AMS 250: An Introduction to High Performance Computing

Parallel Computer Architecture

Shawfeng Dong

shaw@ucsc.edu

(831) 459-2725

Astronomy & Astrophysics

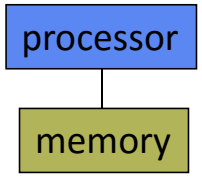
University of California, Santa Cruz

Outline

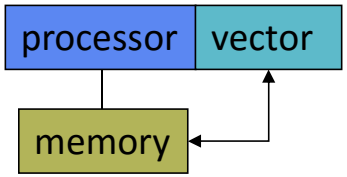
- Parallel architecture types
- Instruction-level parallelism
- Vector processing
- SIMD
- Shared memory
  - Memory organization: UMA, NUMA
  - Coherency: CC-UMA, CC-NUMA
- Interconnection networks
- Distributed memory
- Clusters

Parallel Architecture Types

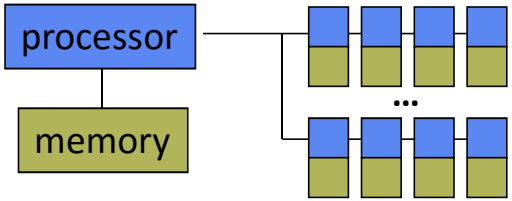
- **Uniprocessor**
  - Scalar processor



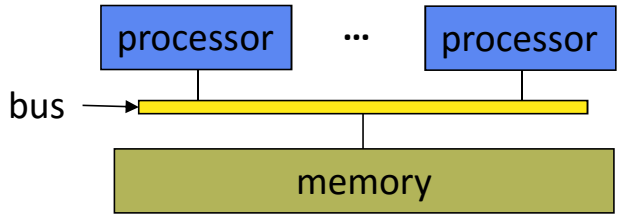
  - Vector processor



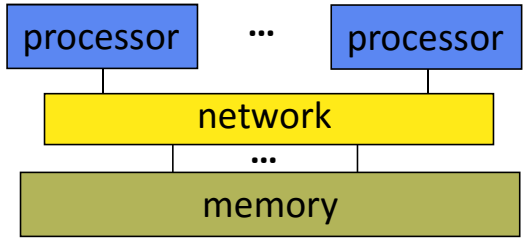
  - Single Instruction Multiple Data (SIMD)



- **Shared Memory Multiprocessor (SMP)**
  - Shared memory address space
  - Bus-based memory system

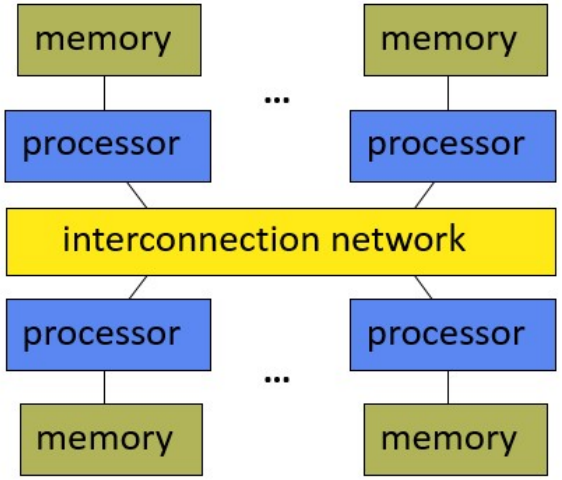


  - Interconnection network



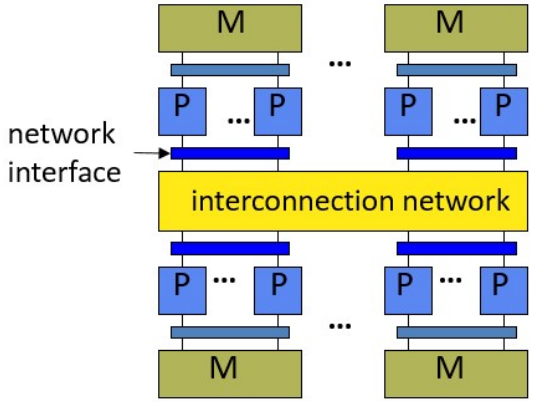
Parallel Architecture Types (2)

- **Distributed Memory Multiprocessor**
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - Many, many processors

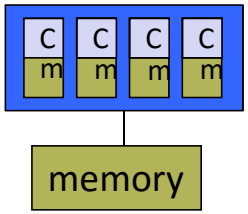
- **Cluster of SMPs**
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes



  - Can also be regarded as MPP if processor number is large

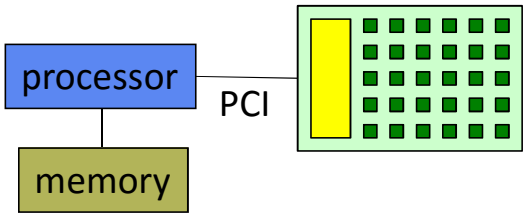
Parallel Architecture Types (3)

- **Multicore**
  - Multicore processor

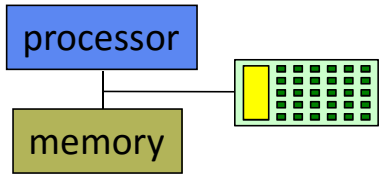


cores can be hardware multithreaded (hyper-threading)

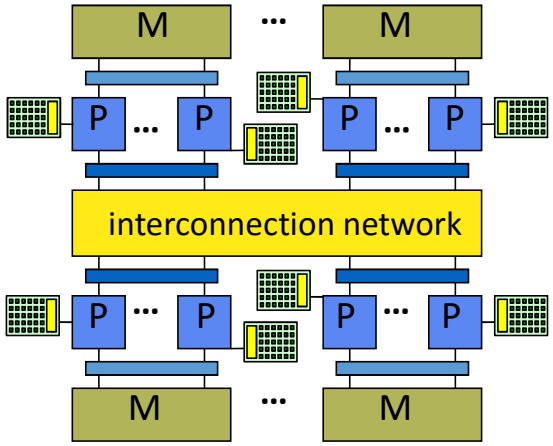
  - GPU accelerator



  - "Fused" processor accelerator



- **Multicore SMP+GPU Cluster**
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes
  - GPU accelerators attached



How do you get parallelism in the hardware?

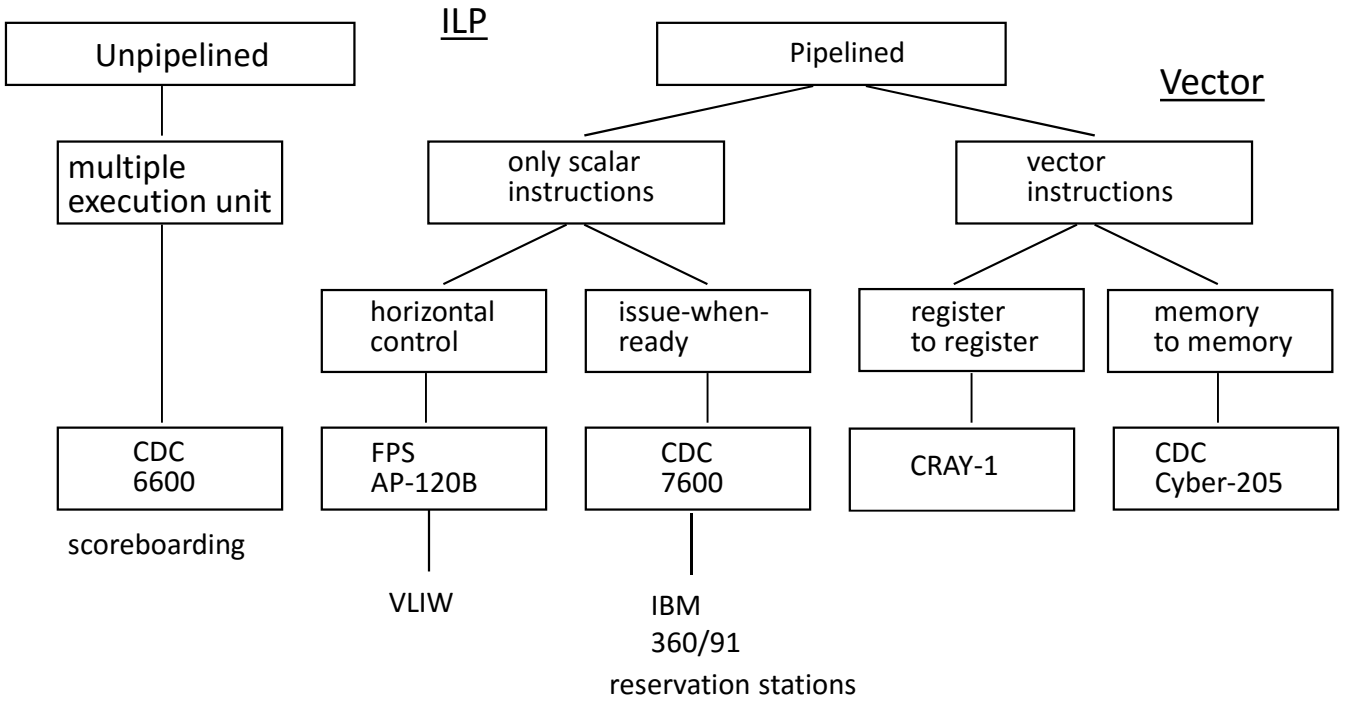
- Instruction-Level Parallelism (ILP)
- Data parallelism
  - Increase amount of data to be operated on at same time
- Processor parallelism
  - Increase number of processors
- Memory system parallelism
  - Increase number of memory units
  - Increase bandwidth to memory
- Communication parallelism
  - Increase amount of interconnection between elements
  - Increase communication bandwidth

Instruction-Level Parallelism

- Opportunity for splitting up instruction processing is called **Instruction-Level Parallelism (ILP)** - https://en.wikipedia.org/wiki/Instruction-level_parallelism
- Micro-architectural techniques that are used to exploit ILP include:
  - Instruction pipelining
  - Overlapped execution
  - Multiple functional units
  - Out-of-order execution
  - Multi-issue execution
  - Superscalar processing
  - Speculative execution
  - Branch prediction
  - Very Long Instruction Word (VLIW)
  - EPIC
  - Register renaming
  - Hardware multithreading (hyper-threading)
- ILP is exploited by both the *hardware* and the *compilers*.

Parallelism in Single Processor Computers

History of processor architecture innovation



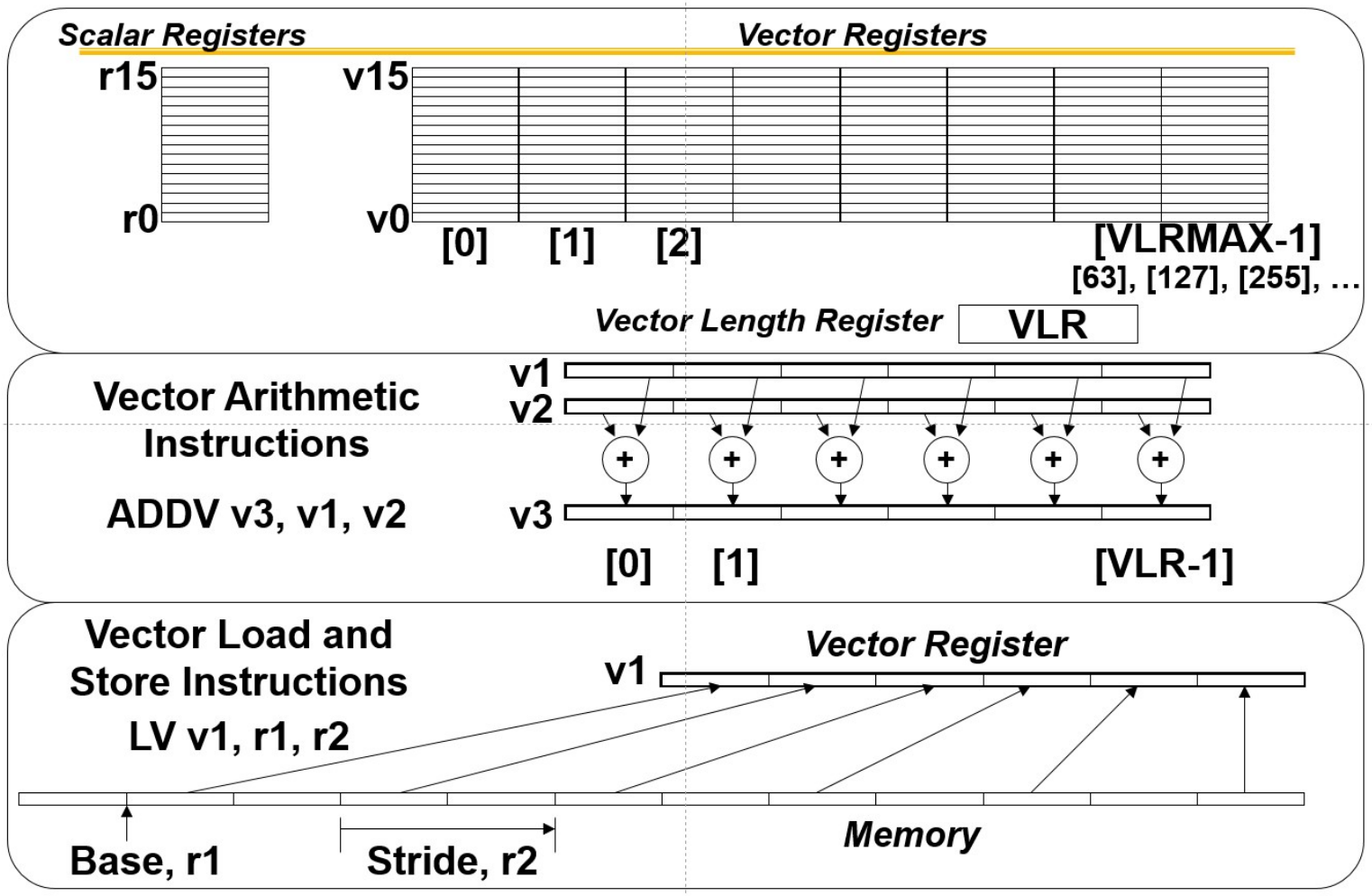
Vector Processing

- Scalar processing
  - Processor instructions operate on scalar values
  - integer registers and floating point registers
- Vectors
  - Set of scalar data
  - Vector registers
    - integer, floating point (typically)
  - Vector instructions operate on vector registers (SIMD)
- Vector unit pipelining
- Multiple vector units
- Vector chaining

