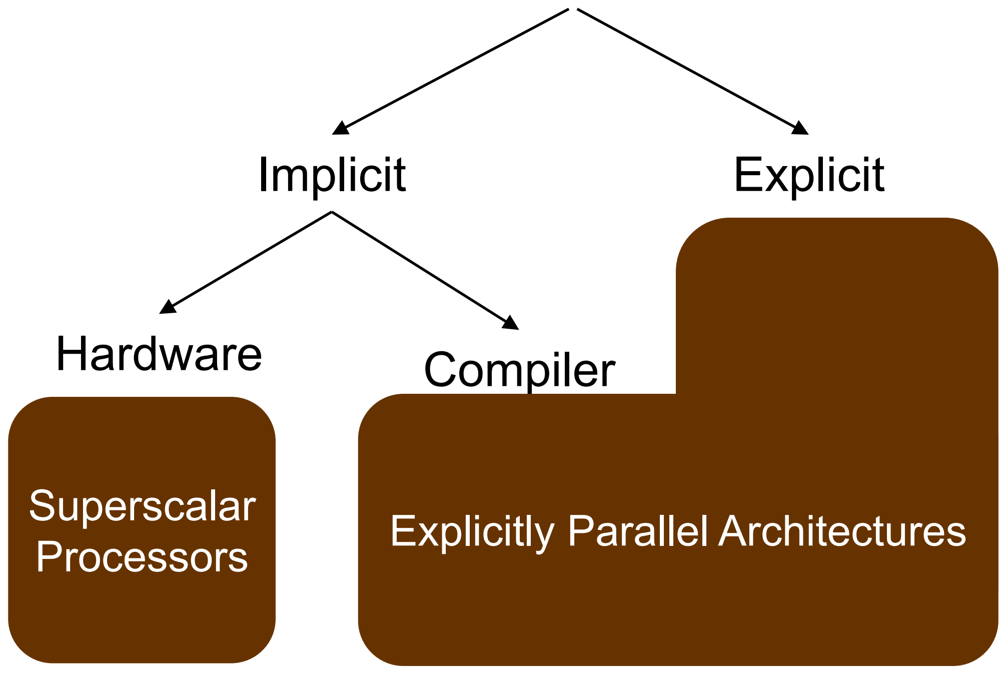
CSE 531(Spring 2023)

Introduction to Parallel Architectures

Implicit vs. Explicit Parallelism (Simplified View)



Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
  - Shared Instruction Processors
  - Shared Sequencer Processors
  - Shared Network Processors
  - Shared Memory Processors
  - Multicore Processors

GPUs will be discussed separately

Implicit Parallelism: Superscalar Processors

- Issue varying numbers of instructions per clock
  - statically scheduled
    - using compiler techniques
    - in-order execution
  - dynamically scheduled
    - Extracting ILP by examining 100’s of instructions
    - Scheduling them in parallel as operands become available
    - Rename registers to eliminate anti-dependences
    - Out-of-order (OOO) execution
    - Speculative execution

Pipelined Execution

IF: Instruction fetch ID : Instruction decode
EX : Execution/Memory Access WB : Write back

Cycles

Instruction #	1	2	3	4	5	6	7	8
Instruction i	IF	ID	EX	WB				
Instruction i+1		IF	ID	EX	WB			
Instruction i+2			IF	ID	EX	WB		
Instruction i+3				IF	ID	EX	WB	
Instruction i+4					IF	ID	EX	WB

Super-Scalar Execution

	Cycles						
Instruction type	1	2	3	4	5	6	7
Integer	IF	ID	EX	WB			
Floating point	IF	ID	EX	WB			
Integer		IF	ID	EX	WB		
Floating point		IF	ID	EX	WB		
Integer			IF	ID	EX	WB	
Floating point			IF	ID	EX	WB	
Integer				IF	ID	EX	WB
Floating point				IF	ID	EX	WB

2-issue super-scalar machine

Data Dependence and Hazards

- Instr_J is data dependent (aka true dependence) on $\text{Instr}_{I:}$

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- If two instructions are data dependent, they cannot execute simultaneously, be completely overlapped or execute in out-of-order
- If data dependence caused a hazard in pipeline, called a Read After Write (RAW) hazard

