Latency tolerance
- Trends in System Architecture
  - Massively parallel processing
  - Hybrid systems

# Tolerating communication delays

- Technological trends continue to increase the cost of communication relative to computation
- One way to reduce communication costs is to overlap communication with computation
- Another is to reformulate the algorithm, replacing expensive communication with inexpensive computation



## Motivating application

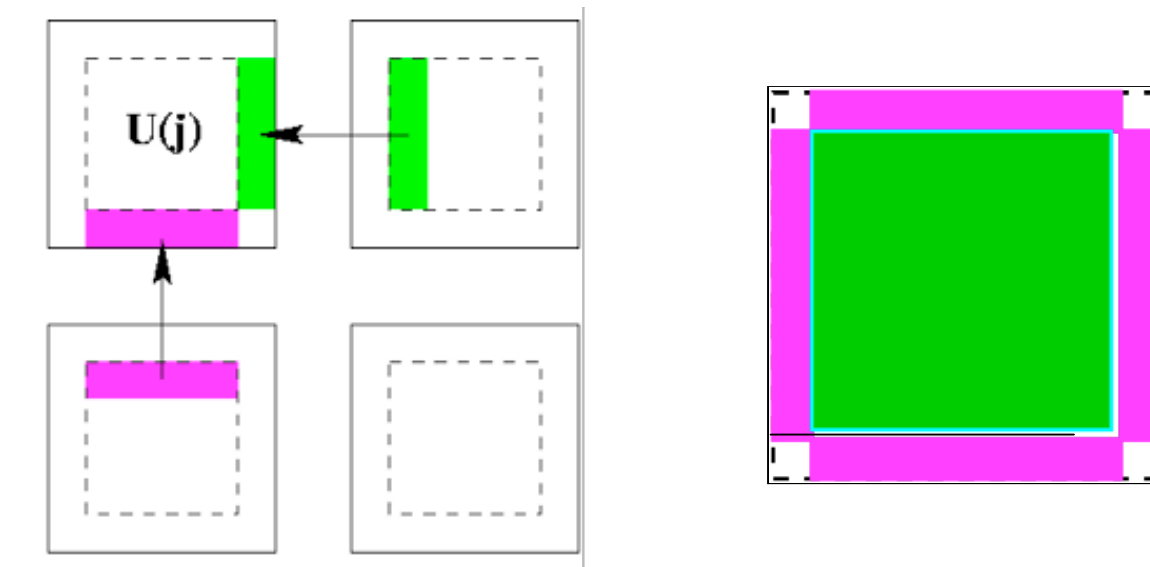
- Iterative finite difference solver for Poisson's equation in 3 dimensions
- Red black ordering

for (i,j,k) in 1:N x 1:N x 1:N  
where (i+j+k) is odd/even

$$u[i][j][k] = (u[i-1][j][k] + u[i+1][j][k] + \\ u[i][j-1][k] + u[i][j+1][k] + \\ u[i][j][k+1] + u[i][j][k-1])/6;$$

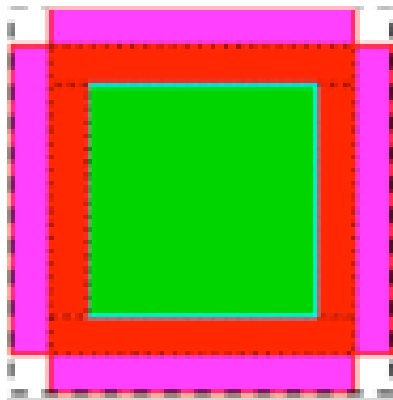
# Synchronous variant

- Decompose the domain into 1 region per process
  - Transmit **halo regions** between processes
  - Compute **inner region** after communication completes



# Overlap strategy

- Isolate the **inner region** from the **halo**
- Defer computation on the **annulus**
- Execute **communication** concurrently with computation on the **inner region**
- Compute on the **annulus** when the halo finishes



## A few implementation details

- Some versions of MPI can realize overlap with MPI\_IRecv and MPI\_Isend
- If not, then we can use multithreading to handle the overlap
- We let one or more processors handle communication