Liquid-cooled with inert fluorocarbon

Cray-2

Vector Programming Model



Vector Code Example

```
# C code
for (i=0; i<64; i++)
    C[i] = A[i] + B[i];

! Fortran code
C(1:64) = A(1:64) + B(1:64)
! or
C = A + B
```

```
# Scalar Code
   LI     R4, 64
loop:
   L.D    F0, 0(R1)
   L.D    F2, 0(R2)
   ADD.D  F4, F2, F0
   S.D    F4, 0(R3)
   DADDIU R1, 8
   DADDIU R2, 8
   DADDIU R3, 8
   DSUBIU R4, 1
   BNEZ   R4, loop
```

```
# Vector Code
   LI      VLR, 64
   LV      V1, R1
   LV      V2, R2
   ADDV.D  V3, V1, V2
   SV      V3, R3
```

Vector Instruction Set Advantages

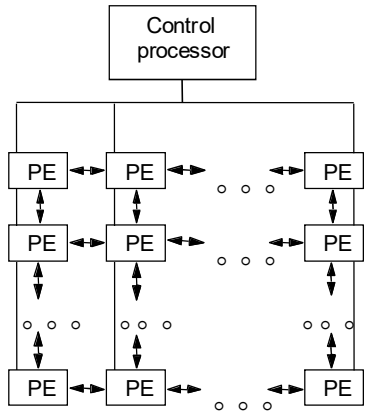
- Compact
  - one short instruction encodes N operations => N*Flop bandwidth
- Expressive, tells hardware that these N operations:
  - are independent
  - use the same functional unit
  - access disjoint registers
  - access registers in the same pattern as previous instructions
  - access a contiguous block of memory (unit-stride load/store)
  - or access memory in a known pattern (strided load/store)
- Scalable
  - can run same object code on more parallel pipelines or *lanes*

Properties of Vector Processors

- Each result independent of previous result
  - => long pipeline, compiler ensures no dependencies
  - => high clock rate
- Vector instructions access memory with known pattern
  - => highly interleaved memory
  - => amortize memory latency of 64-plus elements
  - => no (data) caches required! (but use instruction cache)
- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work (≈ loop)
  - => fewer instruction fetches

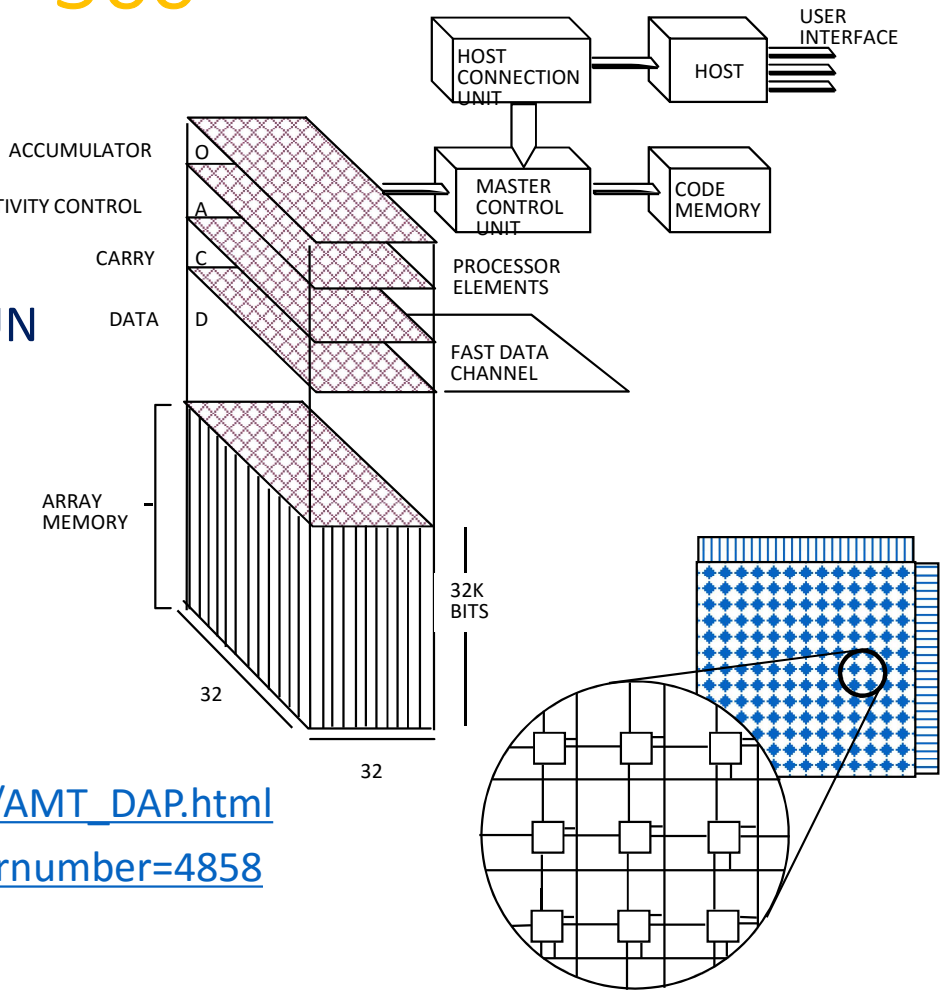
Data Parallel Architectures

- SIMD (Single Instruction Multiple Data)
  - Logical single thread (instruction) of control
  - Processor associated with data elements
- Architecture
  - Array of simple processors with memory
  - Processors arranged in a regular topology
  - Control processor issues instructions
    - All processors execute same instruction (maybe disabled)
  - Specialized synchronization and communication
  - Specialized reduction operations
  - Array processing



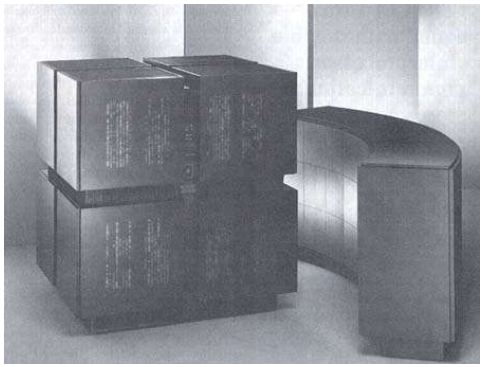
AMT DAP 500

- Applied Memory Technology (AMT)
- Distributed Array Processor (DAP)
  - 32x32 array of bit-organized processors
  - attached to a host computer such as SUN
- ~ 6 – 60 MFLOPS (32-bit)
- first delivered in 1987



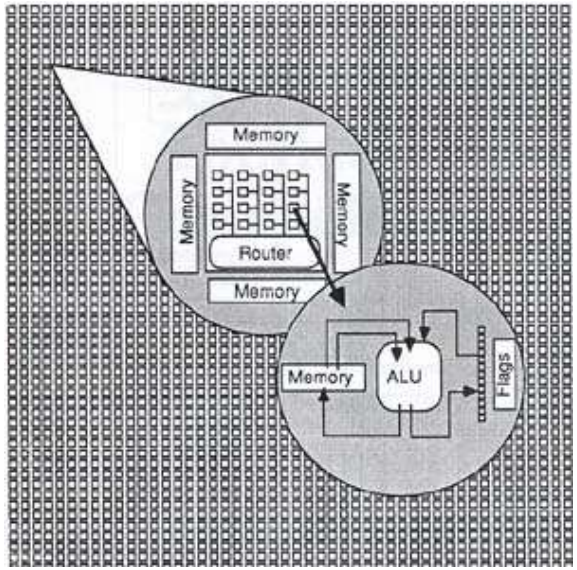
- http://www.computermuseum.org.uk/fixed_pages/AMT_DAP.html
- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4858

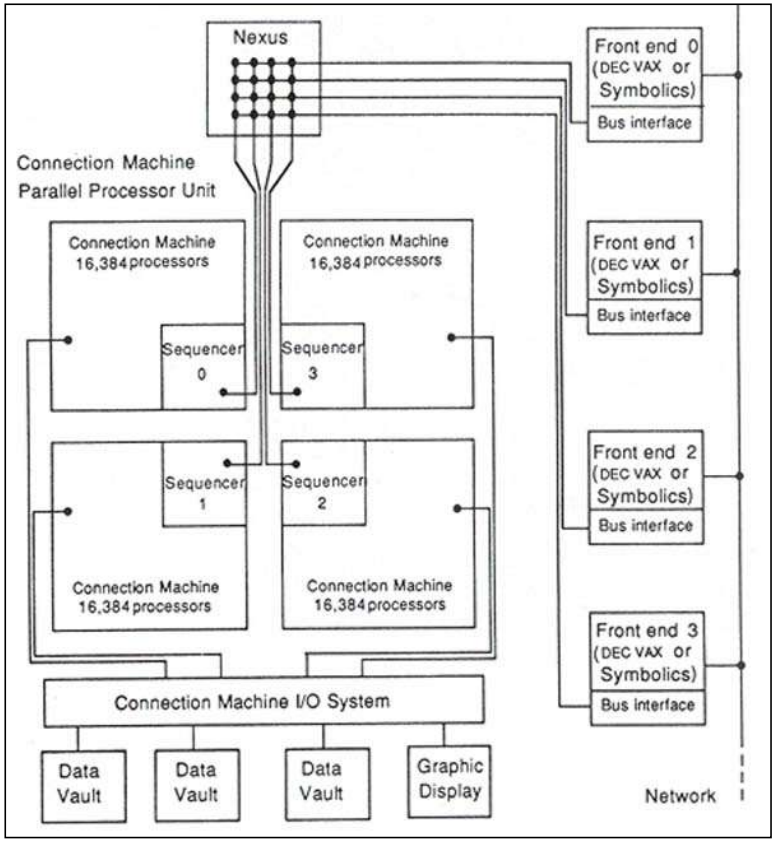
Thinking Machines Connection Machine



16,000 processors!!!

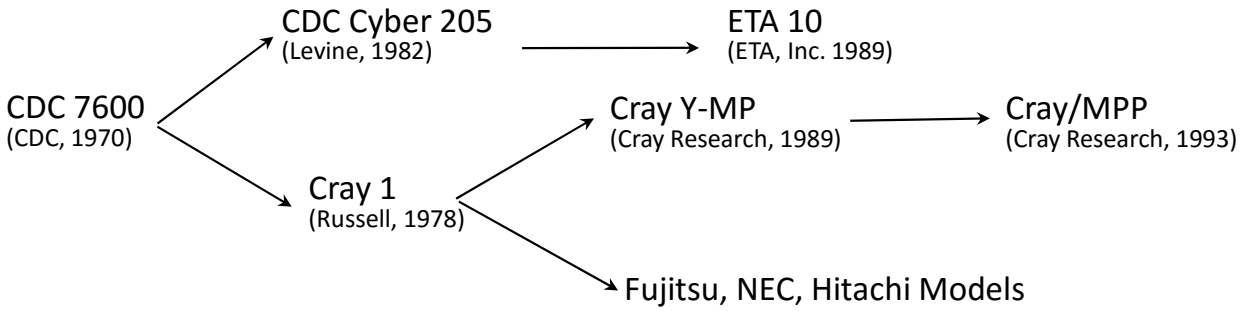
(Tucker, IEEE Computer, Aug. 1988)



