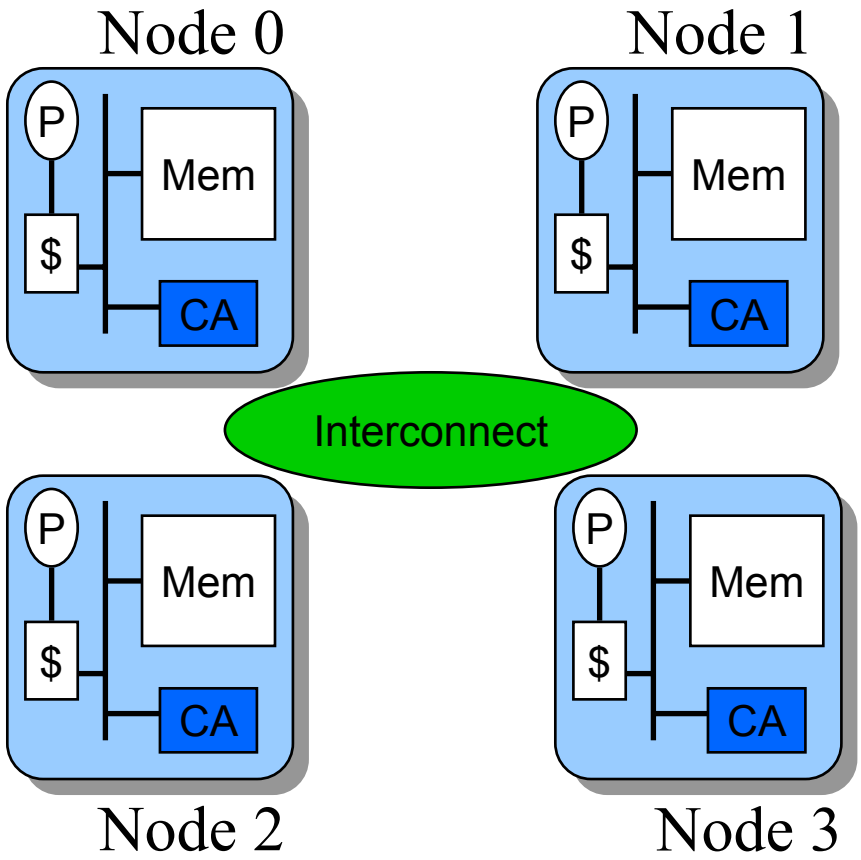
18-742

Lecture 2

Convergence of Parallel Machines



Spring 2005

Prof. Babak Falsafi

http://www.ece.cmu.edu/~ece742

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Reinhardt, Smith, and Singh of University of Illinois, Carnegie Mellon University, University of Wisconsin, Duke University, University of Michigan, and Princeton University.

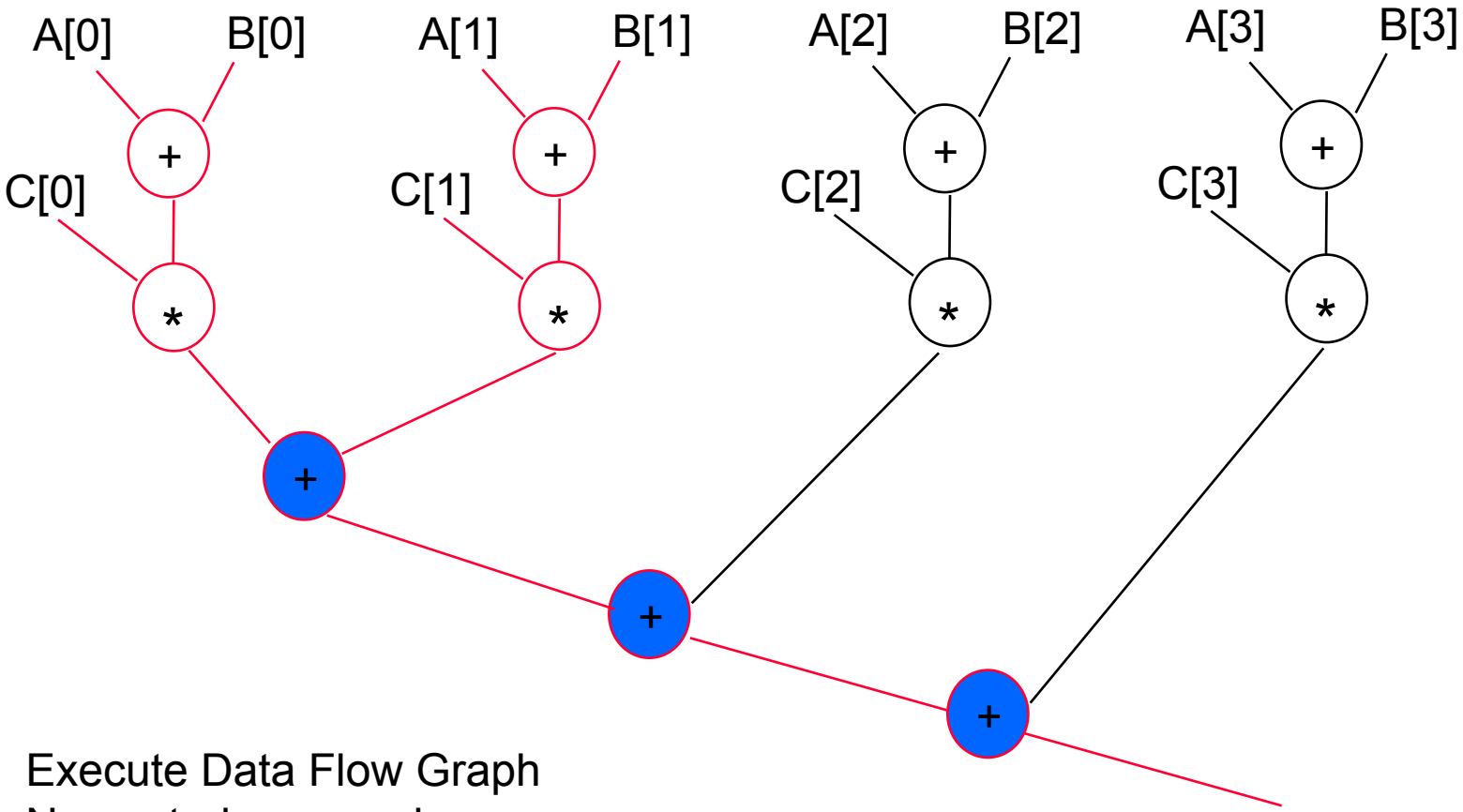
Homework 1

- **Due by next Friday**
- **Assigned on the web by Friday**
  - **Consists of review questions on material read**