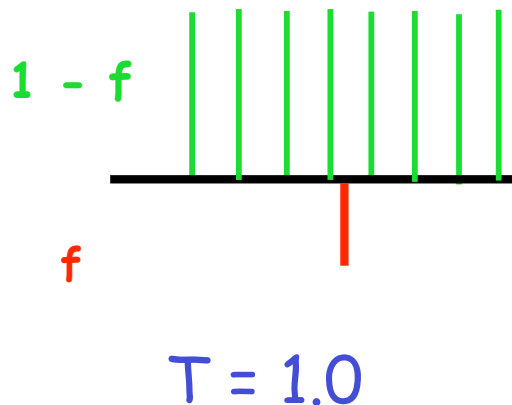
# A performance model of overlap

- Assumptions

$p$  = number of processors per node

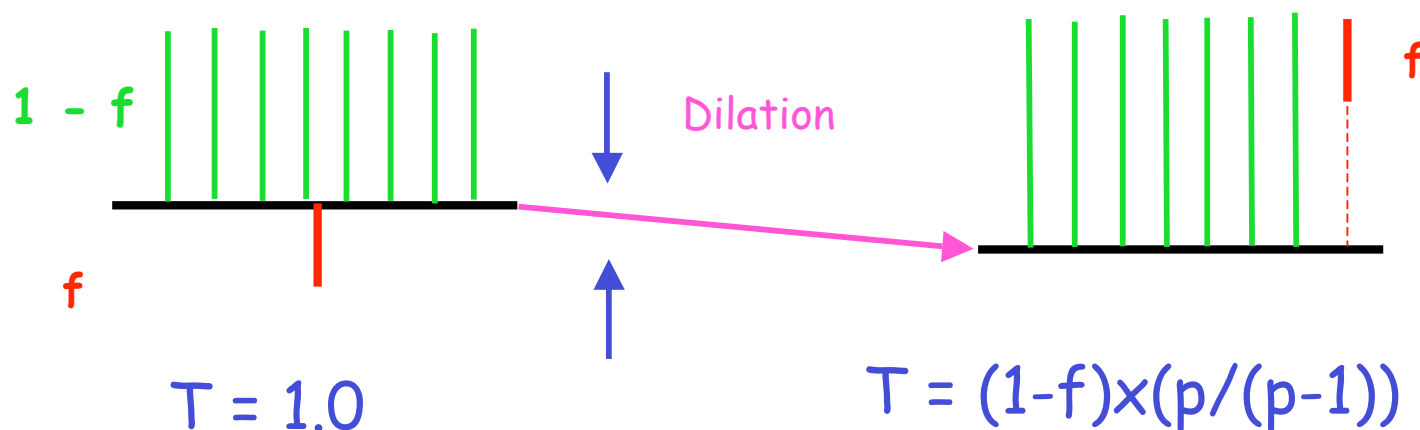
running time = 1.0

$f < 1$  = communication time with MT(k)  
(i.e. not overlapped)