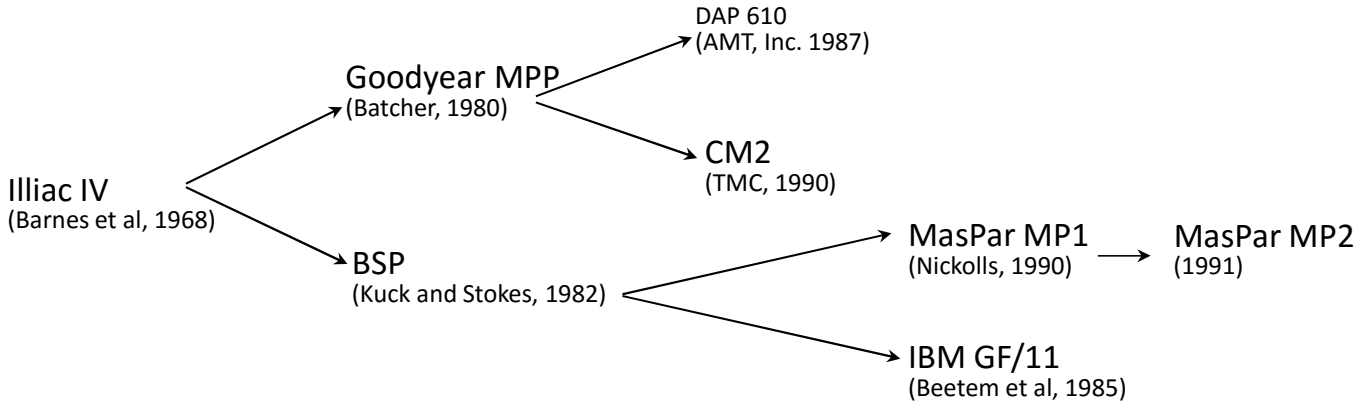


https://en.wikipedia.org/wiki/Connection_Machine

Vector and SIMD Processing Timeline



(a) Multivector track



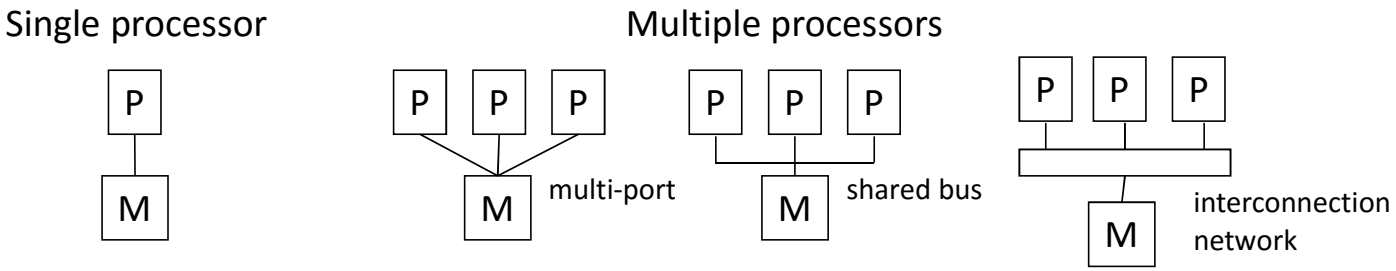
(b) SIMD track

Shared Physical Memory

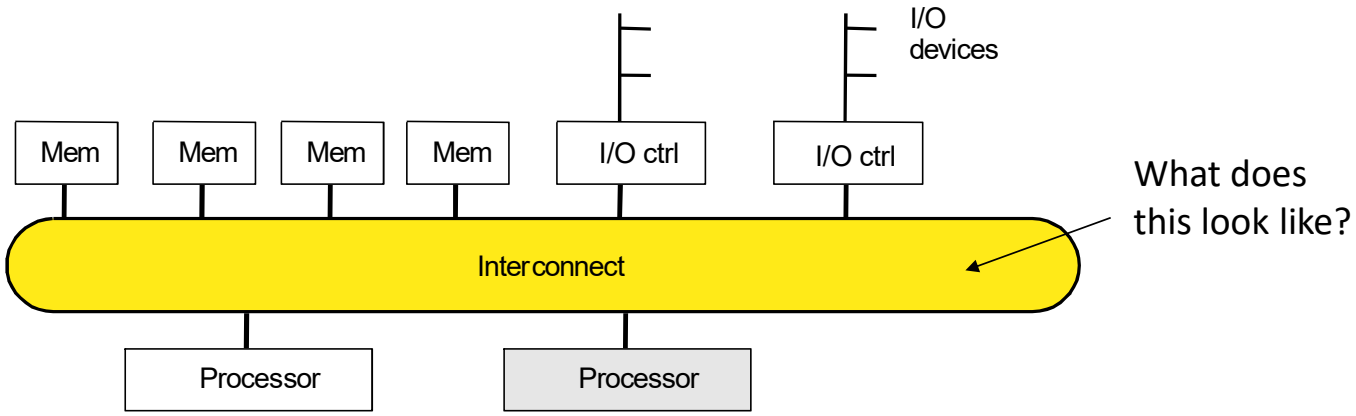
- Add processors to single processor computer system
- Processors *share* computer system resources
  - Memory, storage, …
- Sharing physical memory
  - Any processor can reference any memory location
  - Any I/O controller can reference any memory address
  - Single physical memory address space
- Operating system runs on any processor, or all
  - OS see single memory address space
  - Uses shared memory to coordinate
- Communication occurs as a result of loads and stores

Shared Memory Multiprocessors (SMP)

- Architecture types



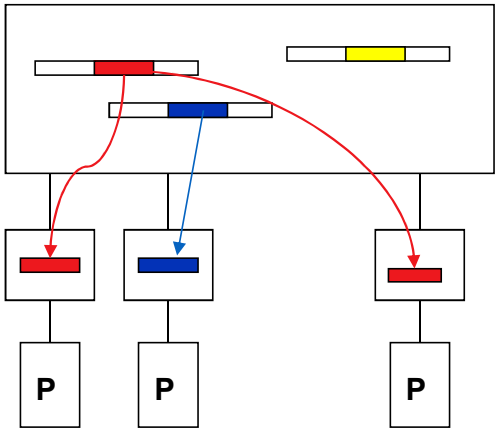
- Differences lie in memory system interconnection



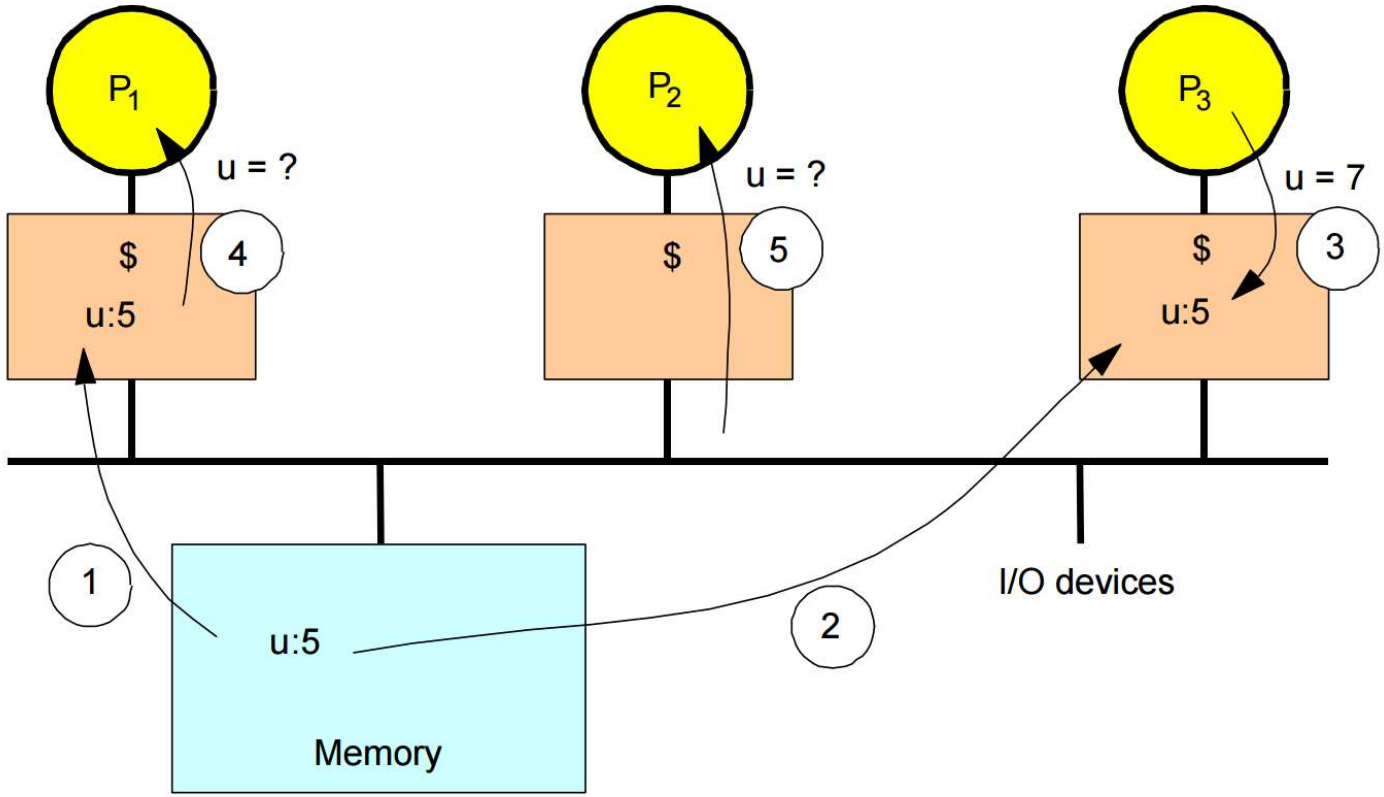
Caching in Shared Memory Systems

- Reduce average latency
  - automatic replication closer to processor
- Reduce average bandwidth
- Data is logically transferred from producer to consumer to memory
  - store: reg → mem
  - load: reg ← mem
- Processors can share data efficiently
- What happens when store and load are executed on different processors?

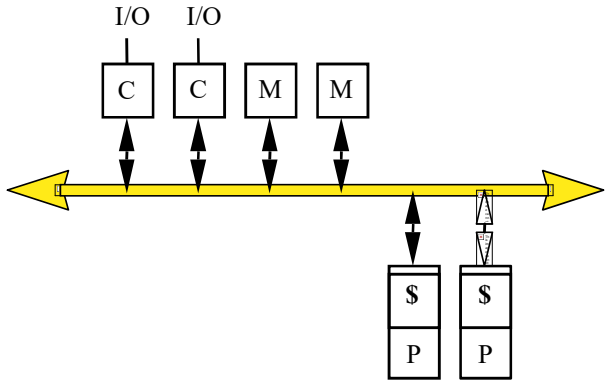
  => Cache coherence problems



Cache Coherence Problem

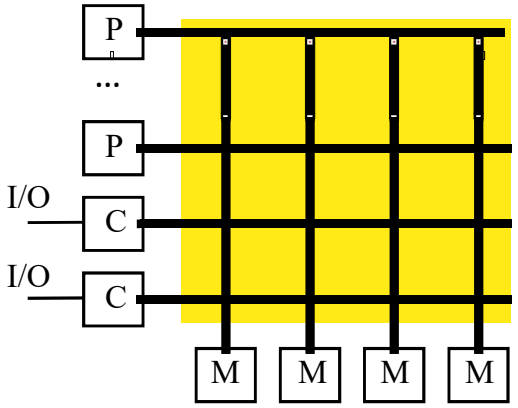


Bus-based SMP



- Memory bus handles all memory read/write traffic
- Processors share bus
- *Uniform Memory Access* (*UMA*)
  - Memory (not cache) uniformly equidistant
  - Take same amount of time (generally) to complete
- May have multiple memory modules
  - Interleaving of physical address space
- Caches introduce memory hierarchy
  - Lead to data consistency problems
  - Cache coherency hardware necessary (*CC-UMA*)

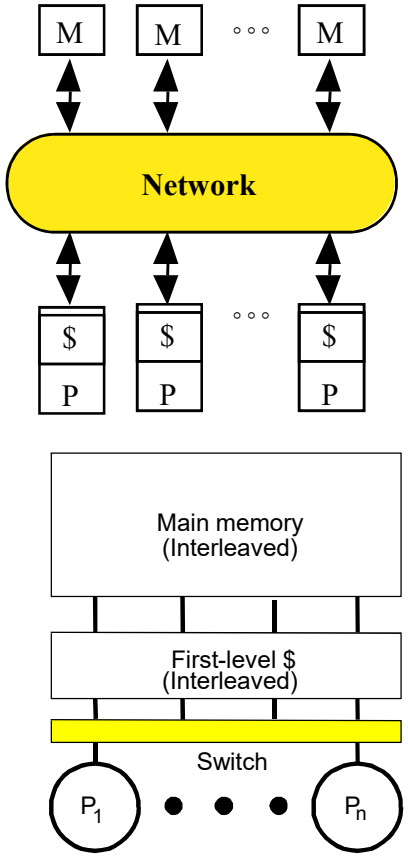
Crossbar SMP



- Replicates memory bus for every processor and I/O controller
  - Every processor has direct path
- UMA SMP architecture
- Can still have cache coherency issues
- Multi-bank memory or interleaved memory
- Advantages
  - Bandwidth scales linearly (no shared links)
- Problems
  - High incremental cost (cannot afford for many processors)
  - One solution is to use switched multi-stage interconnection network

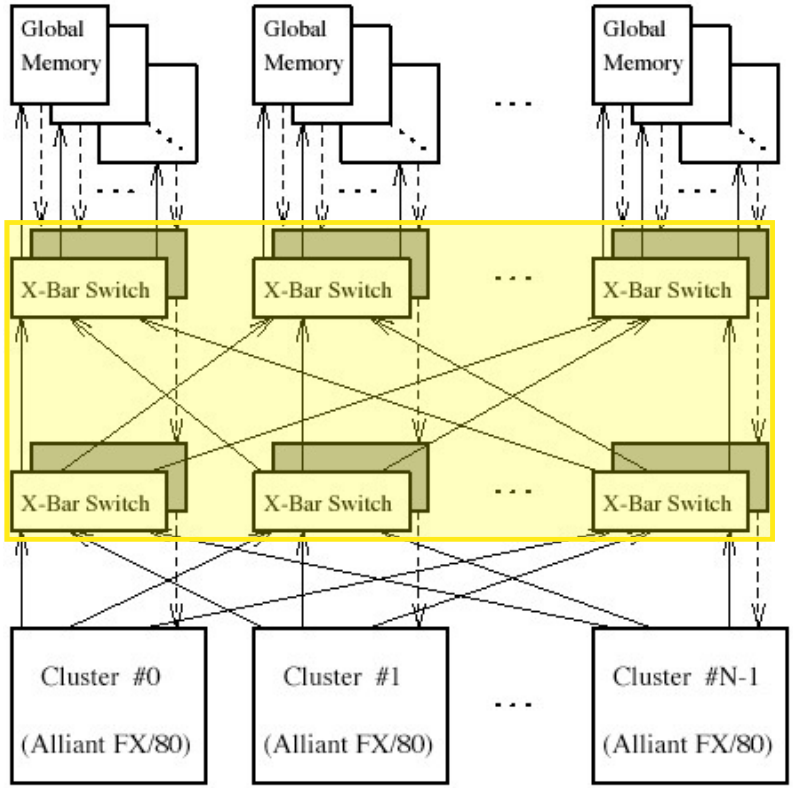
"Dance Hall" SMP and Shared Cache SMP

- "Dance Hall" SMP
  - Interconnection network connects processors to memory
  - Centralized memory (UMA)
  - Network determines performance
    - Continuum from bus to crossbar
    - Scalable memory bandwidth
  - Memory is physically separated from processors
  - Could have cache coherence problems
- Shared cache SMP reduces coherence problem and provides fine grained data sharing