Readings

- **From Reader 2**
  - **The book will arrive later. There will be a copies of chapter 2 outside of A302 after class**
  - **G. A. Geist, J. A. Kohl, and P. M. Papadopoulos, *PVM and MPI: A comparison of features*, Calculateurs Paralleles, 8(2), 1996.**
  - **OpenMP: Simple, Portable, Scalable SMP Programming, Web Site**
  - **S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, *The SPLASH-2 Programs: Characterization and Methodological Considerations*, ISCA 1995.**

Data Flow Architectures



Execute Data Flow Graph
No control sequencing

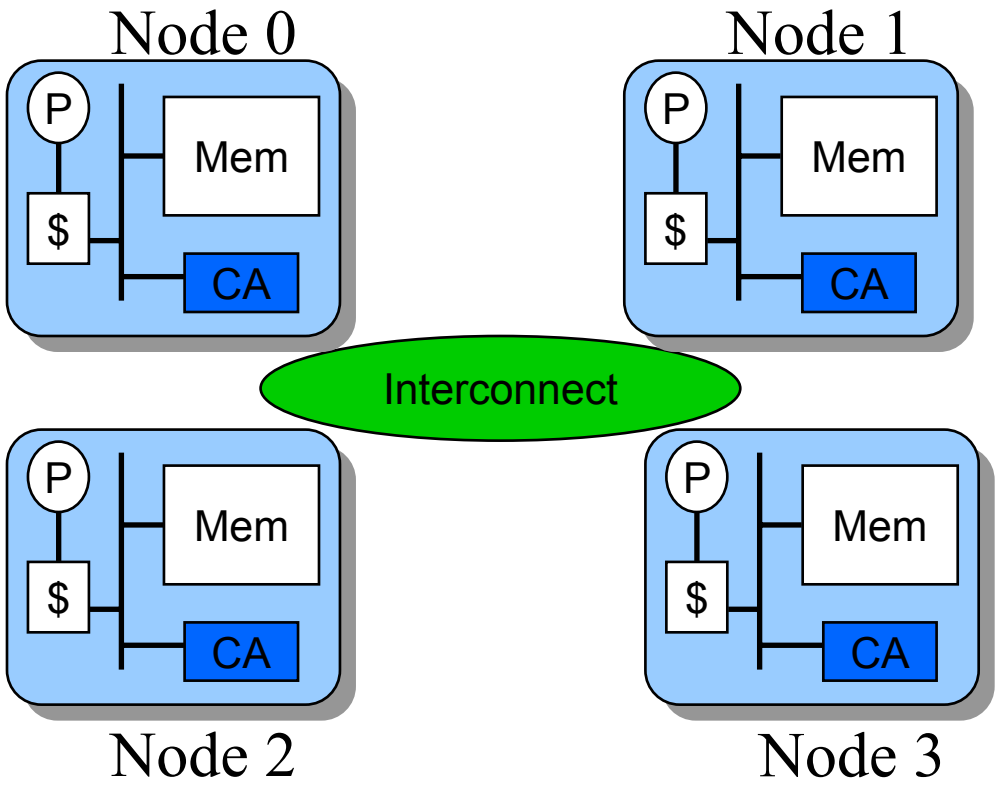
Data Flow Architectures

- Explicitly represent data dependencies (Data Flow Graph)
- No artificial constraints, like sequencing instructions!
  - Early machines had no registers or cache
- Instructions can "fire" when operands are ready
  - Remember tomasulo's algorithm
- How do we know when operands are ready?
- Matching store
  - large associative search!
- Later machines moved to coarser grain (threads)
  - allowed registers and cache for local computation
  - introduced messages (with operations and operands)

Review: Separation of Model and Architecture

- **Shared Memory**
  - Single shared address space
  - Communicate, synchronize using load / store
  - Can support message passing
- **Message Passing**
  - Send / Receive
  - Communication + synchronization
  - Can support shared memory
- **Data Parallel**
  - Lock-step execution on regular data structures
  - Often requires global operations (sum, max, min...)
  - Can support on either SM or MP

Review: A Generic Parallel Machine



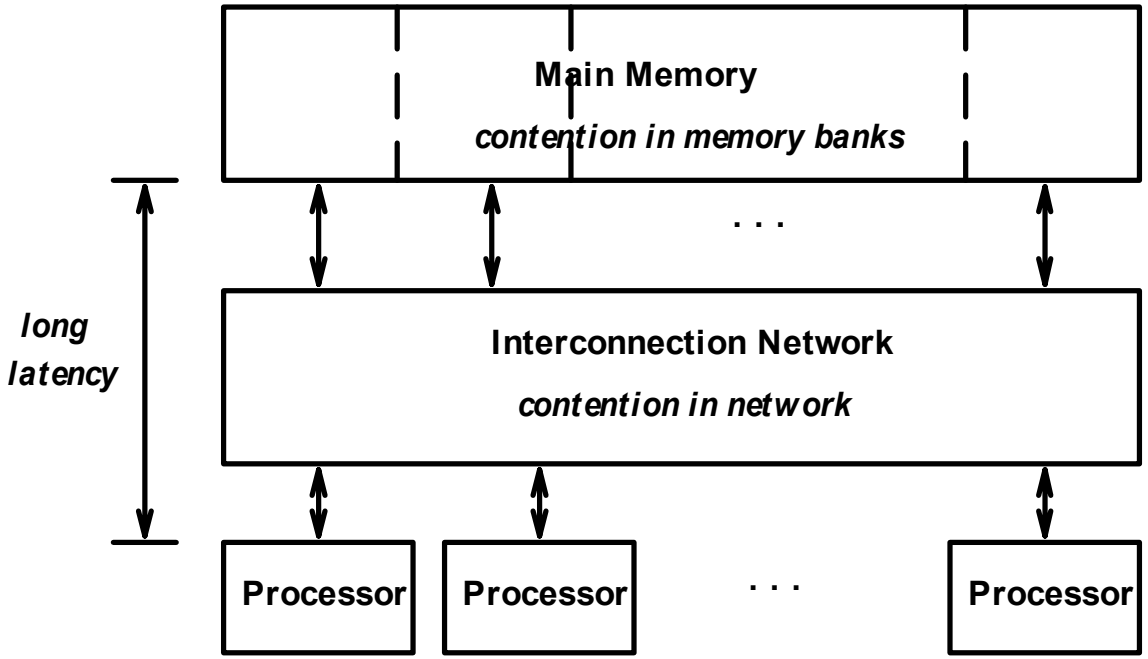
- **Separation of programming models from architectures**
- **All models require communication**
- **Node with processor(s), memory, communication assist**