# Performance

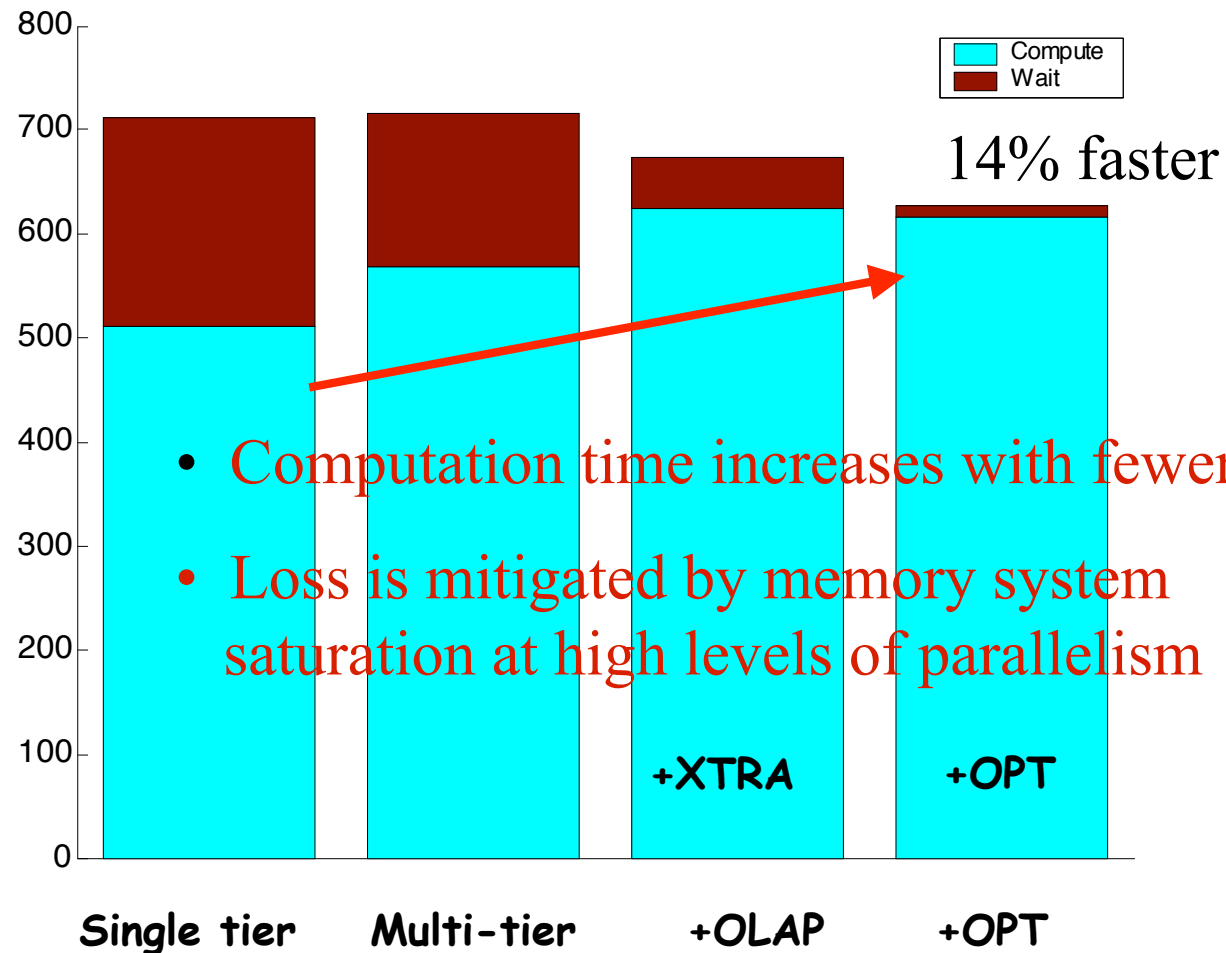
- When we displace computation to make way for the proxy, computation time *increases*
- Wait on communication drops to zero, ideally
- When  $f < p/(2p-1)$ : improvement is  $(1-f) \times (p/(p-1))^{-1}$
- Communication bound: improvement is  $1/(1-f)$



# NPACI Blue Horizon

- Multiple SMP nodes
  - 144 8-way Power3+ “high” nodes
  - 375 MHz CPU
  - 4 GB memory per node
  - 64 KB L1\$, 4MB L2\$ per processor
  - Caches have a 128 Byte line size
- Differential MPI communication rates  
(peak Ring)
  - 400 MB/sec off-node
  - 500 MB/sec on node

# Performance improves with overlap



- Computation time increases with fewer CPUs
- Loss is mitigated by memory system saturation at high levels of parallelism