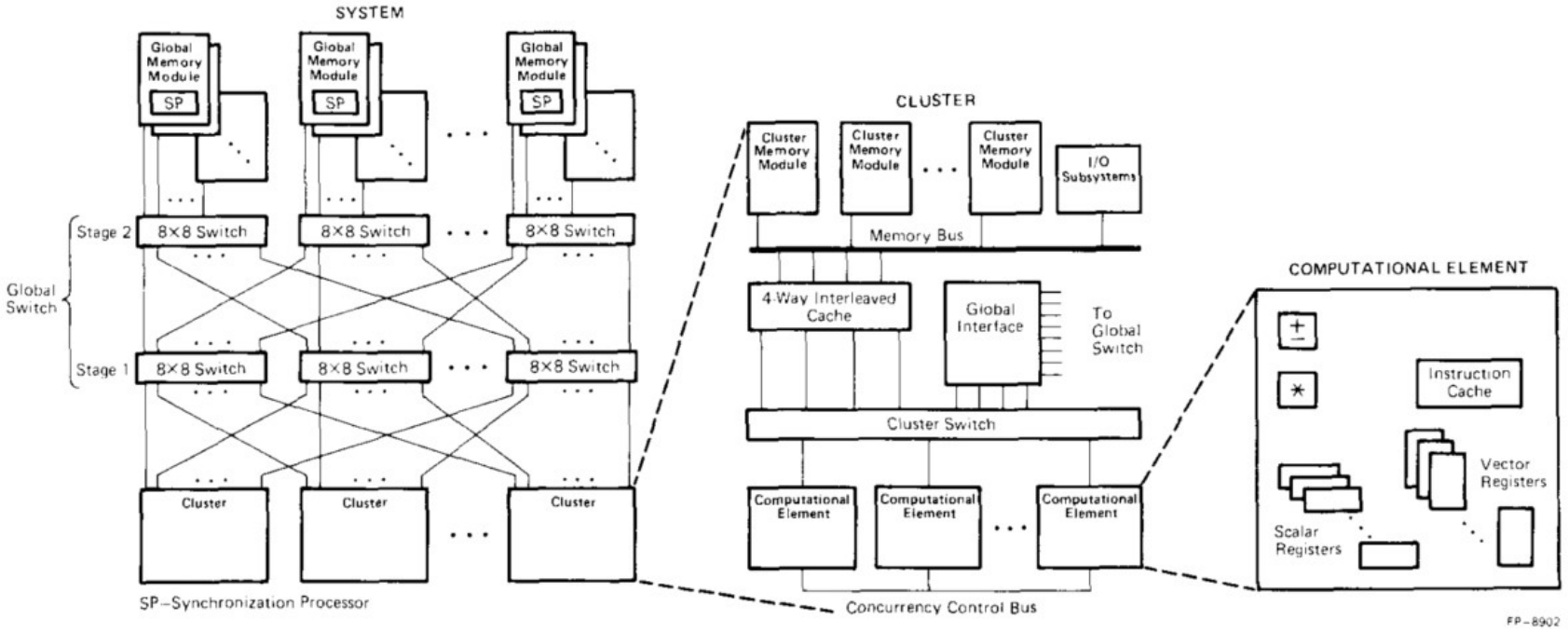


University of Illinios CSRD Cedar Machine

- Center for Supercomputing Research and Development
- Multi-cluster scalable parallel computer
- Alliant FX/80
  - 8 processors w/ vectors
  - Shared cache
  - HW synchronization
- Omega switching network
- Shared global memory
- SW-based global memory coherency
- 1988

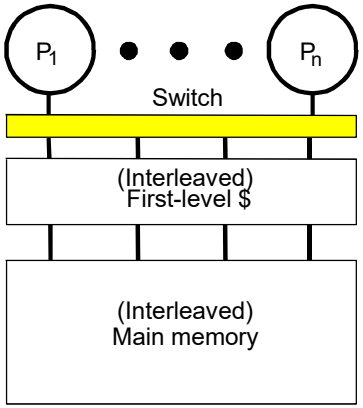


Organization of the Cedar Parallel Processor

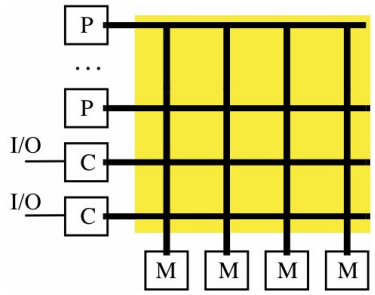


http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10362

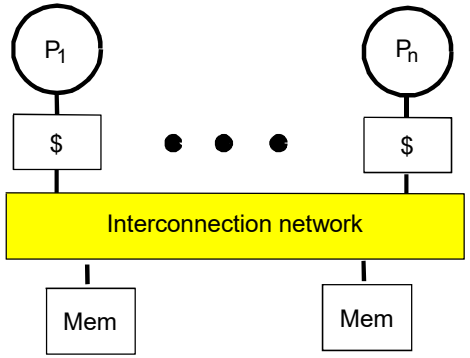
Natural Extensions of the Memory System



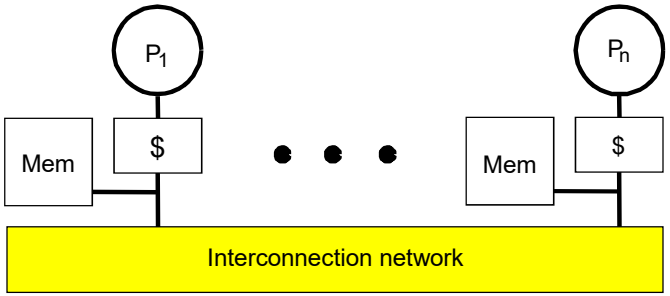
(a) Shared Cache



(b) Crossbar, Interleaved



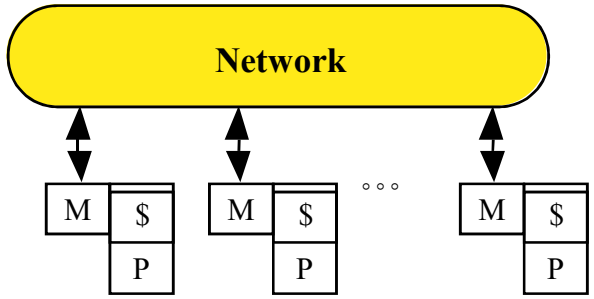
(c) Centralized Memory Dance Hall, UMA



(d) Distributed Memory (NUMA)

Non-Uniform Memory Access (NUMA) SMPs

- Distributed memory
- Memory is physically resident close to each processor
- Memory is still shared
- *Non-Uniform Memory Access* (NUMA)
  - Local memory and remote memory
  - Access to local memory is faster, remote memory slower
  - Access is non-uniform
  - Performance will depend on data locality
- Cache coherency is still an issue (more serious)
- Interconnection network architecture is more scalable



Definitions

- Memory operation (load, store, read-modify-write, …)
- Processor perspective
  - Write: subsequent reads return the value
  - Read: subsequent writes cannot affect the value
- *Coherent memory system*
  - There exists a serial order of memory operations on each location such that
    - operations issued by a process appear in order issued
    - value returned by each read is that written by previous write

    $\Rightarrow$ write propagation + write serialization

Motivation for Memory Consistency

- Coherence implies that writes to a location become visible to all processors in the same order
- But when does a write become visible?
- How do we establish orders between a write and a read by different processors?
  - Use event synchronization
- Implement hardware protocol for cache coherency
- Protocol will be based on model of memory consistency

Memory Consistency

- Specifies constraints on the order in which memory operations (from any process) can appear to execute with respect to each other
  - What orders are preserved?
  - Given a load, constrains the possible values returned by it
- Implications for both programmer and system designer
  - Programmer uses to reason about correctness
  - System designer can use to constrain how much accesses can be reordered by compiler or hardware
- Contract between programmer and system
- Need coherency systems to enforce memory consistency

Sequential Consistency

- Total order achieved by interleaving accesses from different processes
  - Maintains *program order*
  - Memory operations (from all processes) appear to issue, execute, and complete atomically with respect to others
  - As if there was a single memory (no cache)

"A multiprocessor is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." [Lamport, 1979]

Sequential Consistency (Sufficient Conditions)

- There exist a total order consistent with the memory operations becoming visible in program order
- Sufficient Conditions
  - every process issues memory operations in program order
  - after write operation is issued, the issuing process waits for write to complete before issuing next memory operation (atomic writes)
  - after a read is issued, the issuing process waits for the read to complete and for the write whose value is being returned to complete (globally) before issuing its next memory operation
- Cache-coherent architectures implement consistency

Requirements of a Cache Coherent System

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (0) Determine when to invoke coherence protocol
  - (a) Find info about state of block in other caches to determine action
  - (b) Locate the other copies
  - (c) Communicate with those copies (invalidate/update)
- (0) is done the same way on all systems
  - state of the line is maintained in the cache
  - protocol is invoked if an “access fault” occurs on the line
- Different approaches distinguished by (a) to (c)

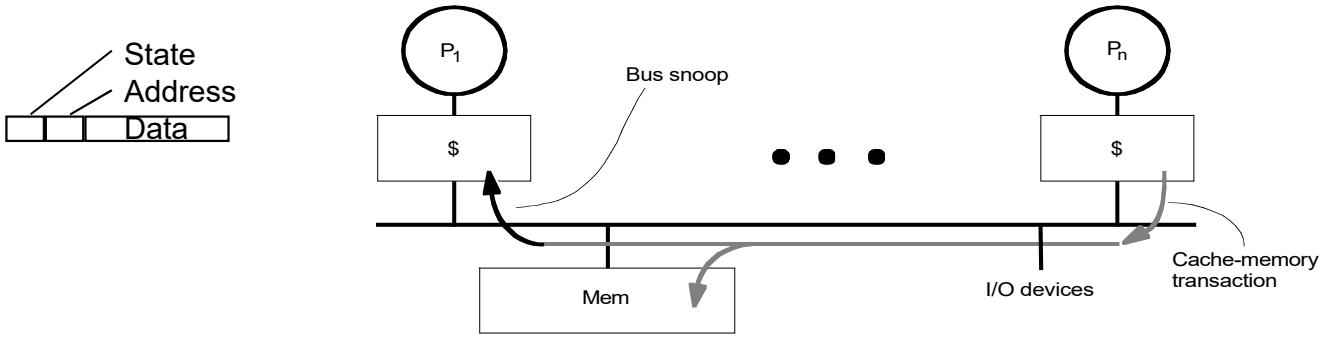
Bus-based Cache Coherence

- All of (a), (b), (c) done through broadcast on bus
  - faulting processor sends out a “search”
  - others respond to the search probe and take necessary action
- Could do it in scalable network too
- Conceptually simple, but broadcast doesn’t scale
  - on bus, bus bandwidth doesn’t scale
  - on scalable network, every fault leads to at least p network transactions
- Scalable coherence:
  - can have same cache states and state transition diagram
  - different mechanisms to manage protocol

Bus-based Cache-Coherent Architecture

- Bus Transactions
  - Single set of wires connect several devices
  - Bus protocol: arbitration, command/addr, data
  - Every device observes every transaction
- Cache block state transition diagram
  - FSM (Finite Series Model) specifying how disposition of block changes
    - invalid, valid, dirty
  - *Snoopy protocol*
- Basic Choices
  - Write-through vs Write-back
  - Invalidate vs. Update

Snoopy Cache-Coherency Protocols



- Bus is a broadcast medium
- Caches know what they have
- Cache controller "snoops" all transactions on shared bus
  - relevant transaction if for a block its cache contains
  - take action to ensure coherence
    - invalidate, update, or supply value
  - depends on state of the block and the protocol

Basic Snoopy Protocols

- Write Invalidate :
  - Multiple readers, single writer
  - Write to shared data: an invalidate is sent to all caches
  - Read Miss:
    - Write-through: memory is always up-to-date
    - Write-back: snoop in caches to find most recent copy
- Write Broadcast (typically write through):
  - Write to shared data: broadcast on bus, snoop and update
- Write serialization: bus serializes requests!
- Write Invalidate versus Broadcast