Let's see UMA, NUMA, Examples, & More History

UMA: Uniform Memory Access

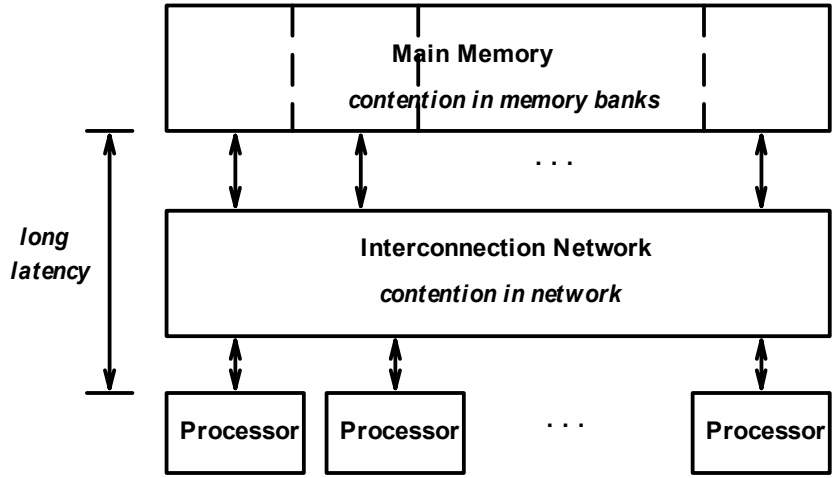
- **Latencies are the same,**
  - ***but may be relatively high***
- **Latencies get worse as system grows**

  => scaling difficulties

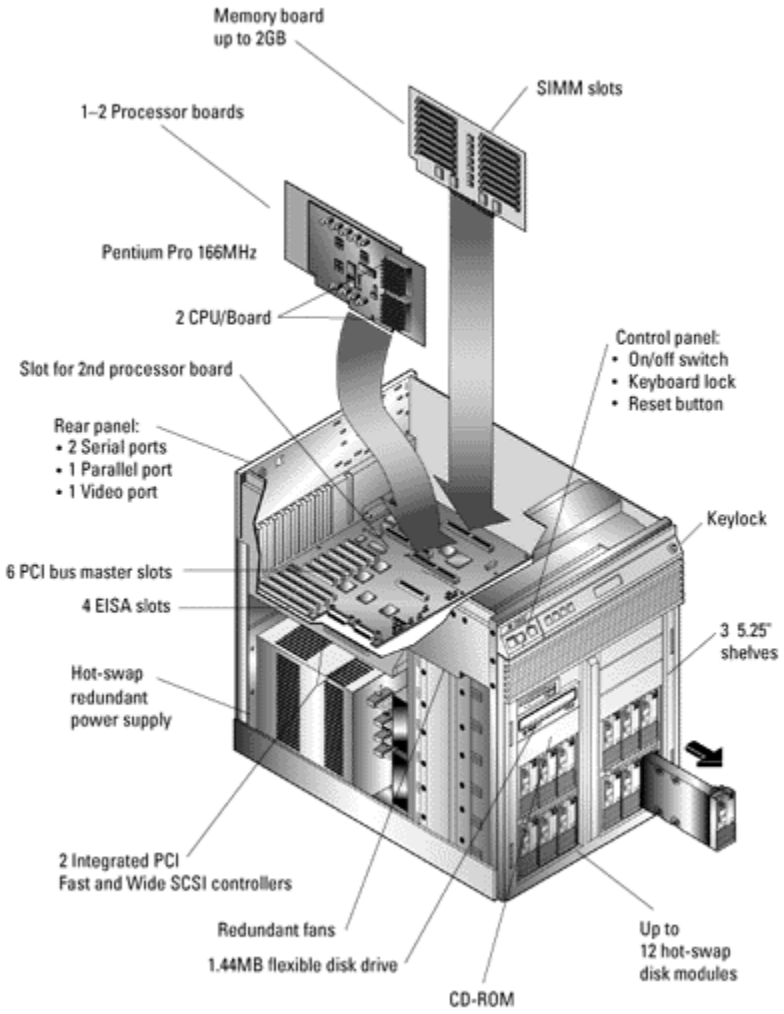


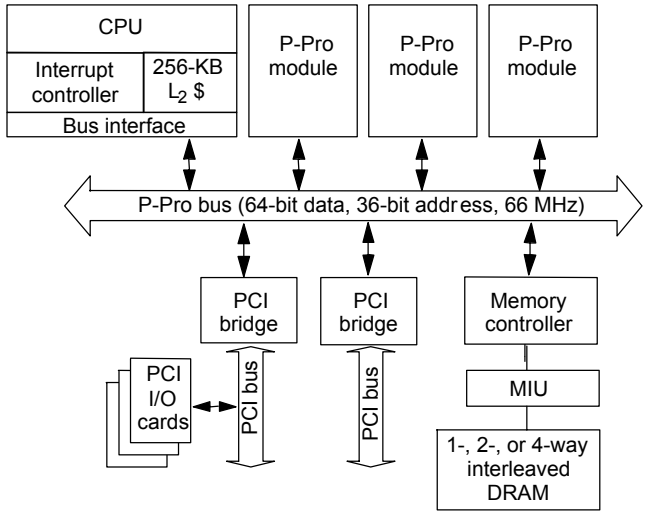
Uniform Memory Access

- Data placement unimportant
- Typically used in small MPs only
- Contention restricts bandwidth
- Caches are often "allowed" in UMA systems
- Also called symmetric multiprocessors (SMP)