# Where we are so far

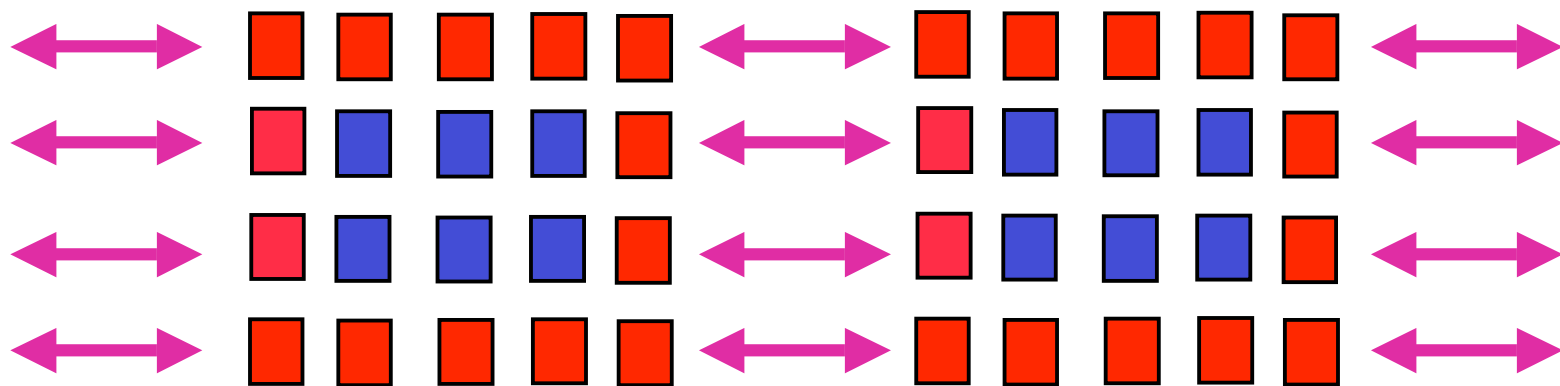
- The synchronous (SY) and an overlapped (OV) variants differ in two fundamental ways
  - SY employs a single level of parallel control flow and it exhibits distinct computation and communication phases
  - OV has two levels of control flow and it communicates asynchronously to tolerate latency. Elaborate partitioning required
- Scheduling is messy, and each application has its own overlap structure

# Non-SPMD programming

- Asynchronous task graph model of execution
  - Communication and computation do not execute as distinct phases but are coupled activities
  - Tolerate unpredictable or irregular task and communication latencies
- 2 projects: Thyme and Tarragon
  - Jake Sorensen and Pietro Cicotti

## A data driven variant

- “Overdecompose” the problem so that each process obtains several tasks
- Construct a task graph indicating the **data dependences**
- A task is **runnable** when its data dependences have been satisfied
- A tasks **suspends** until the required **communication** completes



# Observations

- The exact execution order depends on the data dependence structure: communication & computation
- We don't have to hard code a particular overlap strategy
- We can alter the behavior by changing the data dependences, e.g. disable overlap, or by varying the on-node decomposition geometry
- For other algorithms we can add priorities to force a preferred ordering
- Applies to many scales of granularity (i.e. memory locality, network, etc.)

# Architectural directions

# Blue Gene/L

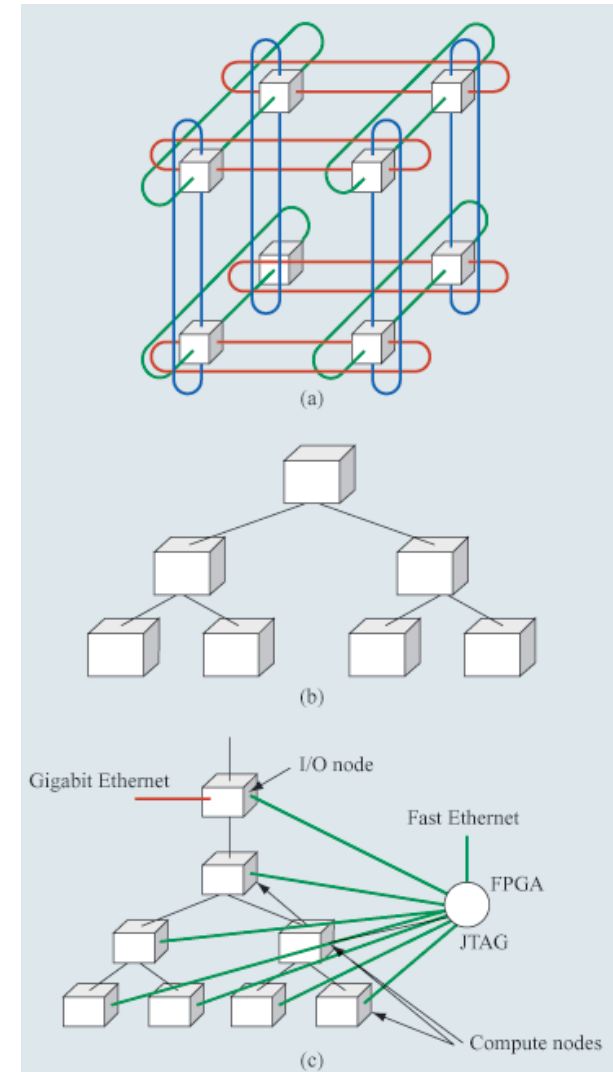
- IBM-US Dept of Energy collaboration
- 64K dual processor nodes: 180 (360) TF peak
  - One CPU dedicated to communication on each node (But may be used for computation)
- Low power
  - Relatively slow processors; power PC 440
  - Small memory (256 MB)
- High performance interconnect
- Lightweight kernel runs on each node
  - Single user, single 2-thread process
  - No context switching, no demand paging