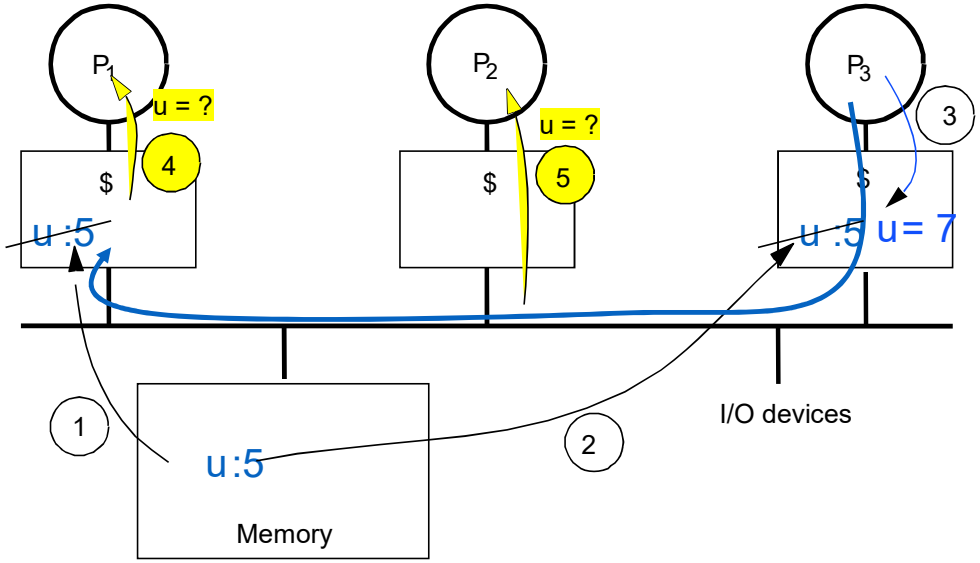
Snooping Cache Variations

Basic Protocol	Berkeley Protocol	Illinois Protocol	MESI protocol
Exclusive Shared Invalid	Owned Exclusive Owned Shared Shared Invalid	Private Dirty Private Clean Shared Invalid	Modified (private,!=Memory) Exclusive (private,=Memory) Shared (shared,=Memory) Invalid

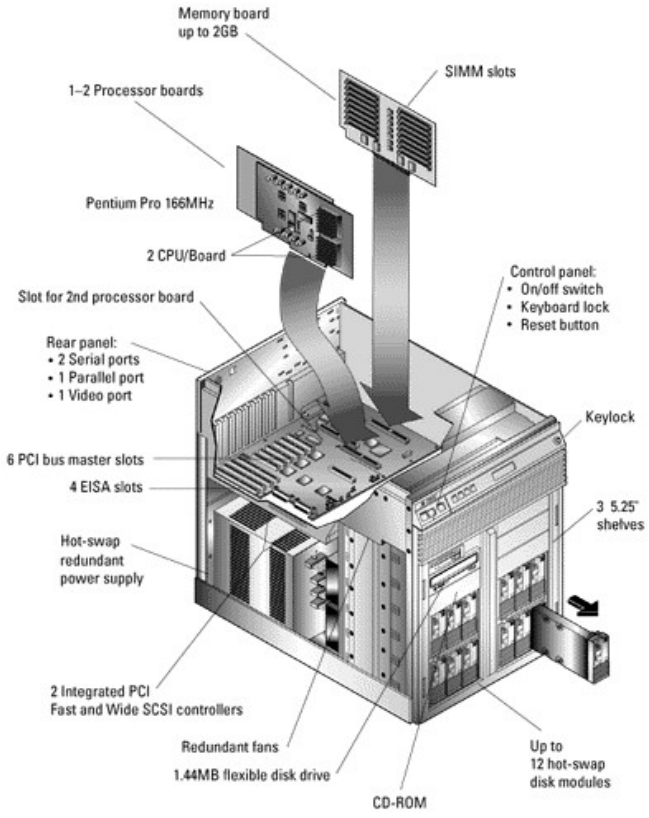
Owner can update via bus invalidate operation
Owner must write back when replaced in cache

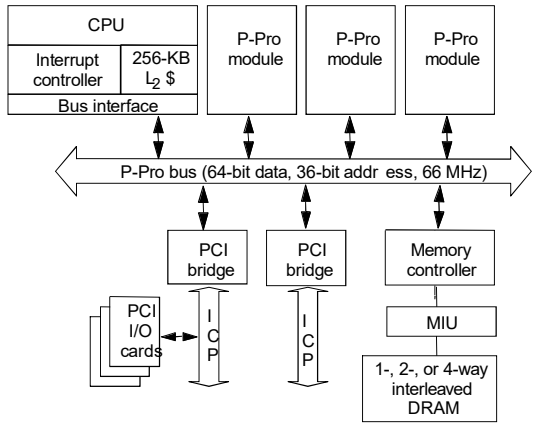
If read sourced from memory, then Private Clean
if read sourced from other cache, then Shared
Can write in cache if held private clean or dirty

Example: Write-back Invalidate



Intel Pentium Pro Quad Processor



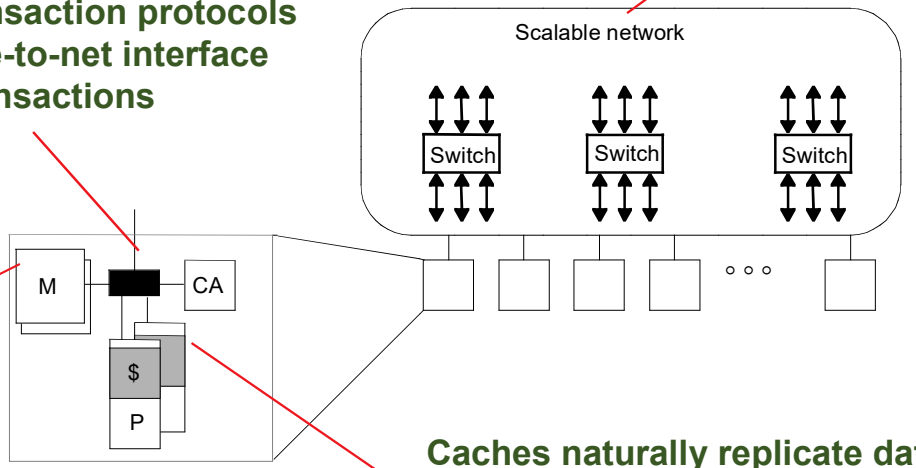


- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth
- 1995 - 1998

Context for Scalable Cache Coherence

Scalable Networks
- many simultaneous transactions

Realizing programming models through net transaction protocols
- efficient node-to-net interface
- interprets transactions

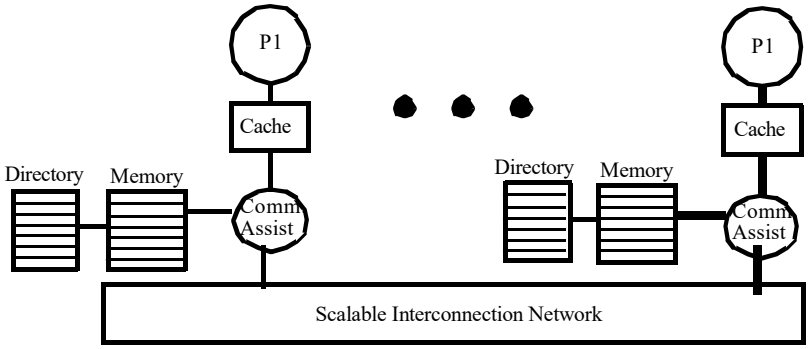


Scalable distributed memory

Caches naturally replicate data
- coherence through bus
- snooping protocols
- consistency

Need cache coherence protocols that scale!
- no broadcast or single point of order

Generic Solution: Directories

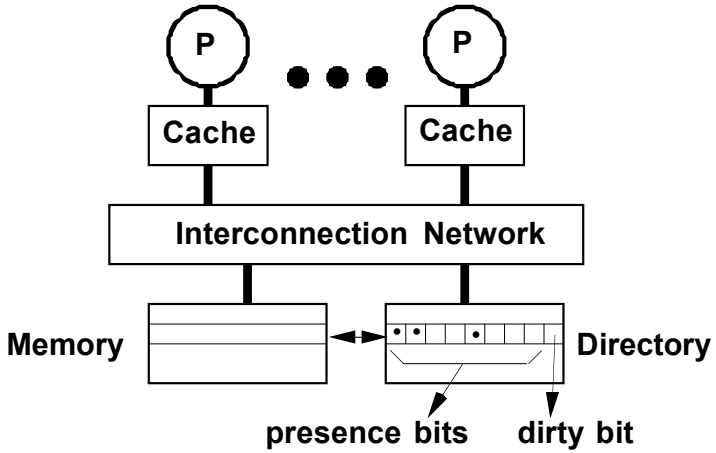


- Maintain state vector explicitly
  - associate with memory block
  - records state of block in each cache
- On miss, communicate with directory
  - determine location of cached copies
  - determine action to take
  - conduct protocol to maintain coherence

Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

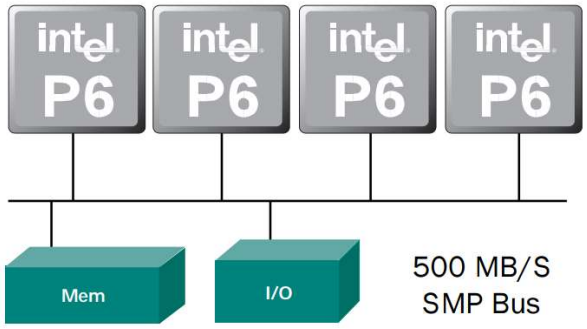
Basic Operation of Directory

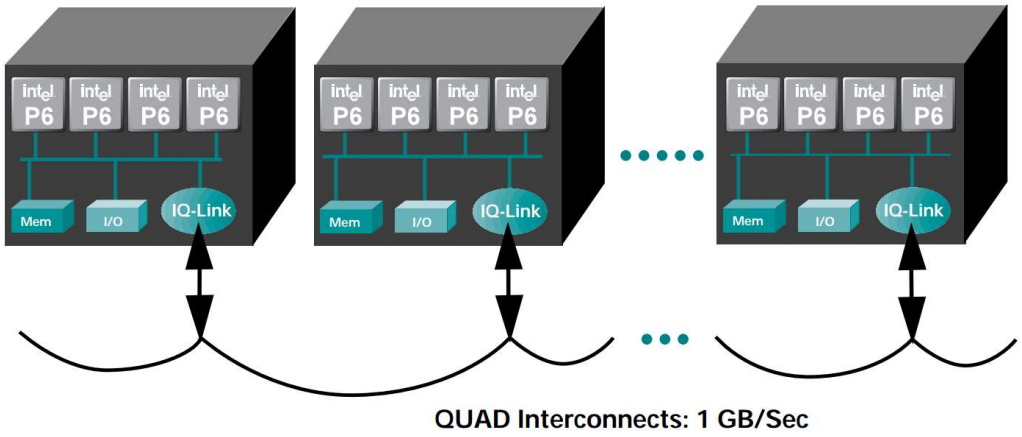


- k processors
- Each cache-block in memory
  - k presence bits and 1 dirty bit
- Each cache-block in cache
  - 1 valid bit and 1 dirty (owner) bit

- Read from memory
  - dirty bit OFF
  - dirty bit ON
- Write to memory
  - dirty bit OFF

Sequent NUMA-Q



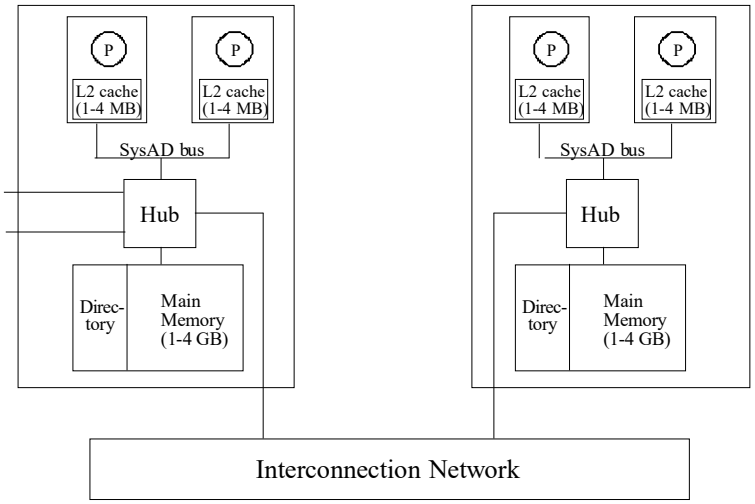


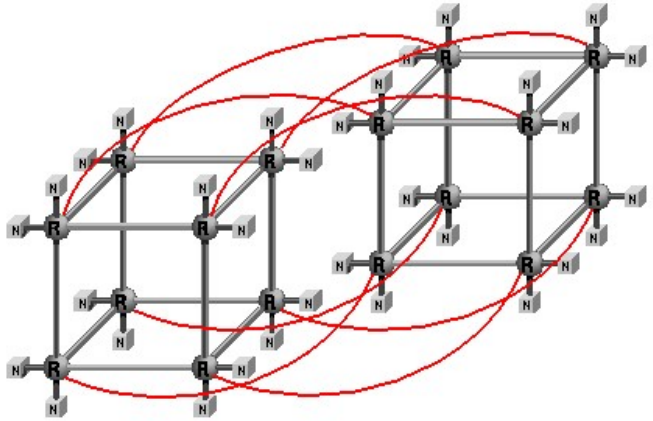
- Ring-based SCI (Scalable Coherent Interface) network
  - 1 GB/second
  - Built-in coherency
- Commodity SMPs as building blocks
  - Extend coherency mechanism
- Split transaction bus
- 1996

https://parallel.ru/sites/default/files/ftp/computers/sequent/Numa_Arch_wp.pdf

SGI Origin 2000

- Scalable shared memory multiprocessor
- MIPS R10000 CPUs @ 250MHz
- NUMAlink router
- Directory-based cache coherency (MESI)
- ASCI Blue Mountain (at LANL)
  - 3.072 TFLOPS
  - #2 in June 1999