ILP and Data Dependencies, Hazards

- HW/SW must preserve program order: instructions would execute as if executed sequentially as determined by original source program
  - Dependences are a property of programs
    - They are independent of architecture
- Importance of the data dependencies
  - 1) indicate the possibility of a hazard
  - 2) determine order in which results must be calculated
  - 3) set an upper bound on how much parallelism can possibly be exploited
  - 4) determine pressure on hardware resources
- Goal: exploit parallelism by preserving program order only where it affects the outcome of the program

Name Dependence #1: Anti-dependence

- Name dependence (a.k.a. pseudo dependence): when 2 instructions use same register or memory location, called a name, but no flow of data between the instructions associated with that name; 2 versions of name dependence
- Instr$_J$ writes operand *before* Instr$_I$ reads it

```
I:  sub r4,r1,r3
J:  add r1,r2,r3
K:  mul r6,r1,r7
```

Called an "anti-dependence" by compiler folks
This results from reuse of the name "r1"

- If anti-dependence caused a hazard in the pipeline, called a Write After Read (WAR) hazard

Name Dependence #2: Output dependence

- Instr_J writes operand *before* Instr_I writes it.

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "output dependence" by compiler writers.
  This also results from the reuse of name "r1"
- If output-dependence caused a hazard in the pipeline, called a Write After Write (WAW) hazard

- Instructions involved in a name dependence can execute simultaneously if name used in instructions is changed so instructions do not conflict
  - Register renaming resolves name dependence for registers
  - Renaming can be done either by compiler or by HW

Control Dependencies

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order

```
    if p1 {
      S1;
    };
    if p2 {
      S2;
    }
```

- `S1` is control dependent on `p1`, and `S2` is control dependent on `p2` but not on `p1`.

- Control dependence need not be preserved
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, if can do so without affecting correctness of the program
- Speculative Execution

Speculation

- Greater ILP: Overcome control dependence by hardware speculating on outcome of branches and executing program as if guesses were correct

- Branch Prediction ⇒ fetch, issue, and execute instructions as if branch predictions were always correct

Speculation is Rampant in Modern Superscalars

- Different predictors
  - Branch Prediction
  - Value Prediction
  - Prefetching (address prediction)
    - Instructions and data

- Inefficient
  - Predictions can go wrong
  - Must flush out wrongly predicted data
  - While not impacting performance, it consumes power