# Blue Gene/L Interconnect

- 3D toroidal mesh (end around)
  - ▶ 175 MB/sec Bandwidth peak
  - ▶ 200 ns latency (32 bytes)
  - ▶ bidirectional
- Rapid combining-broadcast network
  - ▶ 350MB/sec
  - ▶ 1.5  $\mu$ s latency (1-way)
  - ▶ Fast barriers (1.5  $\mu$ s)

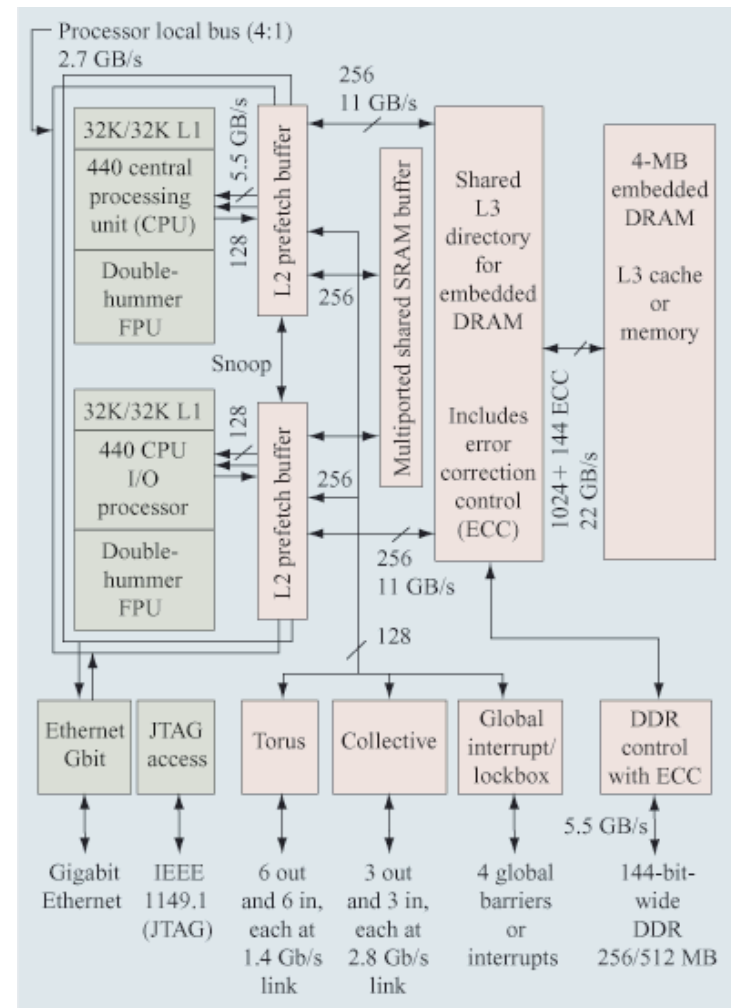
Image courtesy of IBM

<http://www.research.ibm.com/journal/rd/492/gara4.gif>