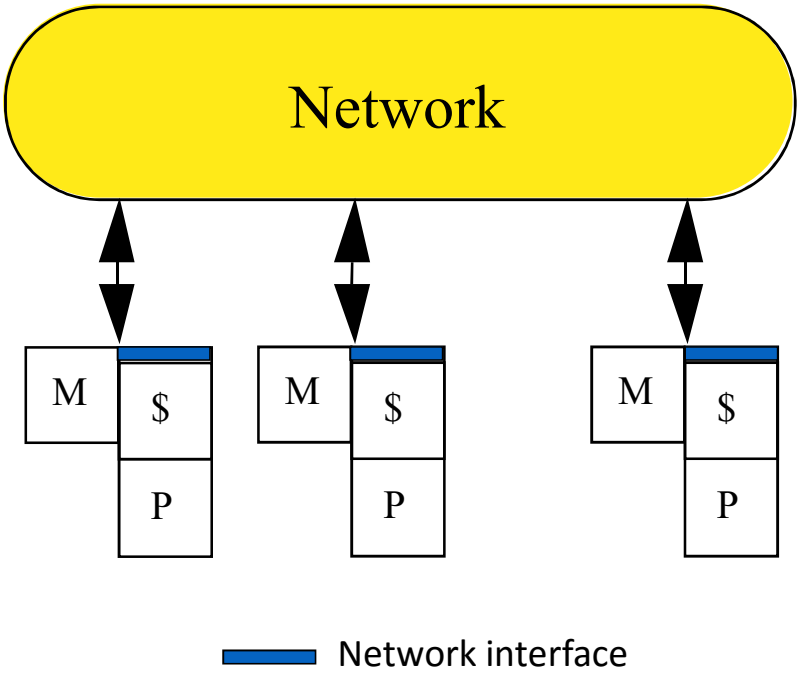


Distributed Memory Multiprocessors

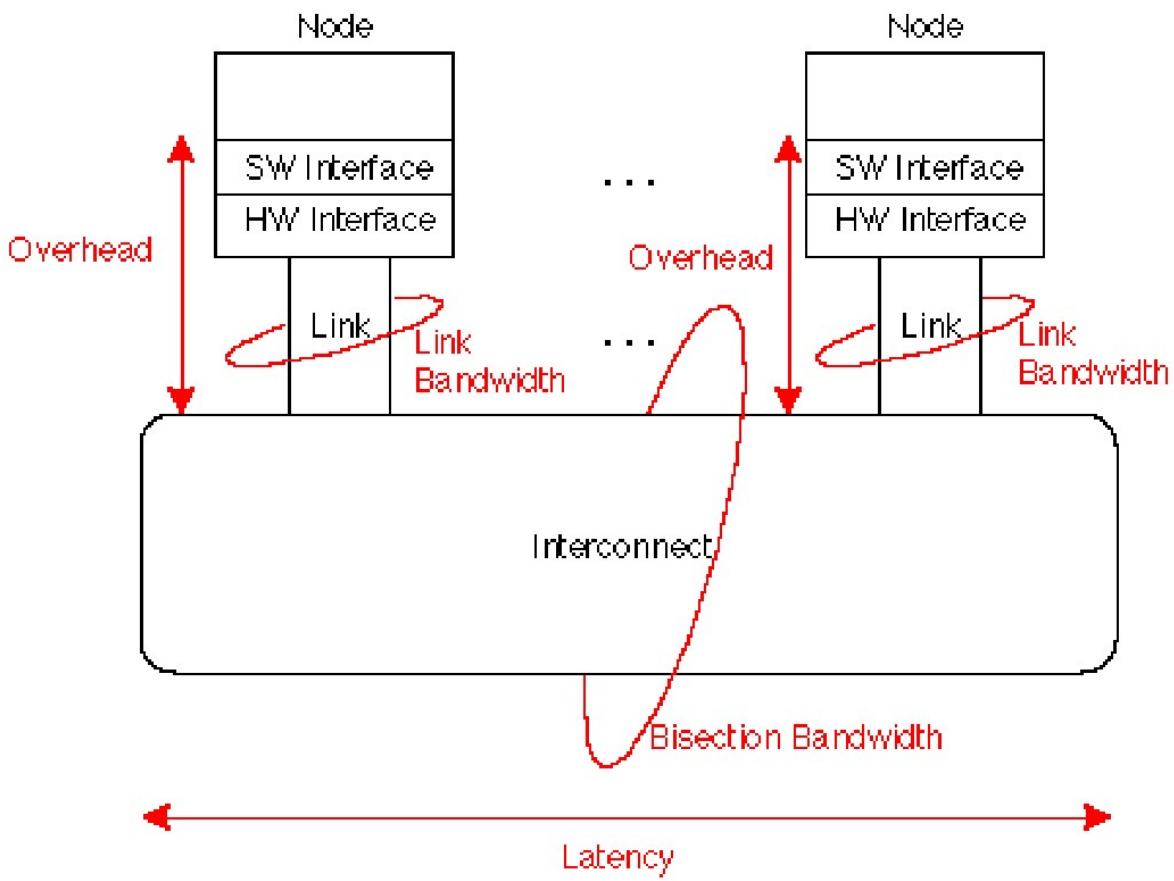
- Each processor has a local memory
  - Physically separated memory address space
- Processors must communicate to access non-local data
  - Message communication (message passing)
    - *Message passing architecture*
  - Processor interconnection network
- Parallel applications must be partitioned across
  - Processors: execution units
  - Memory: data partitioning
- Scalable architecture
  - Small incremental cost to add hardware (cost of node)

Distributed Memory Architecture

- Nodes are complete computer systems
  - Including I/O
- Nodes communicate via interconnection network
  - Standard networks
  - Specialized networks
- Network interfaces
  - Communication integration
- Easier to build



Network Performance Measures



Overhead: latency of interface vs. Latency: network

Performance Metrics: Latency and Bandwidth

- Bandwidth
  - Need high bandwidth in communication
  - Match limits in network, memory, and processor
  - Network interface speed vs. network bisection bandwidth
- Latency
  - Performance affected since processor may have to wait
  - Harder to overlap communication and computation
  - Overhead to communicate is a problem in many machines
- Latency hiding
  - Increases programming system burden
  - Examples: communication/computation overlaps, prefetch

Scalable, High-Performance Interconnect

- Interconnection network is core of parallel architecture
- Requirements and tradeoffs at many levels
  - Elegant mathematical structure
  - Deep relationship to algorithm structure
  - Hardware design sophistication
- Little consensus
  - Performance metrics?
  - Cost metrics?
  - Workload?
  - …

What Characterizes an Interconnection Network?

- Topology (what)
  - Interconnection structure of the network graph
- Routing Algorithm (which)
  - Restricts the set of paths that messages may follow
  - Many algorithms with different properties
- Switching Strategy (how)
  - How data in a message traverses a route
  - *circuit switching* vs. *packet switching*
- Flow Control Mechanism (when)
  - When a message or portions of it traverse a route
  - What happens when traffic is encountered?

Topological Properties

- Routing distance
  - Number of links on route from source to destination
- Diameter
  - Maximum routing distance
- Average distance
- Partitioned network
  - Removal of links resulting in disconnected graph
  - Minimal cut
- Scaling increment
  - What is needed to grow the network to next valid degree

Interconnection Topologies

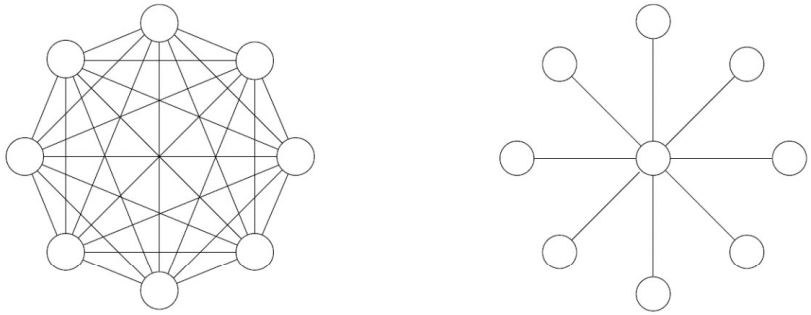


Figure 2.14 (a) A completely-connected network of eight nodes; (b) a Star connected network of nine nodes.

Introduction to Parallel Computing, Ananth Grama, et al., Addison Wesley, 2nd Ed., 2003

Interconnection Topologies (cont'd)

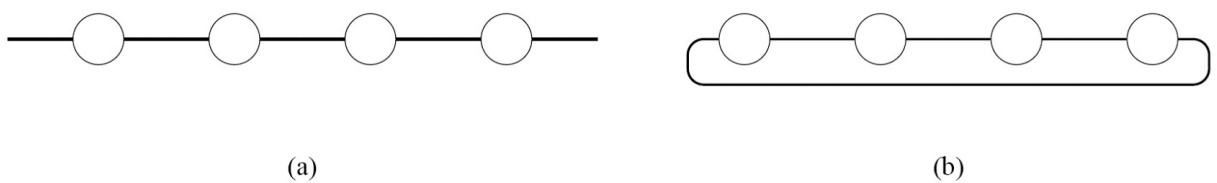


Figure 2.15 Linear arrays: (a) with no wraparound links; (b) with wraparound link.

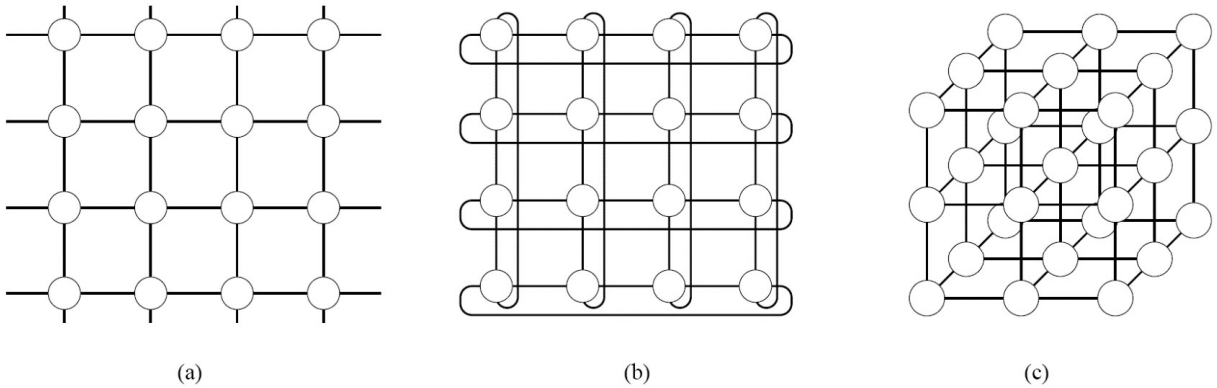
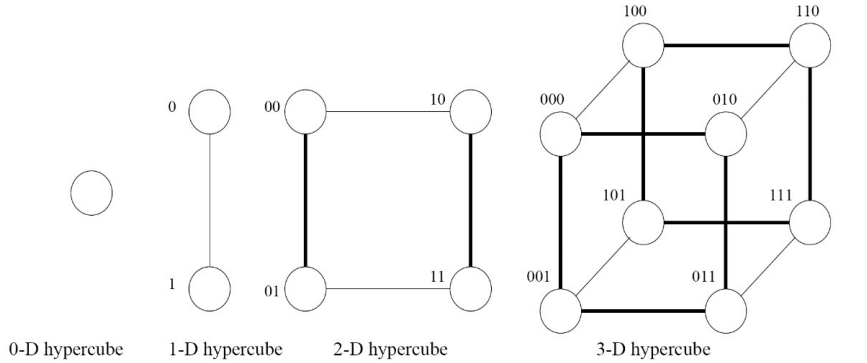


Figure 2.16 Two and three dimensional meshes: (a) 2-D mesh with no wraparound; (b) 2-D mesh with wraparound link (2-D torus); and (c) a 3-D mesh with no wraparound.

Introduction to Parallel Computing, Ananth Grama, et al., Addison Wesley, 2nd Ed., 2003

Interconnection Topologies (cont'd)



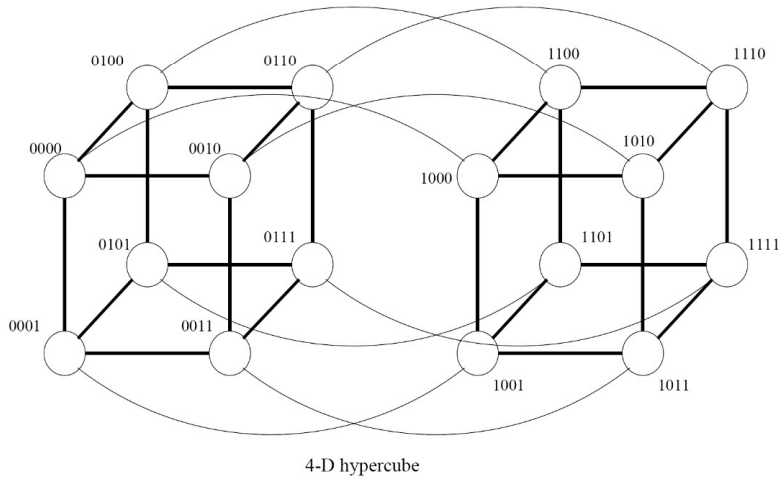


Figure 2.17 Construction of hypercubes from hypercubes of lower dimension.