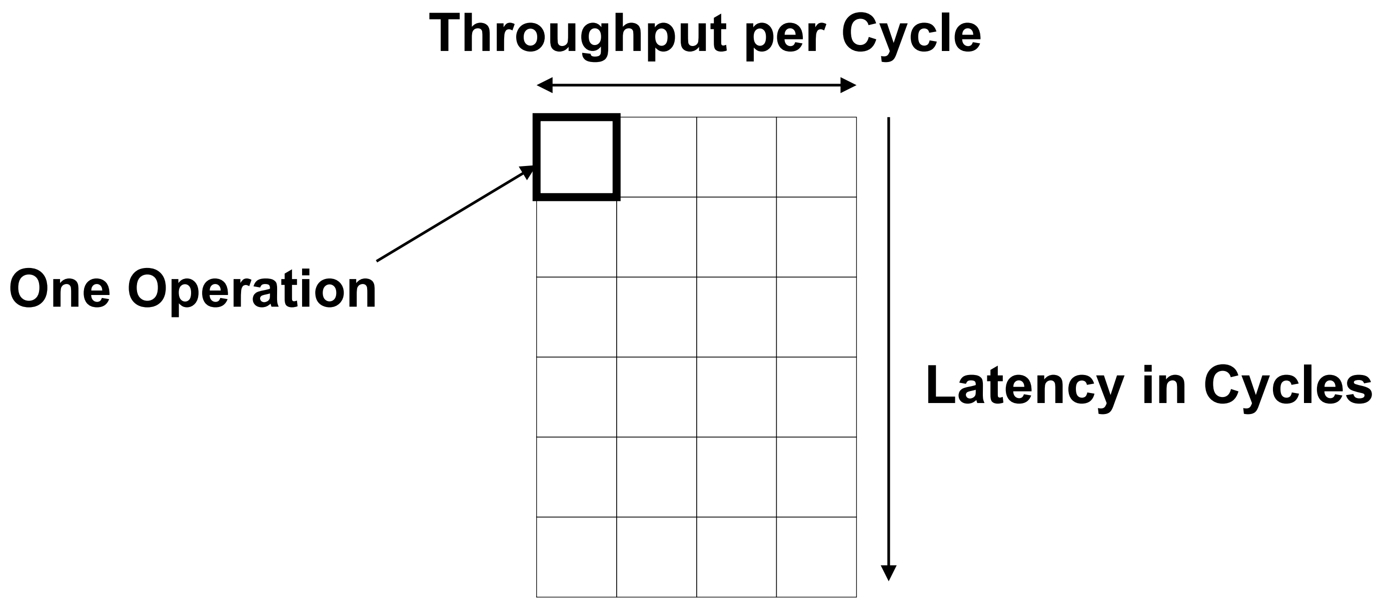
Outline

- Implicit Parallelism: Superscalar Processors
- **Explicit Parallelism**
  - Shared Instruction Processors
  - Shared Sequencer Processors
  - Shared Network Processors
  - Shared Memory Processors
  - Multicore Processors

Explicit Parallel Processors

- Parallelism is exposed to software
  - Compiler or Programmer
- Many different forms
  - Loosely coupled Multiprocessors to tightly coupled VLIW

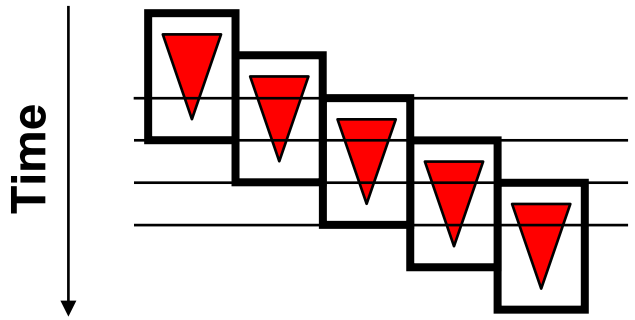
Little's Law



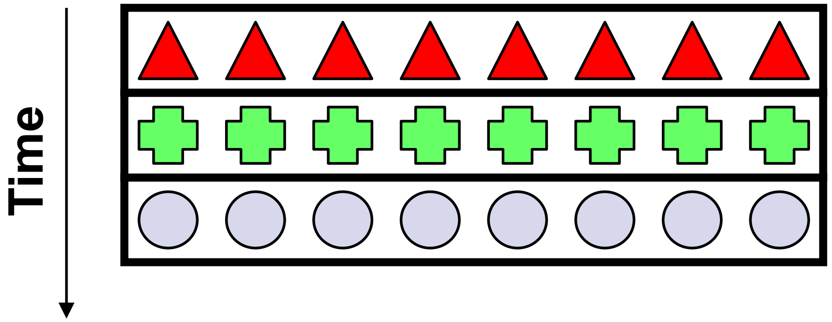
Parallelism = Throughput * Latency

- To maintain throughput T/cycle when each operation has latency L cycles, need T*L *independent* operations
- For fixed parallelism:
  - decreased latency allows increased throughput
  - decreased throughput allows increased latency tolerance

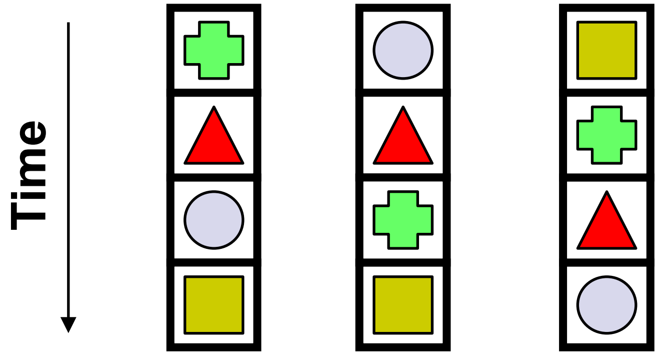
Types of Compute Parallelism



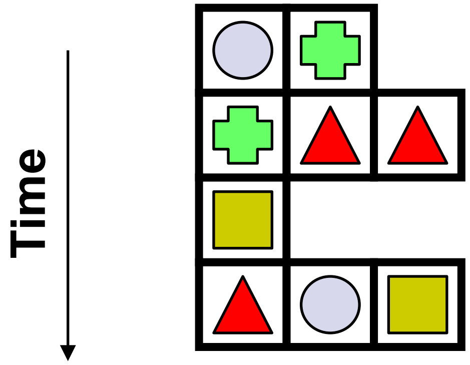
Pipelining



Data-Level Parallelism (DLP)