# Blue Gene/L Compute chip

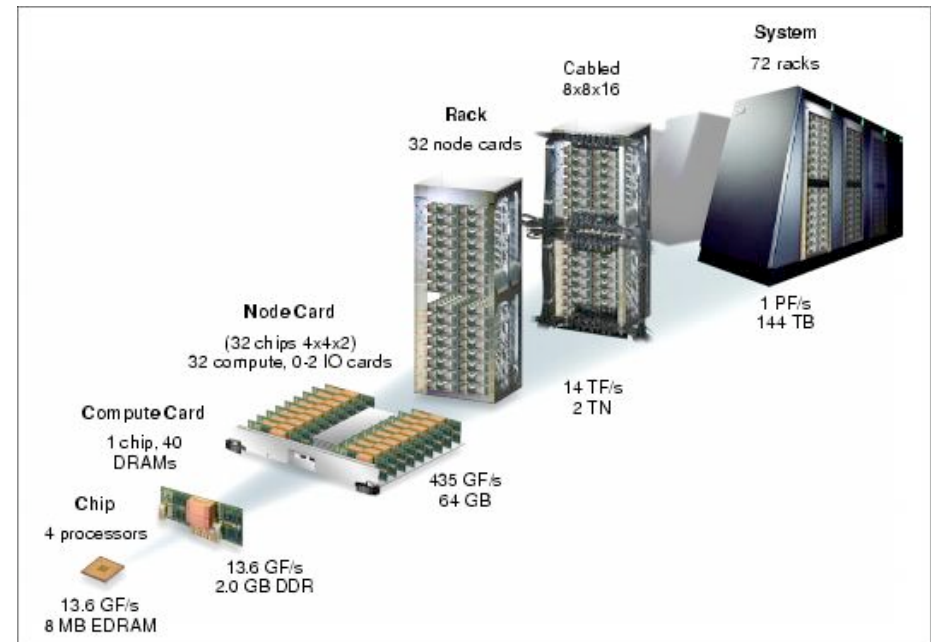
- Shared L3 Cache
- L1 caches not coherent



G. Almasi et. Al, © 2002 IEEE

# Next Generation: Blue Gene/P

- 4-way nodes, 3D Torus
- PowerPC 450 (850 MHz)
- 2 GB memory
- L1 is cache coherent
- Switch latency reduced to 100 ns (32B)



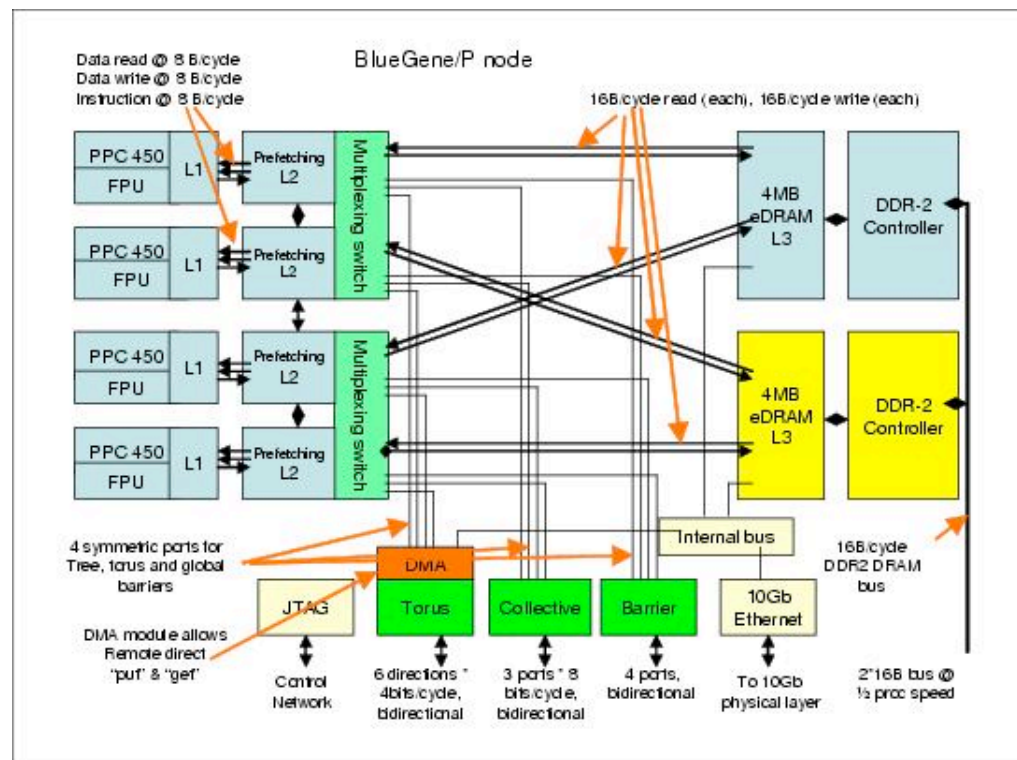
<http://www.redbooks.ibm.com/redbooks/SG247287>