Introduction to Parallel Computing, Ananth Grama, et al., Addison Wesley, 2nd Ed., 2003

Interconnection Topologies (cont'd)

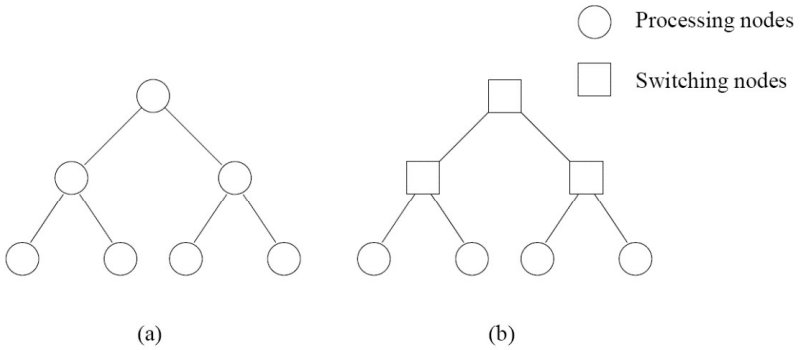


Figure 2.18 Complete binary tree networks: (a) a static tree network; and (b) a dynamic tree network.

Introduction to Parallel Computing, Ananth Grama, et al., Addison Wesley, 2nd Ed., 2003

Interconnection Topologies (cont'd)

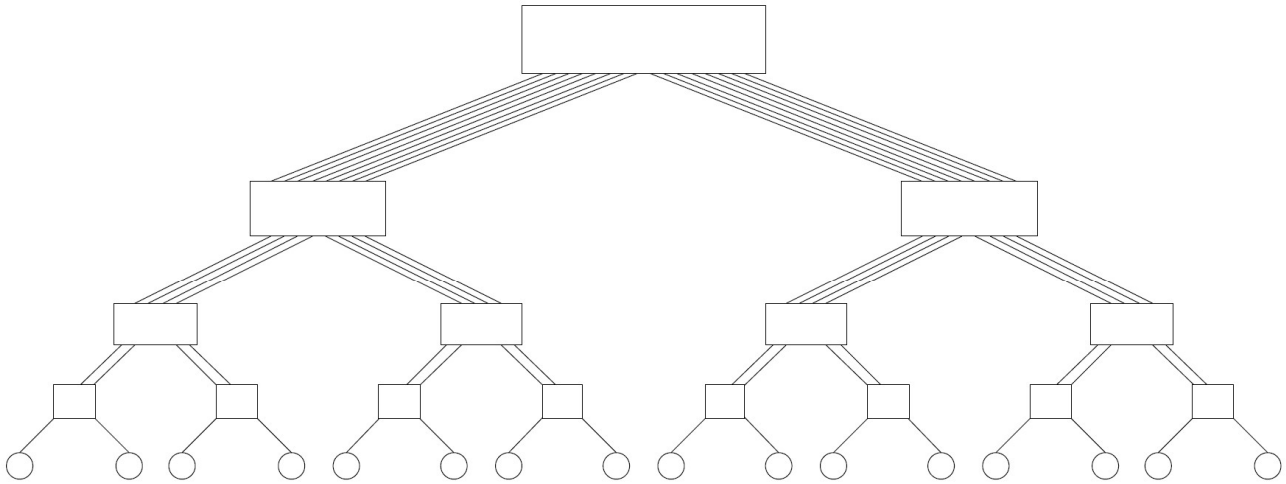


Figure 2.19 A fat tree network of 16 processing nodes.

Introduction to Parallel Computing, Ananth Grama, et al., Addison Wesley, 2nd Ed., 2003

Interconnection Topologies (cont'd)

Table 2.1 A summary of the characteristics of various static network topologies connecting p nodes.

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	1	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	1	$p-1$
Linear array	$p-1$	1	1	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p\log p)/2$
Wraparound k-ary d-cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	$2d$	dp

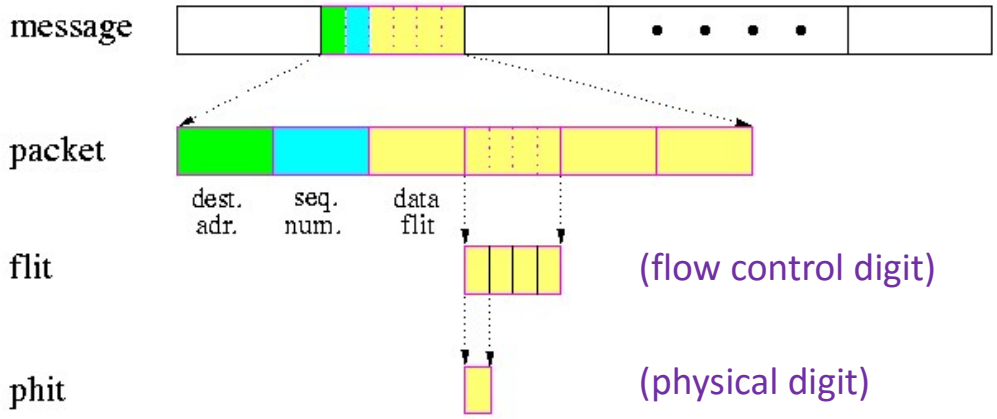
Introduction to Parallel Computing, Ananth Grama, et al., Addison Wesley, 2nd Ed., 2003

Communication Performance

- **$Time(n)_{s\text{-}d}$ = overhead + routing delay + channel occupancy + contention delay**

- **occupancy = $(n + n_h) / b$**

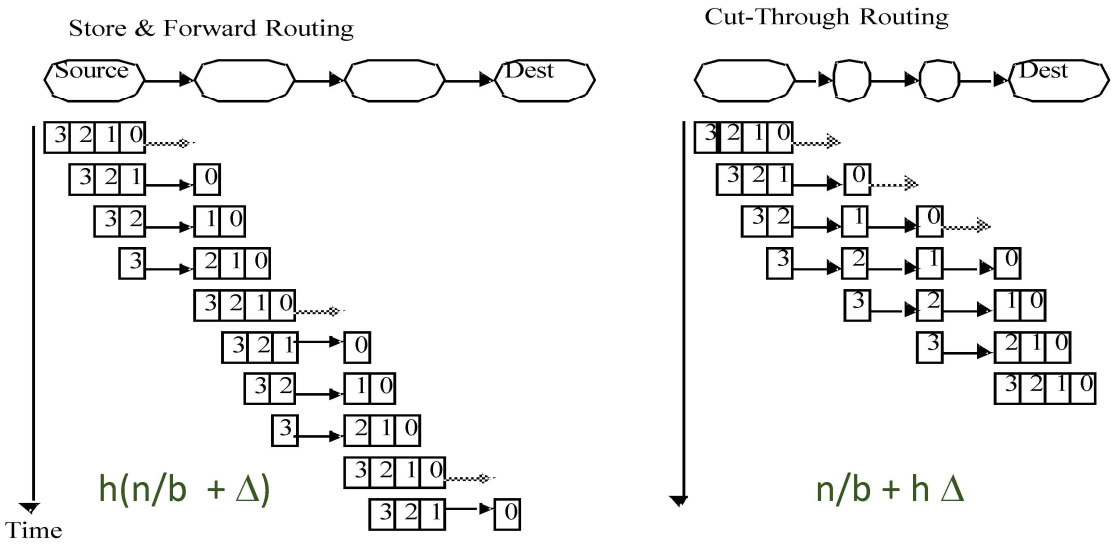
  n = size of message
  n_h = size of header
  b = bitrate of communication link



- What is the routing delay?
- Does contention occur and what is the cost?

Store-and-Forward vs. Cut-Through Routing

Flow Control:



- ***Store-and-Forward***: head flits waits at router until entire packet is buffered before being forwarded to the next hop
- ***Cut-Though***: flits can proceed to next hop before tail flit has been received by current router; but only if next router has enough buffer space for entire packet
- ***Wormhole***: just like cut-through, but with buffers allocated per flit (not channel)

Message Passing Model

- Hardware maintains send and receive message buffers
- Send message (synchronous)
  - Build message in local message send buffer
  - Specify receive location (processor id)
  - Initiate send and wait for receive acknowledge
- Receive message (synchronous)
  - Allocate local message receive buffer
  - Receive message byte stream into buffer
  - Verify message (e.g., checksum) and send acknowledge
- Memory to memory copy with acknowledgement and pairwise synchronization

Advantages of Shared Memory Architectures

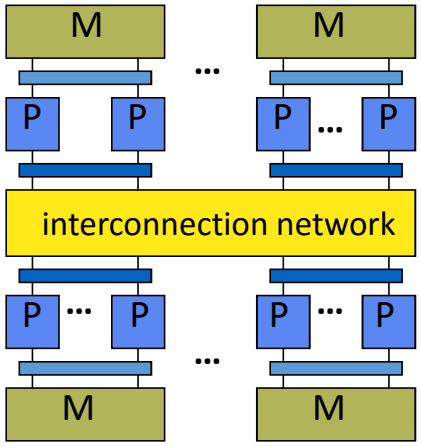
- Compatibility with SMP hardware
- Ease of programming when communication patterns are complex or vary dynamically during execution
- Ability to develop applications using familiar SMP model, attention only on performance critical accesses
- Lower communication overhead, better use of BW for small items, due to implicit communication and memory mapping to implement protection in hardware, rather than through I/O system
- HW-controlled caching to reduce remote communication by caching of all data, both shared and private

Advantages of Distributed Memory Architectures

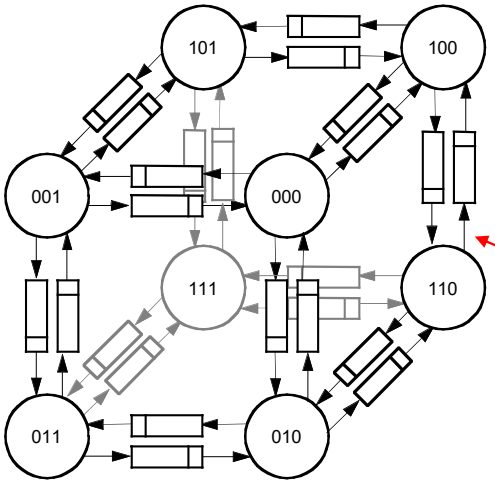
- The hardware can be simpler (especially versus NUMA) and is more scalable
- Communication is explicit and simpler to understand
- Explicit communication focuses attention on costly aspect of parallel computation
- Synchronization is naturally associated with sending messages, reducing the possibility for errors introduced by incorrect synchronization
- Easier to use sender-initiated communication, which may have some advantages in performance

Clusters of SMPs

- Clustering
  - Integrated packaging of nodes
- Motivation
  - Ammortize node costs by sharing packaging and resources
  - Reduce network costs
  - Reduce communications bandwidth requirements
  - Reduce overall latency
  - More parallelism in a smaller space
  - Increase node performance
- Scalable parallel systems today are built as SMP clusters



Caltech Cosmic Cube



FIFO on each link
Store and Forward

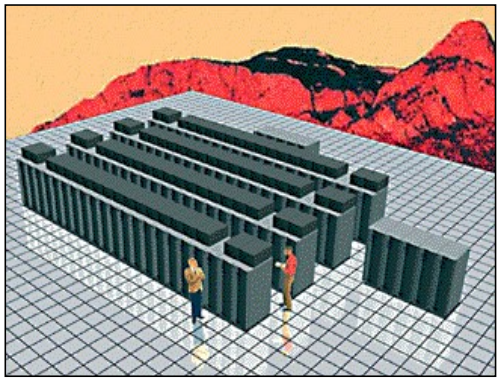
Compute nodes

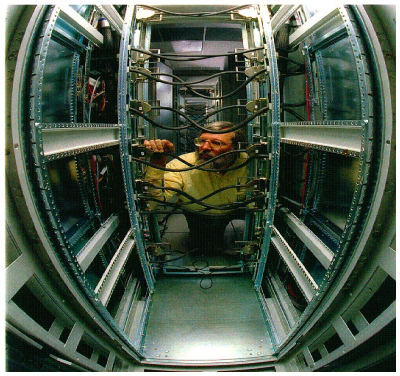
- 1st distributed memory message passing system
- Developed by Charles Seitz and Geoffrey Fox from 1981 onward
- 64 Intel 8086/8087 processors
- 6-D hypercube network
- Inspired Intel iPSC in 1980s and 1990s

http://groups.csail.mit.edu/cag/PSRG/papers/p22-seitz.pdf

ASCI Red

- Based on Intel Paragon
- 1st Supercomputer to achieve over 1 teraflops
- 7,264 Pentium Pro processors @ 200 MHz in 06/1997, R_{max} = 1.068 TFLOPS
- 9,152 Pentium Pro processors @ 200 MHz in 11/1997, R_{max} = 1.338 TFLOPS
- 9,632 Pentium II Overdrive processors @ 333 MHz in 11/1999, R_{max} = 2.379 TFLOPS
- Network topology: 2 dimensional 2-plane mesh (38x32x2)
  - 2 compute nodes (4 processors) on a “Kestrel” board with a NIC
  - Link bandwidth: 800 MB/s
  - Bisection bandwidth: 51 GB/s
- #1 from June 1997 to June 2000



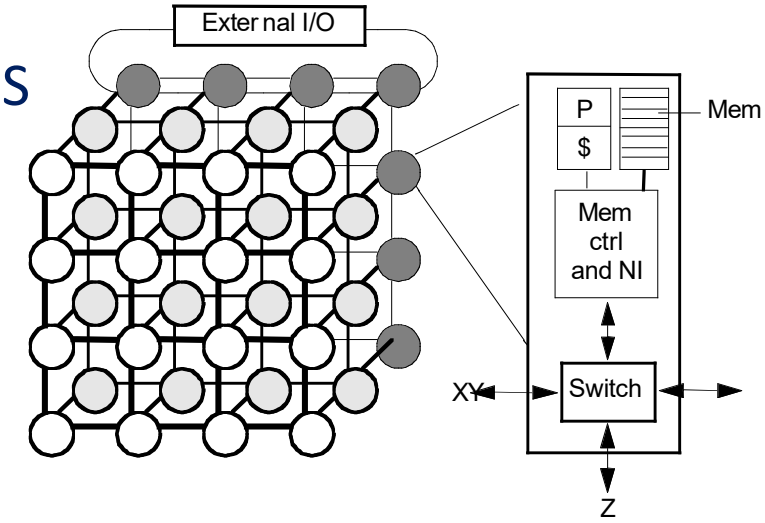


Beowulf Cluster

- A cluster of *commodity-grade* computers
- Originally referred to a cluster built in 1994 by Thomas Sterling and Donald Becker at NASA
- Similar to Berkeley NOW (Network of Workstations)
- *Commodity* Interconnects
  - Ethernet, Myrinet, SCI, InfiniBand, etc.
- *Commodity* software
  - Unix-like OS, such as Linux, BSD, Solaris, etc.
  - Cluster toolkit, such as Rocks, Warewulf, etc.
  - Libraries and programs, such as MPI, PVM, etc.