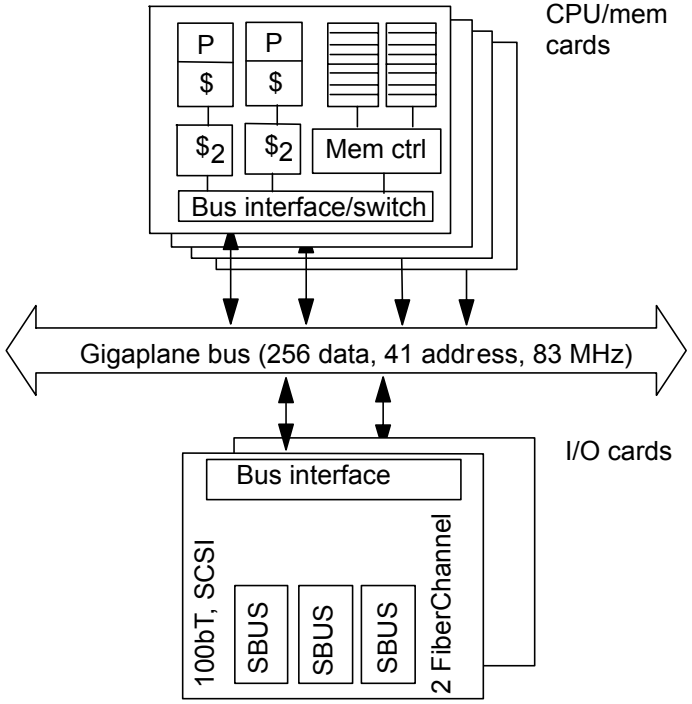
Example: Intel Pentium Pro Quad





- **All coherence and multiprocessing glue in processor module**
- **Highly integrated, targeted at high volume**
- **Low latency and bandwidth**

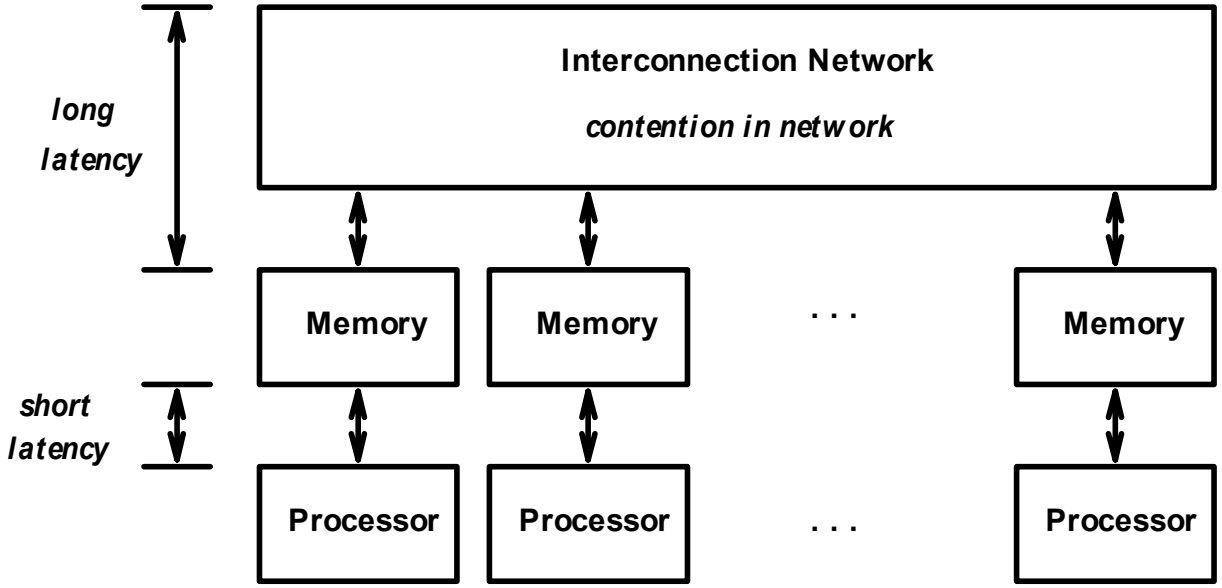
Example: SUN Enterprise



- **16 cards of either type: processors + memory, or I/O**
- **All memory accessed over bus, so symmetric**
- **Higher bandwidth, higher latency bus**

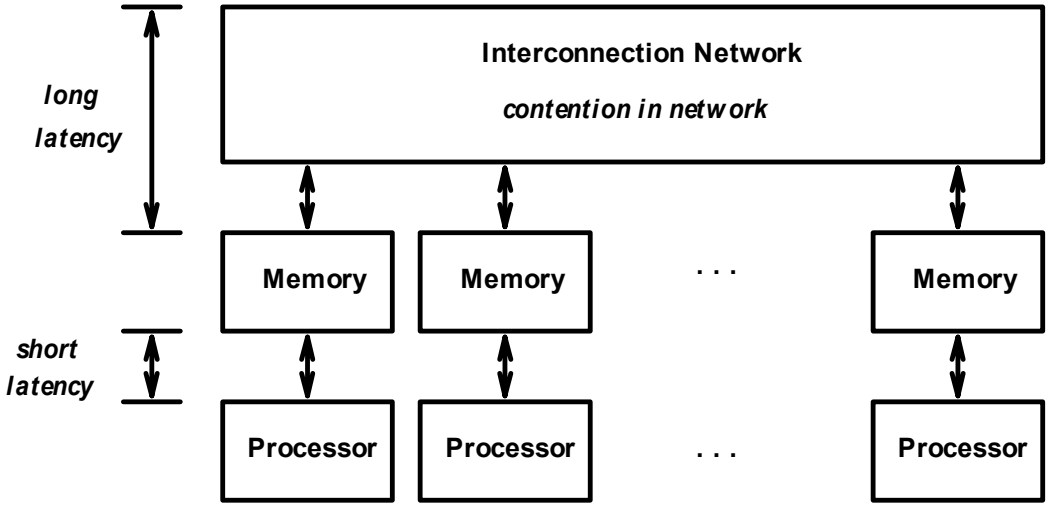
NUMA: NonUniform Memory Access

- Latency low to local memory
- Latency much higher to remote memories
- Performance very sensitive to data placement
- Bandwidth to local memory may be higher
- Contention in network and for memories