Thread-Level Parallelism (TLP)



Instruction-Level Parallelism (ILP)

Issues in Parallel Machine Design

- Communication/Data Sharing
  - how do parallel operations communicate/share data results?
- Synchronization
  - how are parallel operations coordinated?
- Resource Management
  - how are a large number of parallel tasks scheduled onto finite hardware?
- Scalability
  - how large a machine can be built?
  - how effectively can we utilize a large machine?

Flynn’s Classification (1966)

Broad classification of parallel computing systems based on number of instruction and data streams

- SISD: Single Instruction, Single Data
  - conventional uniprocessor
- SIMD: Single Instruction, Multiple Data
  - one instruction stream, multiple data paths
  - distributed memory SIMD (MPP, DAP, CM-1&2, Maspar)
  - shared memory SIMD (STARAN, vector computers)
- MIMD: Multiple Instruction, Multiple Data
  - message passing machines (Transputers, nCube, CM-5)
  - non-cache-coherent shared memory machines (BBN Butterfly, T3D)
  - cache-coherent shared memory machines (Sequent, Sun Starfire, SGI Origin)
- MISD: Multiple Instruction, Single Data
  - no commercial examples

Classification by the Level of Sharing (MIT)

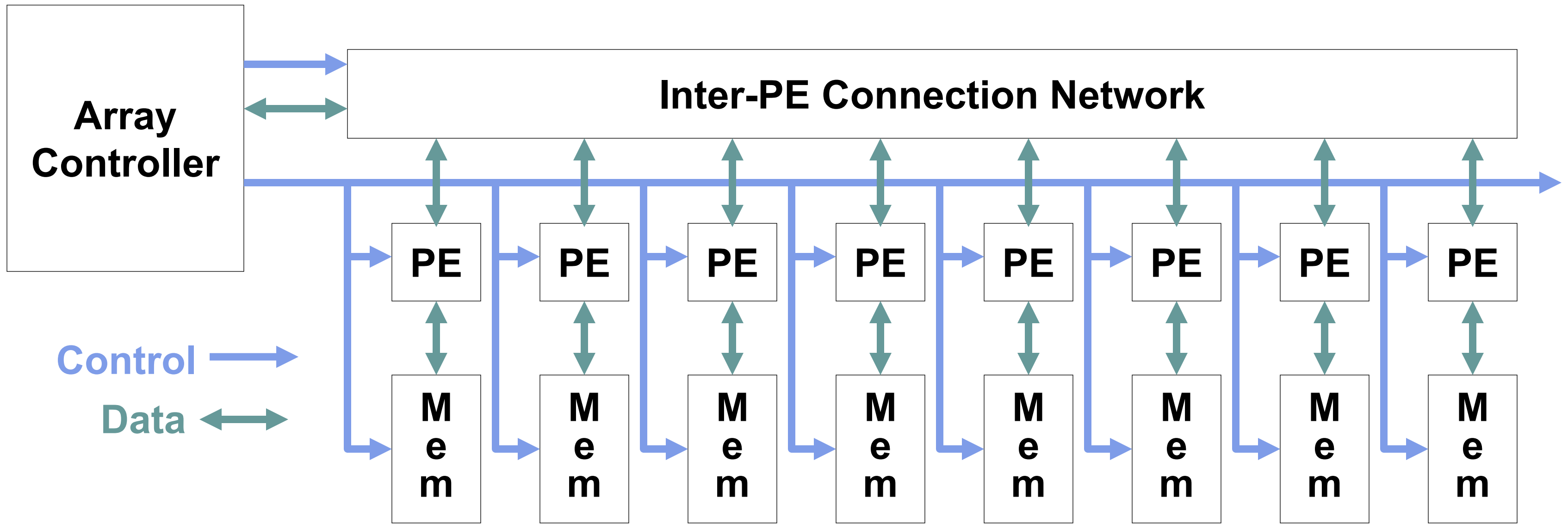
- Shared Instruction
- Shared Sequencer
- Shared Memory
- Shared Network

Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
  - **Shared Instruction Processors**
  - Shared Sequencer Processors
  - Shared Network Processors
  - Shared Memory Processors
  - Multicore Processors

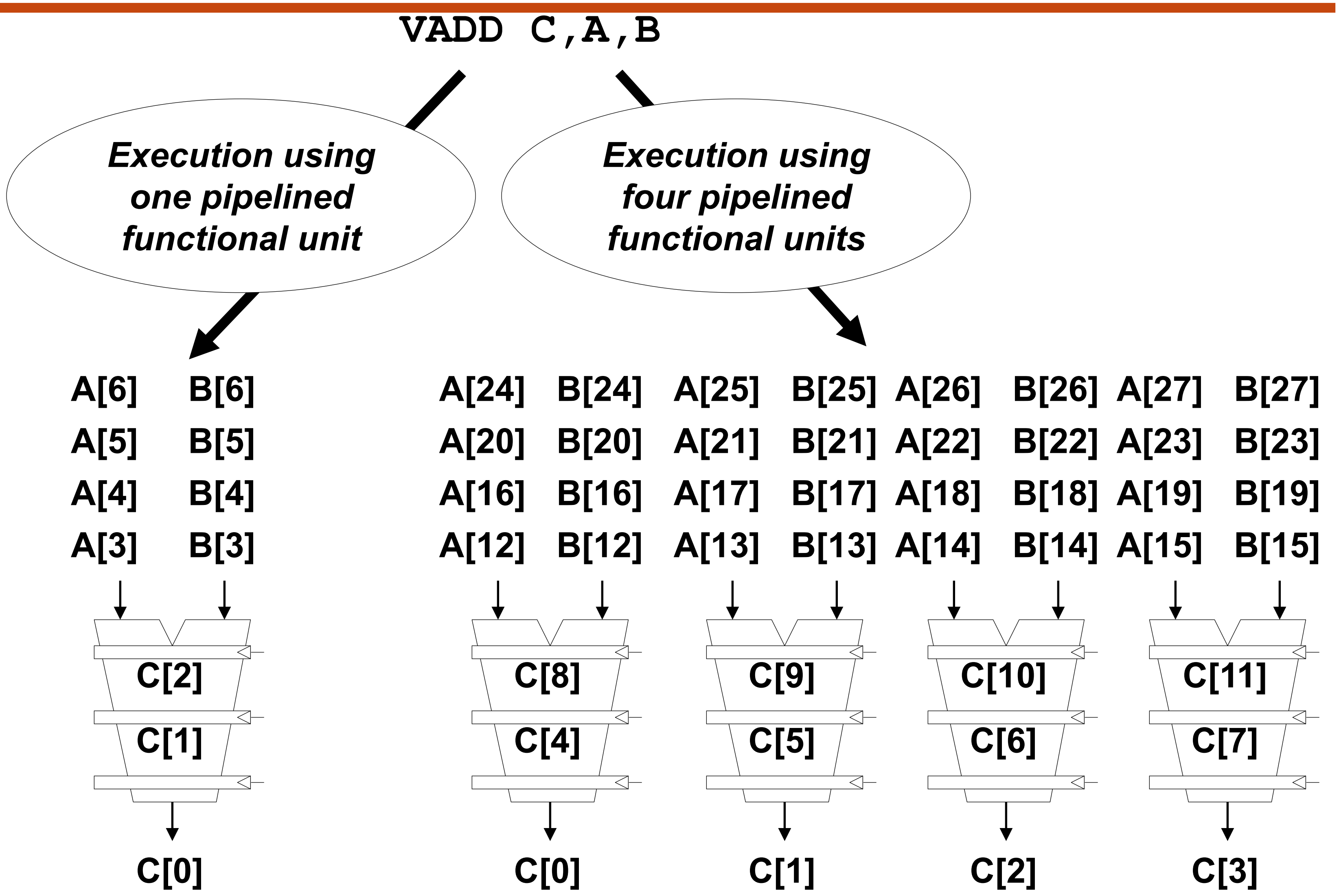
Shared Instruction: SIMD Architecture

- Central controller broadcasts instructions to multiple processing elements (PEs)