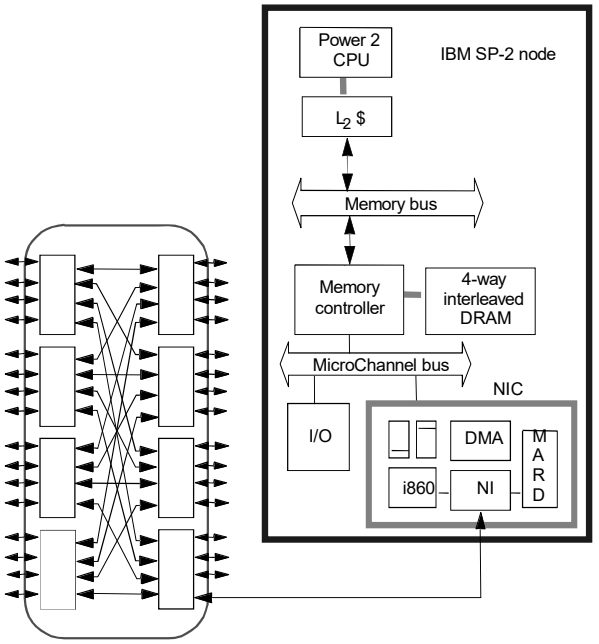
Cray T3E

- Launched in November 1995
- From 8 to 2,176 Processing Elements (PEs)
  - DEC Alpha processors @ 300, 450, 600 or 675 MHz
- 3D torus network
  - 6-way interconnect router with 480 MB/s in each direction
- 1st supercomputer to achieve over 1 TFLOPS running science simulation, in 1998
- Distributed memory machine
  - Message passing
  - SHMEM (Symmetric Hierarchical Memory)
    - *put* / *get* – one-sided operations



ASCI White

- IBM SP-2
- Made out of commercial RS/6000 workstations
  - 512 nodes
  - 16 PowerPC processors @ 375 MHz per node
  - 8,192 processors in total
- Network interface integrated in I/O bus
- SP network very advanced
  - Formed from 8-port switches
- R_{max} = 7.226 TFLOPS, R_{peak} = 12.3 TFLOPS
- #1 from 11/2000 – 11/2001
- 3MW for computer & 3MW for cooling
- Costed $110 million



NASA Columbia

- 20 interconnected SGI Altix 3700 nodes, each with:
  - 512 Itanium 2 2-core processors @ 1.6 GHz
  - 1TB memory
- NUMAflex architecture
  - NUMAlink “fat tree” network
  - 20TB of single shared memory!!!
- Nodes were also connected with InfiniBand SDR and DDR cabling
- R_{max} = 51.87 TFLOPS, R_{peak} = 60.96 TFLOPS
- #2 in November 2004

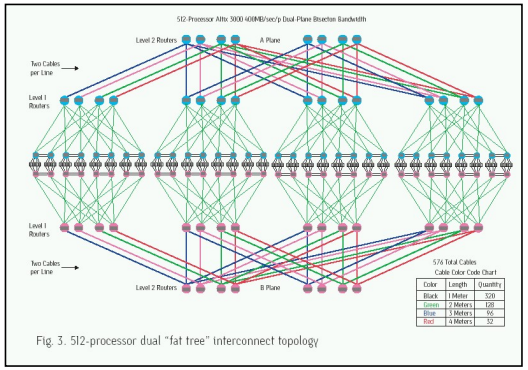
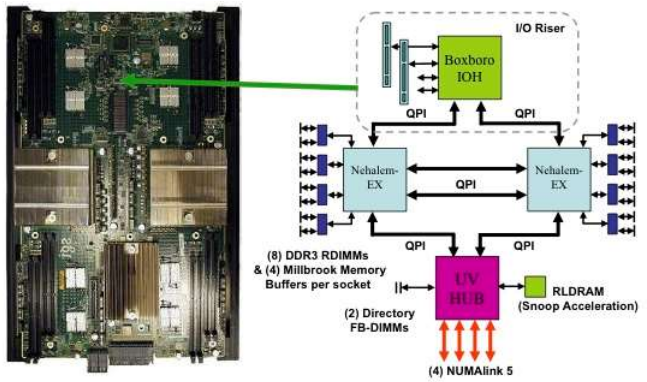
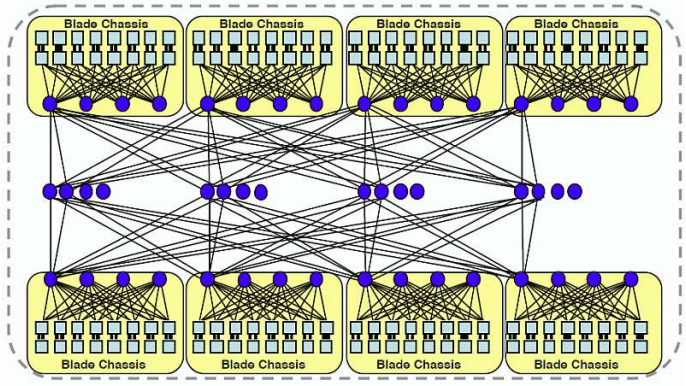


Fig. 3. 512-processor dual “fat tree” interconnect topology

SGI UV

- Latest generation scalable shared memory architecture
- Scaling up to 256 sockets
  - Intel Xeon E5-4600 v3 processors
- Architectural provisioning for up to 262,144 cores
- Up to 64TB of cache-coherent shared memory in a single system image (SSI)
- NUMAlink 6 interconnect (6.7GB/s bidirectional)

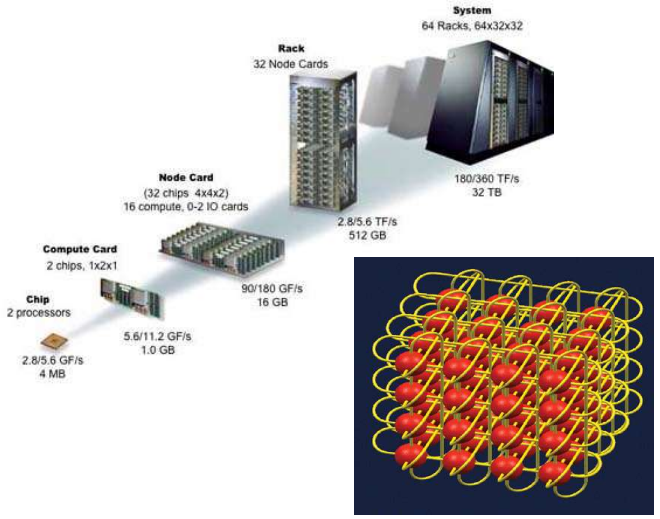




https://www.sgi.com/pdfs/4555.pdf

LLNL Blue Gene/L

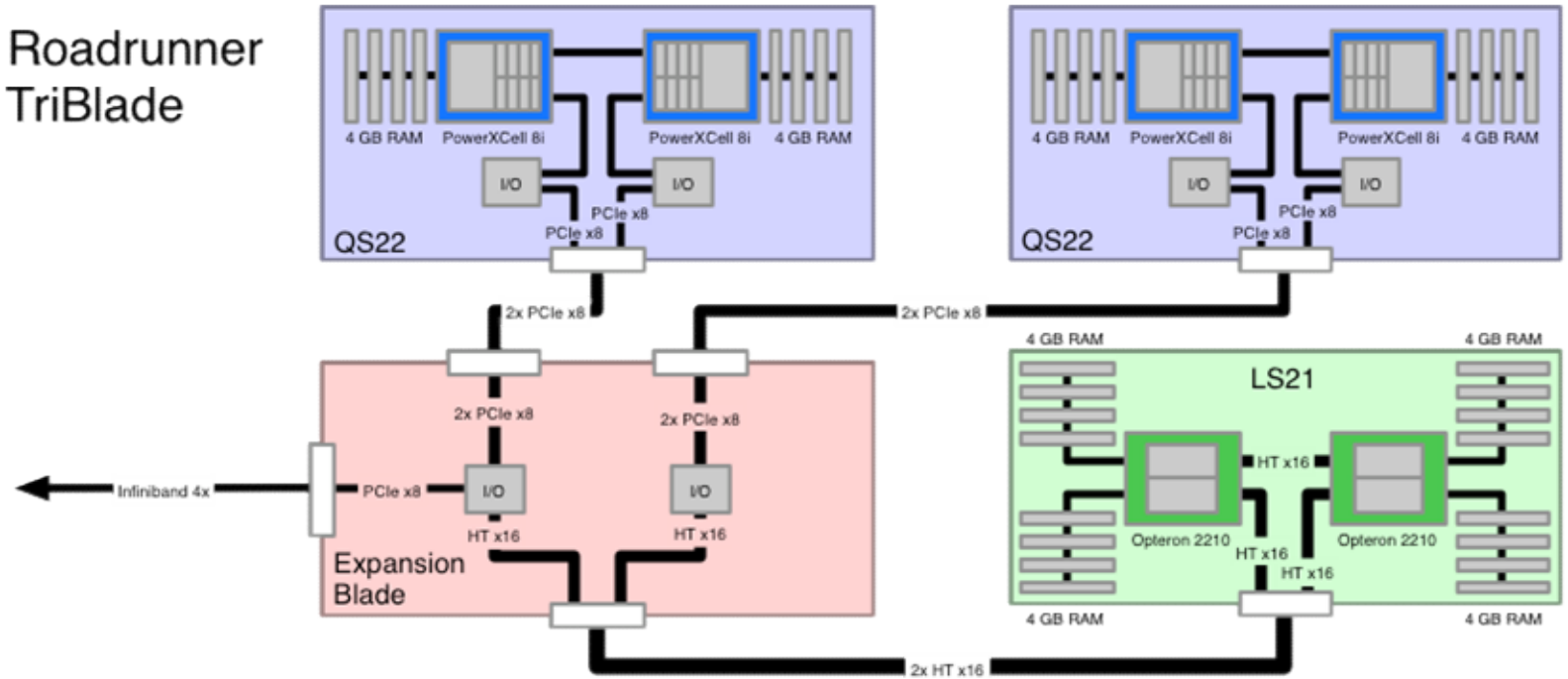
- Initially 65,536 dual-processor compute nodes
- Later upgraded to 106,496 compute nodes, each with
  - Two PowerPC 440 2-core processors @ 700MHz
  - Two working modes: co-processor mode & virtual-node mode
  - Double “hummer” floating point unit (FPU) per core
- One I/O node every 32 compute nodes
- 3D torus network (initially 32x32x64)
- Global reduction tree
- Global barrier and interrupt networks
- Scalable tree network for I/O
- R_{max} = 478.2 TFLOPS, R_{peak} = 596.4 TFLOPS
- #1 from November 2004 to June 2008



IBM Roadrunner

- 1st petaflops Supercomputer
- Hybrid architecture:
  - Opteron cluster with Cell accelerators
- 3,240 TriBlades, each with
  - One LS21 Opteron blade, with two dual-core AMD Opteron 2210 processors @1.8 GHz and 16GB memory
  - Two QS22 Cell blades, each with two PowerXCell 8i CPUs (@3.2GHz) and 8GB memory
    Peak performance = 102.4 GFLOPS per PowerXCell 8i
  - One expansion blade, connecting the two QS22 via four PCIe x8 links to the LS21
- 26 288-port Voltaire ISR 2012 InfiniBand 4x DDR switches
  - fat tree topology
- R_{max} = 1.026 PFLOPS, R_{peak} = 1.376 PFLOPS, Power = 2.345 MW
- #1 in June & November 2008

IBM Roadrunner TriBlade



Roadrunner: Hardware and Software Overview
http://www.redbooks.ibm.com/redpapers/pdfs/redp4477.pdf