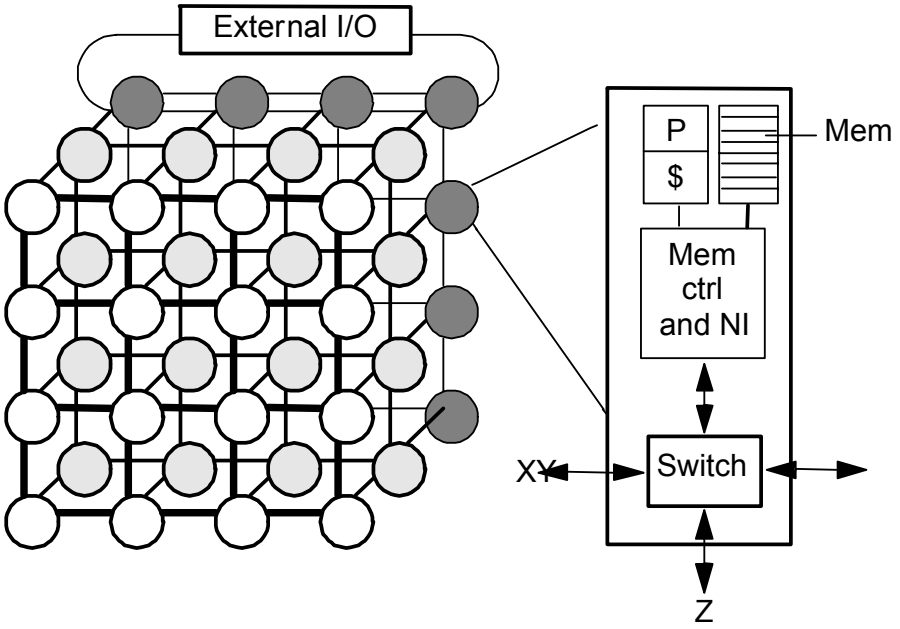


NUMA Multiprocessors, contd.

- **Distributed shared memory**
  - **One logical address space**
  - **Can be treated as shared memory**
- **Multicomputers**
  - **Each processor has its own memory address space**
  - **Use message passing for communication**



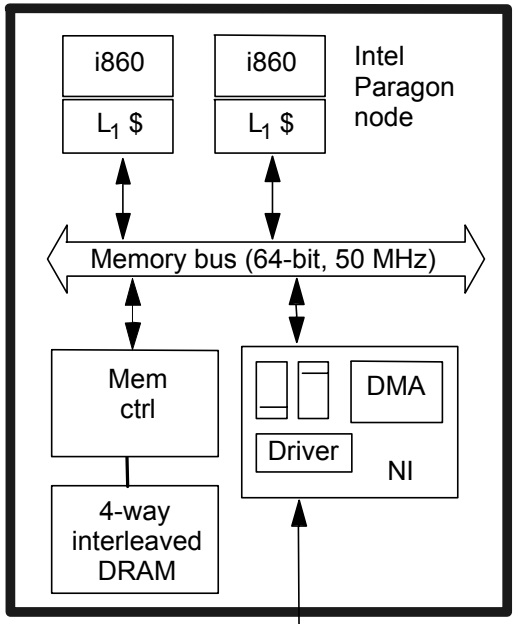
Example: Cray T3E

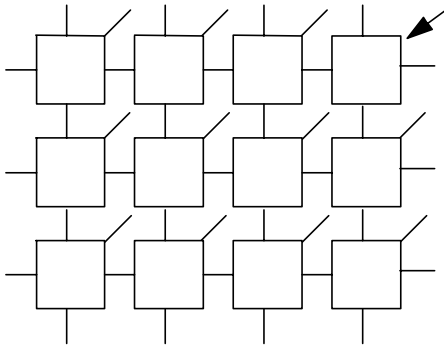


- **Scale up to 1024 processors, 480MB/s links**
- **Memory controller generates comm. request for nonlocal references**
- **No hardware mechanism for coherence (SGI Origin etc. provide this)**