# Interconnection network

- Network routers are embedded within the processor
- Each chip has six nearest-neighbor connections
- Each link provides 425 MB/sec in each direction
  - 5.1 GBps bidirectional bandwidth / node
  - 1.7/3.8 TBps bisection bandwidth, 67 TBps total bandwidth
- Global collective network for broadcast and reduction
- Also used to move data from I/O to compute nodes
- Each Compute and I/O node: 3 links to the global collective network @ 850 MBps per direction
  - 5.1 GBps bidirectional bandwidth per node
  - Latency  $< 2 \mu\text{s}$

# Compute nodes

- Six connections to torus network @ 3.4 Gbp/link
- Three connections to global collective network @ 6.8 Gbps/link



<http://www.redbooks.ibm.com/redbooks/SG247287>

# Programming modes

- Symmetrical Multiprocessing node mode
  - Each node runs 1 MPI process, up to 4 threads
- Virtual node mode
  - Each node runs 4 MPI processes, 1/core
  - Memory and torus network shared by all processes
  - Shared memory is available between processes.
- Dual node mode
  - Each node runs 2 MPI processes, up to 2 threads/process

# nVidia GPU

# NVIDIA GeForce GTX 280

- 240 cores @ 1.296 GHz
- Parallel computing or graphic mode
- 1 GB memory
- 512 bit memory interface @ 141.7 GB/s