- **Only requires one controller for whole array**
- **Only requires storage for one copy of program**
- **All computations fully synchronized (lock-step)**

Vector Instruction Execution


VADD C,A,B
Execution using one pipelined functional unit
Execution using four pipelined functional units
A[6] B[6]
A[5] B[5]
A[4] B[4]
A[3] B[3]
C[2]
C[1]
C[0]
A[24] B[24] A[25] B[25] A[26] B[26] A[27] B[27]
A[20] B[20] A[21] B[21] A[22] B[22] A[23] B[23]
A[16] B[16] A[17] B[17] A[18] B[18] A[19] B[19]
A[12] B[12] A[13] B[13] A[14] B[14] A[15] B[15]
C[8] C[9] C[10] C[11]
C[4] C[5] C[6] C[7]
C[0] C[1] C[2] C[3]

Superword Level Parallelism (SLP)

- Applying vectorization idea at a small-scale
- Small amount of parallelism
  - Typically, 2 to 8-way
- Exists within basic blocks (a few lines of code)
- Uncovered with a simple analysis

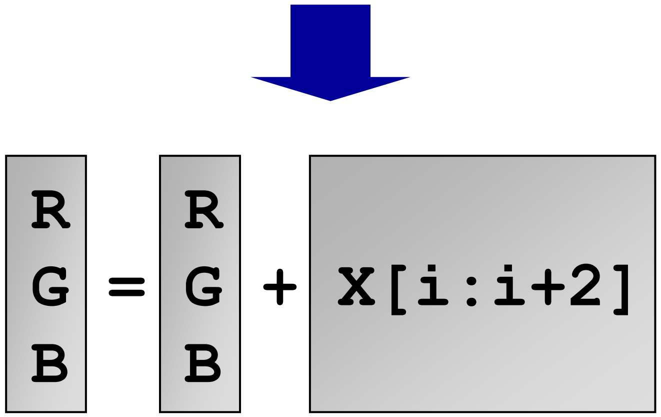
Independent ALU Ops

```
R = R + XR * 1.08327
G = G + XG * 1.89234
B = B + XB * 1.29835
```

$$\begin{bmatrix}R\\G\\B\end{bmatrix} = \begin{bmatrix}R\\G\\B\end{bmatrix} + \begin{bmatrix}XR\\XG\\XB\end{bmatrix} * \begin{bmatrix}1.08327\\1.89234\\1.29835\end{bmatrix}$$

Adjacent Memory References

```
R = R + X[i+0]
G = G + X[i+1]
B = B + X[i+2]
```



Vectorizable Loops

```
for (i=0; i<100; i+=1)
    A[i+0] = A[i+0] + B[i+0]
```

Vectorizable Loops

```
for (i=0; i<100; i+=4)
    A[i+0] = A[i+0] + B[i+0]
    A[i+1] = A[i+1] + B[i+1]
    A[i+2] = A[i+2] + B[i+2]
    A[i+3] = A[i+3] + B[i+3]
```

$$\Downarrow$$

```
for (i=0; i<100; i+=4)
    A[i:i+3] = B[i:i+3] + C[i:i+3]
```

Partially Vectorizable Loops

```
for (i=0; i<16; i+=1)
    L = A[i+0] - B[i+0]
    D = D + abs(L)
```

Partially Vectorizable Loops

```
for (i=0; i<16; i+=2)
    L = A[i+0] - B[i+0]
    D = D + abs(L)
    L = A[i+1] - B[i+1]
    D = D + abs(L)
```

```
for (i=0; i<16; i+=2)
    L0
    L1  = A[i:i+1] - B[i:i+1]
     D = D + abs(L0)
     D = D + abs(L1)
```