Example Intel Paragon

Sandia' s Intel Paragon XP/S-based Supercomputer



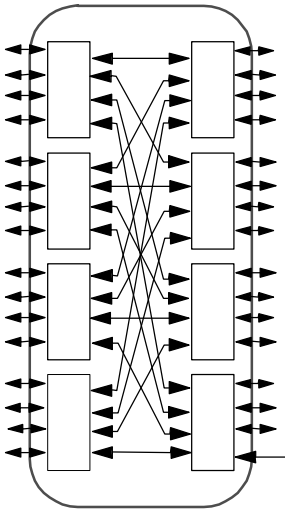


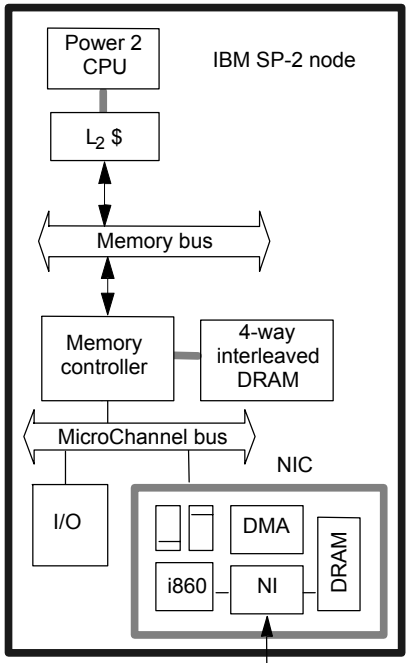
8 bits,
175 MHz,
bidirectional

2D grid network
with processing node
attached to every switch

Example: IBM SP-2

General interconnection network formed from 8-port switches

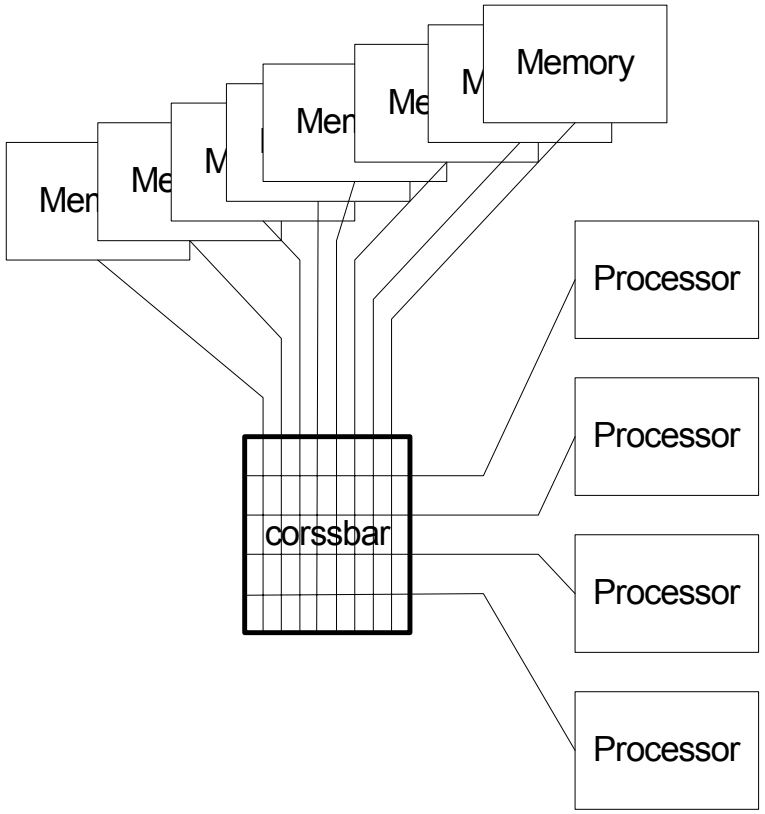




- **Made out of essentially complete RS6000 workstations**
- **Network interface integrated in I/O bus (bw limited by I/O bus)**

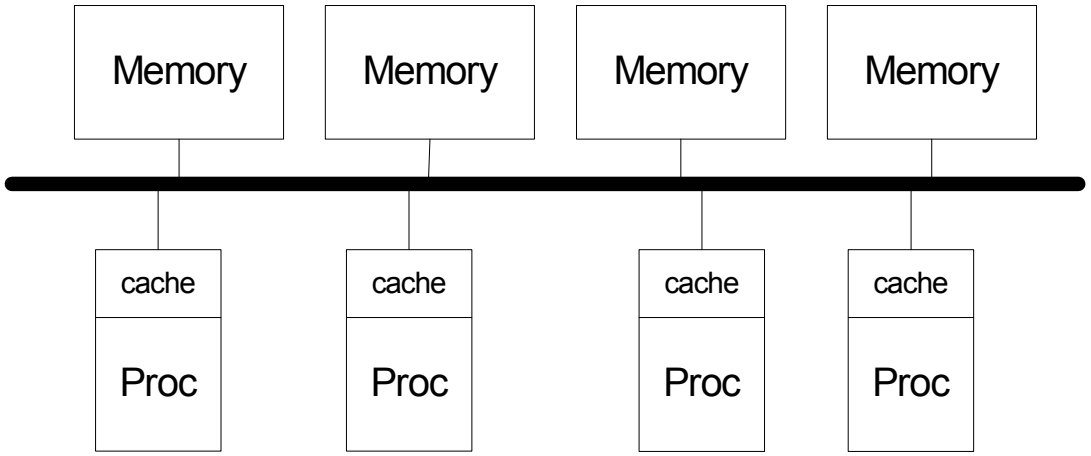
Historical Evolution: 1960s & 70s

- **Early MPs**
  - **Mainframes**
  - **Small number of processors**
  - **crossbar interconnect**
  - **UMA**



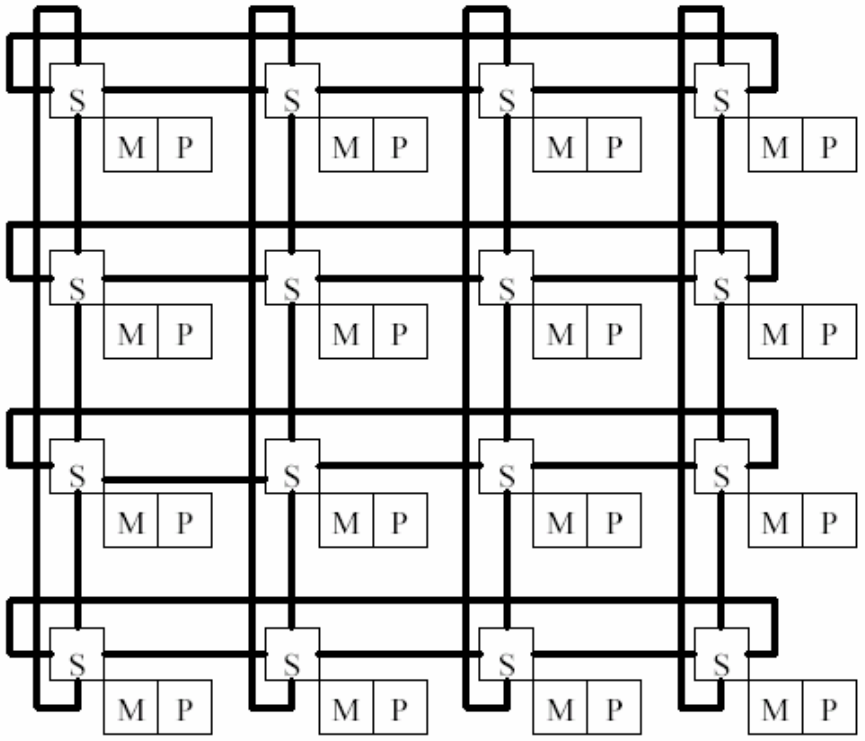
Historical Evolution: 1980s

- **Bus-Based MPs**
  - **enabler: processor-on-a-board**
  - **economical scaling**
  - **precursor of today's SMPs**
  - **UMA**



Historical Evolution: Late 80s, mid 90s

- **Large Scale MPs (Massively Parallel Processors)**
  - **multi-dimensional interconnects**
  - **each node a computer (proc + cache + memory)**
  - **both shared memory and message passing versions**
  - **NUMA**
  - **not commercially viable**
  - **still used for "supercomputing'**