# Streaming processing cluster

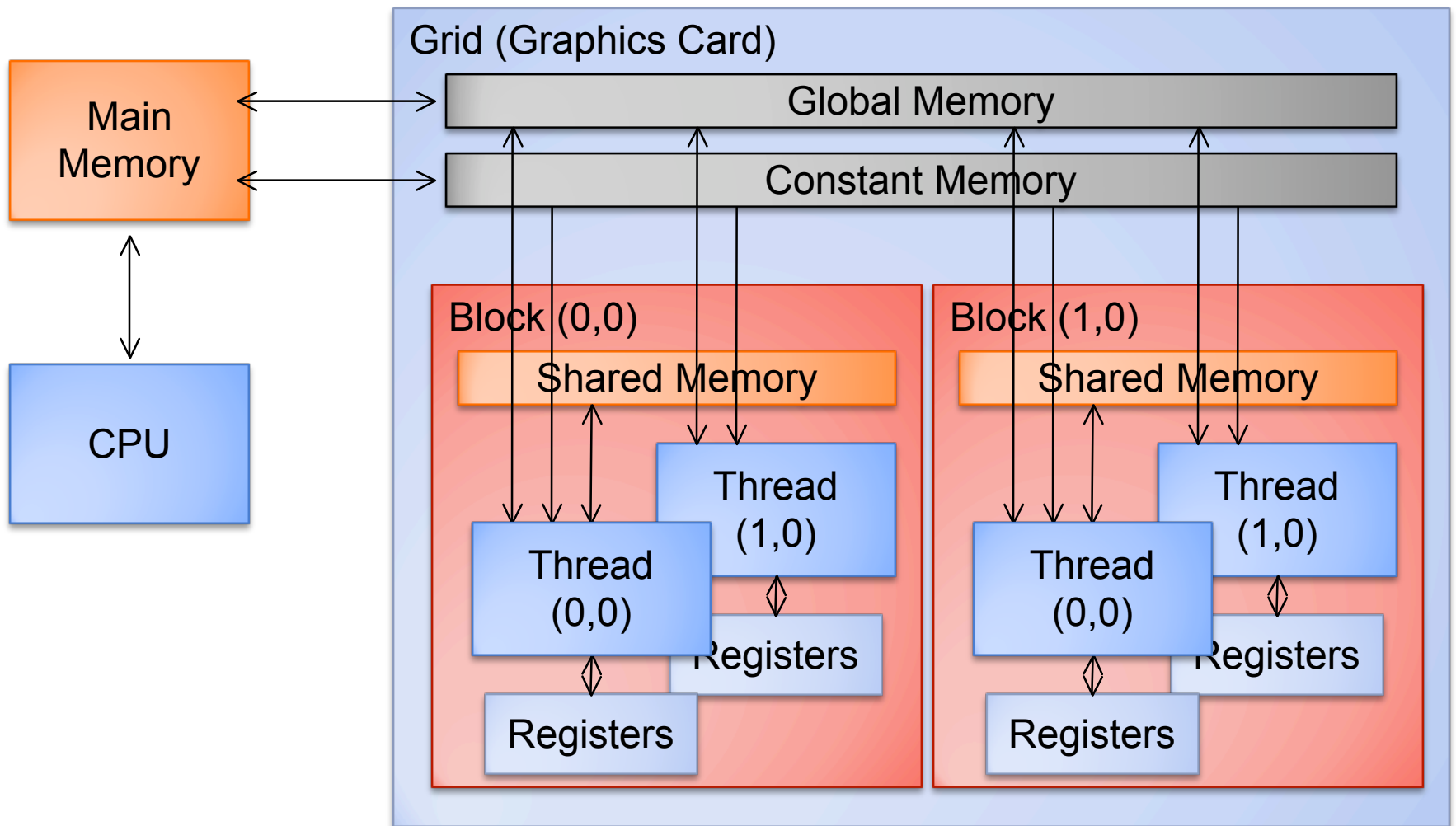
- 3 streaming multiprocessors
- Each multiprocessor has a shared local memory
- Each core supports SIMD
- 128 bits, but S.P. only , DP is 10× slower



# CUDA

- Programming environment with extensions to C
- Basic computational unit is the *grid* (a card)
- Grids comprised of *blocks*
  - 1D, 2D, or 3D in units of threads
  - Threads within the block synchronize together
  - Communicate via fast shared memory
- Blocks within a grid are virtualized. No communication except through slow global memory
- Blocks run multiple *warps*, a group of SIMD threads
  - Half-warps are the unit of scheduling (16 threads currently)
  - Hardware scheduled warps hide latency
  - 4 threads from a warp, context switch to another 4 threads, repeat
- Parallel calls to a kernel specify the layout  
`kernelFunction <<< gridDims, blockDims >>> (args)`
- Compiler will re-arrange loads to hide latencies

# CUDA ARCHITECTURE



## A NOTE ABOUT REGISTERS

- Registers are dynamically partitioned across each block in an Streaming Multiprocessor (8192 in G80)
- Bound to and only accessible from their thread until the block finishes execution
- Can choose between more blocks with fewer threads or more threads with fewer blocks

# MATRIX MULTIPLICATION EXAMPLE

- If each Block has  $16 \times 16$  threads and each thread uses 10 registers, how many threads can run on each SM?
  - Each Block requires  $10 * 256 = 2560$  registers
  - $8192 = 3 * 2560 + \text{change}$
  - So, three blocks can run on an SM as far as registers are concerned
- How about if each thread increases the use of registers by 1?
  - Each Block now requires  $11 * 256 = 2816$  registers
  - $8192 < 2816 * 3$
  - Only two Blocks can run on an SM, **1/3 reduction of parallelism!!!**

\* Slide from David Kirk/NVIDIA and Wen-mei W. Hwu, 2007, ECE 498AL, UIUC