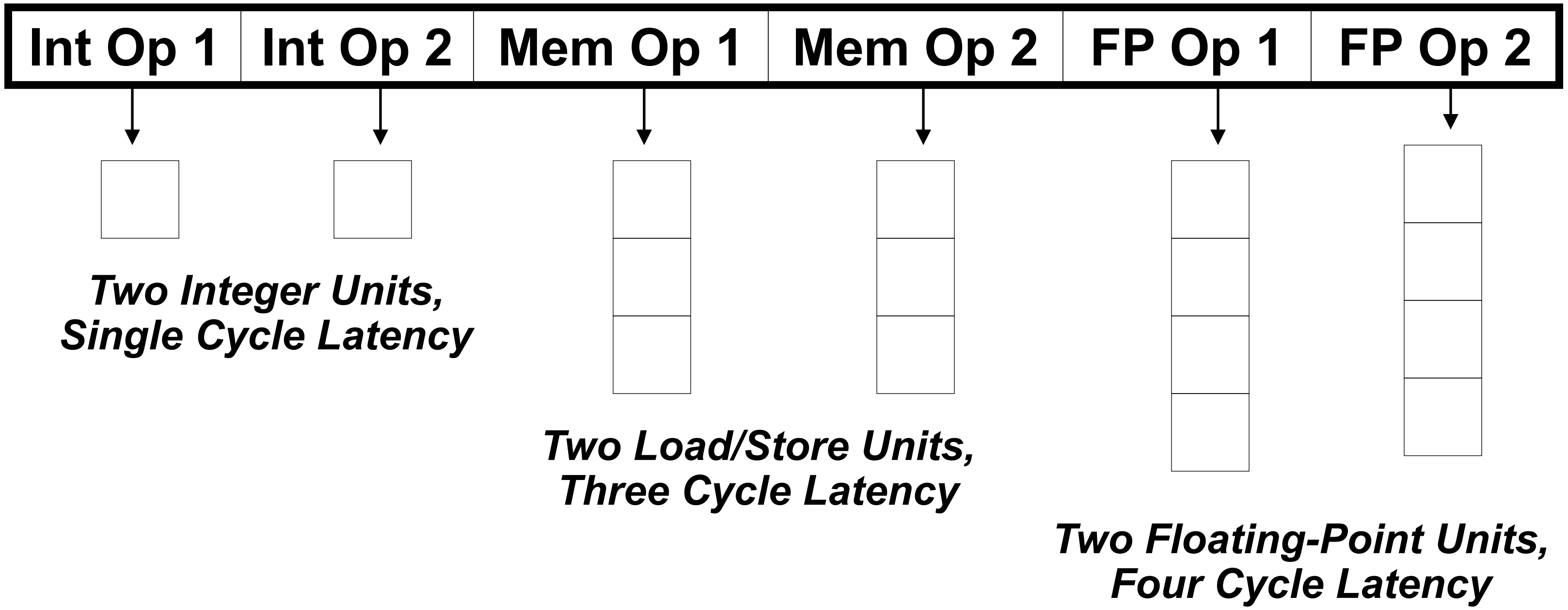
Exploiting SLP with SIMD Execution

- Benefit:
  - Multiple ALU ops $\rightarrow$ One SIMD op
  - Multiple ld/st ops $\rightarrow$ One wide mem op
- Cost:
  - Packing and unpacking
  - Reshuffling within a register

Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
  - Shared Instruction Processors
  - **Shared Sequencer Processors**
  - Shared Network Processors
  - Shared Memory Processors
  - Multicore Processors