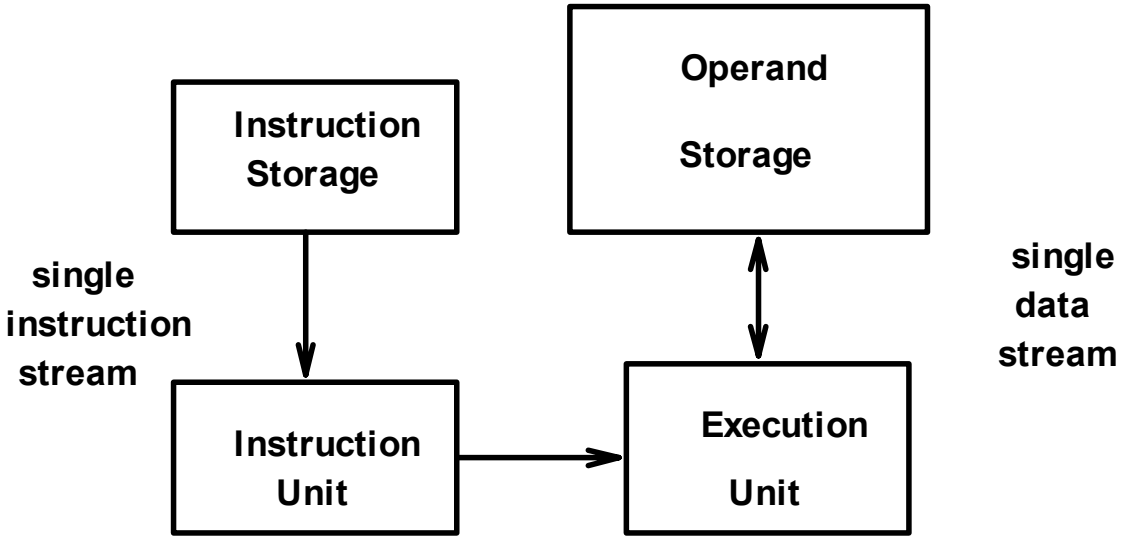


Historical Evolution: Current

- **Small to Mid-Scale SMPs**
  - **One module type: processor + caches + memory**
- **Clusters**
  - **Use high performance LAN to connect small SMPs**
- **Driven by economics**
  - **Smaller systems => higher volumes**
  - **Off-the-shelf components**
- **Driven by applications**
  - **Many more throughput applications (web servers)**
  - **Than parallel applications (weather prediction)**

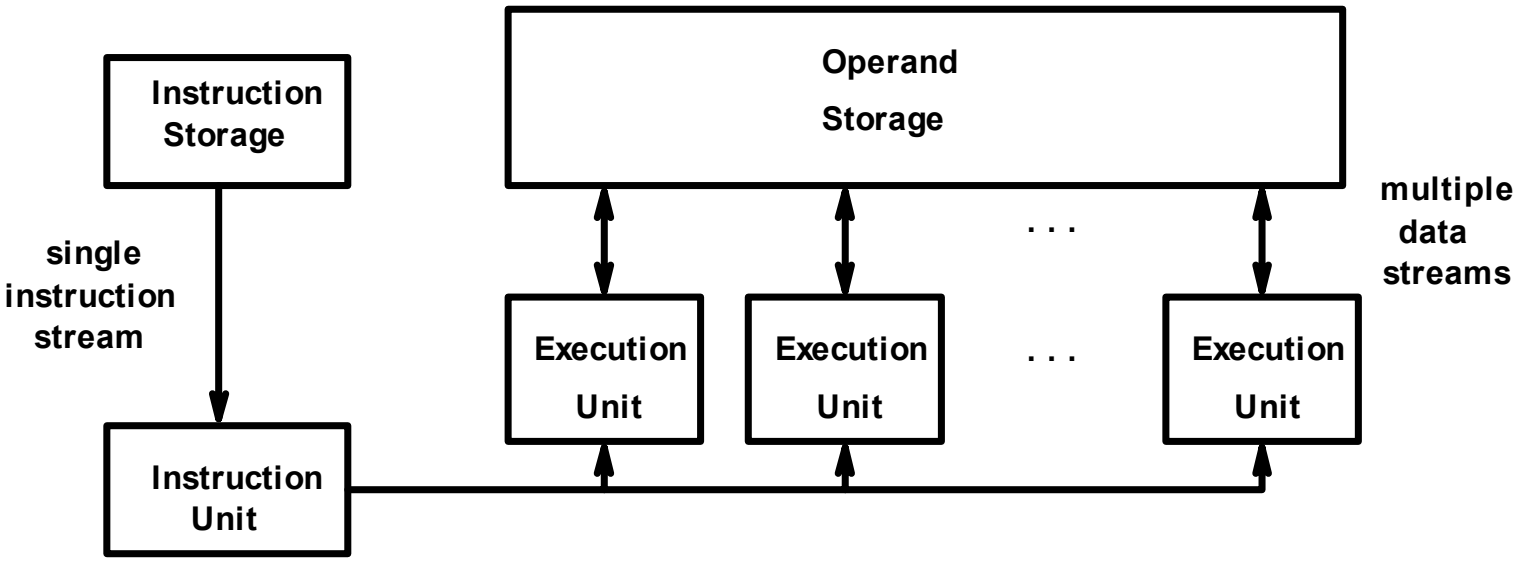
Historical Taxonomy (Flynn)

- **SISD: Single Instruction, Single Data**
- **Operand and instruction storage may be the same**
- **Your basic uniprocessor**



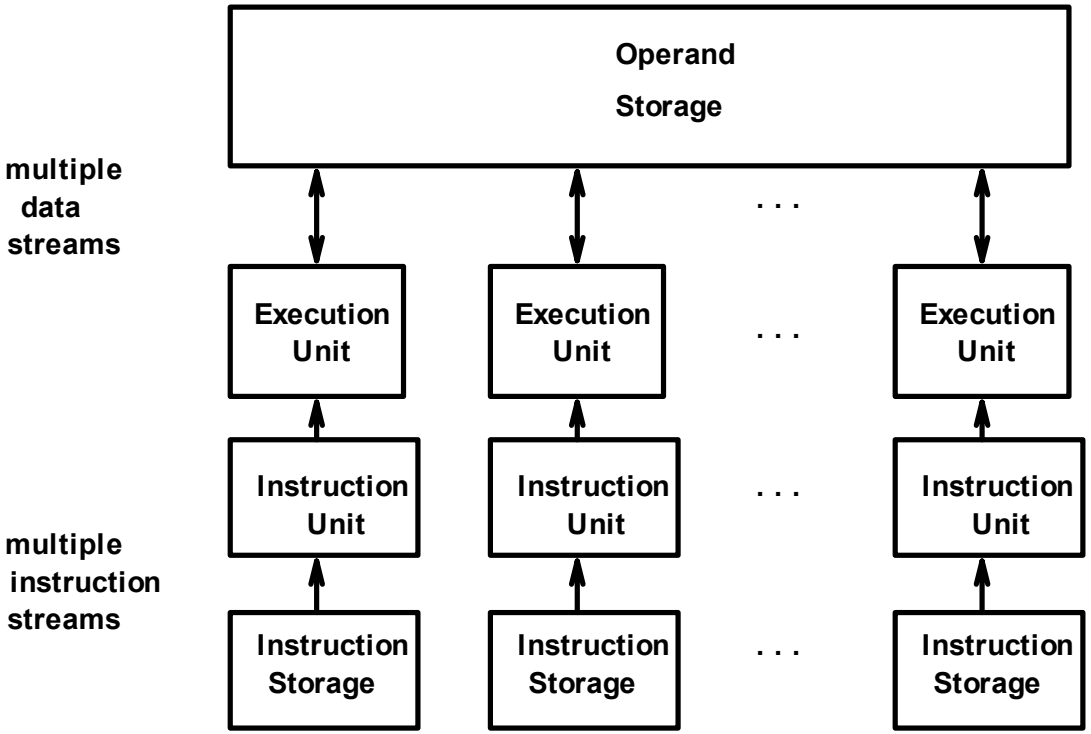
Historical Taxonomy (Flynn)

- **SIMD: Single Instruction, Multiple Data**
  - **Insts and data storage usually separated**
- **Leads to "Data Parallel" programming model**
- **Works better for loop-oriented numerical problems**
- **Automatic parallelization *can* work**



Historical Taxonomy (Flynn)

- **MIMD: Multiple Instruction, Multiple Data**
- **More flexible than SIMD *and of more interest to us***
- **Important for general purpose computing**
- **Automatic parallelization more difficult**



Programming Model Design Issues

- **Naming: How is communicated data and/or partner node referenced?**
- **Operations: What operations are allowed on named data?**
- **Ordering: How can producers and consumers of data coordinate their activities?**
- **Performance**
  - **Latency: How long does it take to communicate in a protected fashion?**
  - **Bandwidth: How much data can be communicated per second? How many operations per second?**

Issue: Naming

- **Single Global Linear-Address-Space (shared memory)**
- **Single Global Segmented-Name-Space (global objects)**
- **Multiple Local Address/Name Spaces (message passing)**
- **Naming strategy affects**
  - **Programmer / Software**
  - **Performance**
  - **Design Complexity**

Issue: Operations

- **Uniprocessor RISC**
  - ld/st and atomic operations on memory
  - arithmetic on registers
- **Shared Memory Multiprocessor**
  - ld/st and atomic operations on local/gobal memory
  - arithmetic on registers
- **Message Passing Multiprocessor**
  - send/receive on local memory
  - broadcast
- **Data Parallel**
  - ld/st
  - Global operations (add, max, etc.)

Issue: Ordering

- **Uniprocessor**
  - **programmer sees order as program order**
  - **out-of-order execution (tomasulo's algorithm) actually changes order**
  - **write buffers**
  - **important to maintain dependencies**
- **Multiprocessor**
  - **What is order among several threads accessing shared data?**
  - **What affect does this have on performance?**
  - **What if implicit order is insufficient?**

Issue: Order/Synchronization

- **Coordination mainly takes three forms:**
  - **mutual exclusion (e.g., spin-locks)**
  - **event notification**
    - » **point-to-point (e.g., producer-consumer)**
    - » **global (e.g., end of phase indication, all or subset of processes)**
  - **global operations (e.g., sum)**
- **Issues:**
  - **synchronization name space (entire address space or portion)**
  - **granularity (per byte, per word, ... => overhead)**
  - **low latency, low serialization (hot spots)**
  - **variety of approaches**
    - » **test&set, compare&swap, ldLocked-stConditional**
    - » **Full / Empty bits and traps**
    - » **queue-based locks, fetch&op with combining**
    - » **scans**

Performance Issue: Latency

- **Must deal with latency when using fast processors**
- **Options:**
  - **Reduce frequency of long latency events**
    - » **algorithmic changes, computation and data distribution**
  - **Reduce latency**
    - » **cache shared data, network interface design, network design**
  - **Tolerate latency**
    - » **message passing overlaps computation with communication (program controlled)**
    - » **SM overlaps access completion and computation using consistency model and prefetching**

Performance Issue: Bandwidth

- **Private and global bandwidth requirements**
- **Private bandwidth requirements can be supported by:**
  - **distributing main memory among PEs**
  - **application changes, local caches, memory system design**
- **Global bandwidth requirements can be supported by:**
  - **scalable interconnect technology**
  - **distributed main memory and caches**
  - **efficient network interfaces**
  - **avoiding contention (hot spots) through application changes**

Cost of Communication

Cost = Frequency x (Overhead + Latency + Xfer size/BW - Overlap)

- **Frequency = number of communications per unit work**
  - **algorithm, placement, replication, bulk data transfer**
- **Overhead = processor cycles spent initiating or handling**
  - **protection checks, status, buffer mgmt, copies, events**
- **Latency = time to move bits from source to dest**
  - **comm assist, topology, routing, congestion**
- **Transfer time = time through bottleneck**
  - **comm assist, links, congestions**
- **Overlap = portion overlapped with useful work**
  - **comm assist, comm operations, processor design**

Summary

- **Motivation & Applications**
- **Theory, History, & A Generic Parallel Machine**
- **UMA, NUMA, Examples, & More History**
- **Programming Models**
  - **Shared Memory**
  - **Message Passing**
  - **Data Parallel**
- **Issues in Programming Models**
  - **Function: naming, operations, & ordering**
  - **Performance: latency, bandwidth, etc.**