Shared Sequencer
VLIW: Very Long Instruction Word



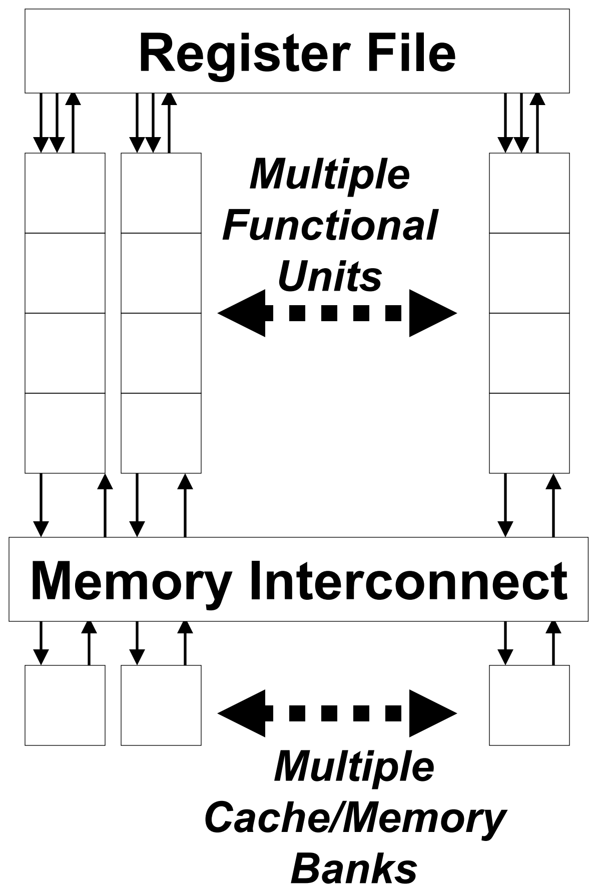
Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP Op 1
FP Op 2
Two Integer Units, Single Cycle Latency
Two Load/Store Units, Three Cycle Latency
Two Floating-Point Units, Four Cycle Latency

Compiler schedules parallel execution

Multiple parallel operations packed into one long instruction word

Compiler must avoid data hazards (no interlocks)

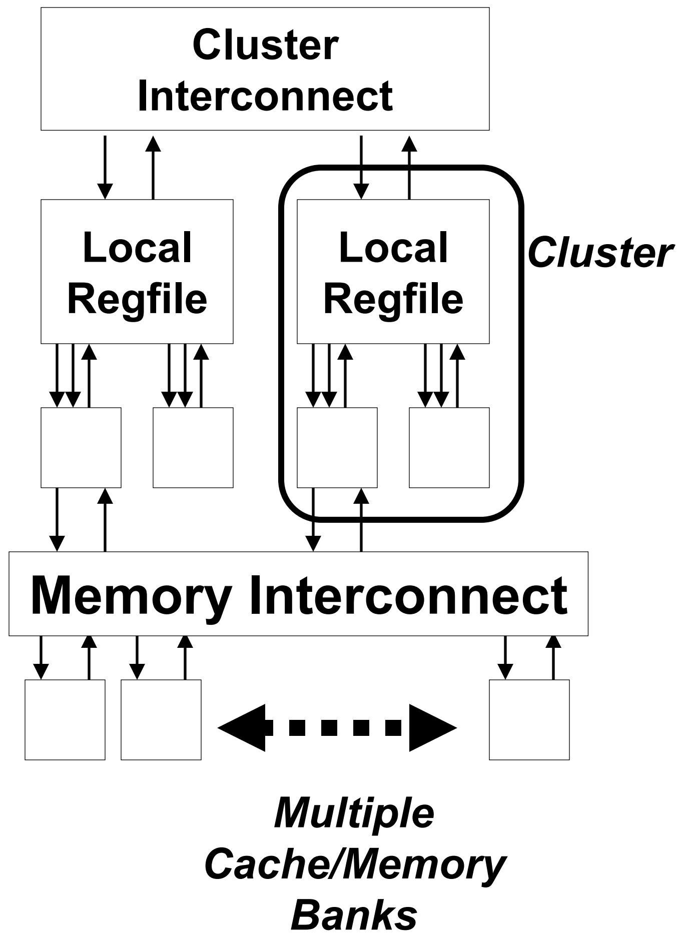
ILP Datapath Hardware Scaling



- Replicating functional units and cache/memory banks is straightforward and scales linearly
- Register file ports and bypass logic for N functional units scale quadratically (N*N)
- Memory interconnection among N functional units and memory banks also scales quadratically
- Technology scaling: *Wires are getting even slower relative to gate delays*
- Complex interconnect adds latency as well as area

=> *Need greater parallelism to hide latencies*

Clustered VLIW



- Divide machine into clusters of local register files and local functional units
- Lower bandwidth/higher latency interconnect between clusters
- Software responsible for mapping computations to minimize communication overhead

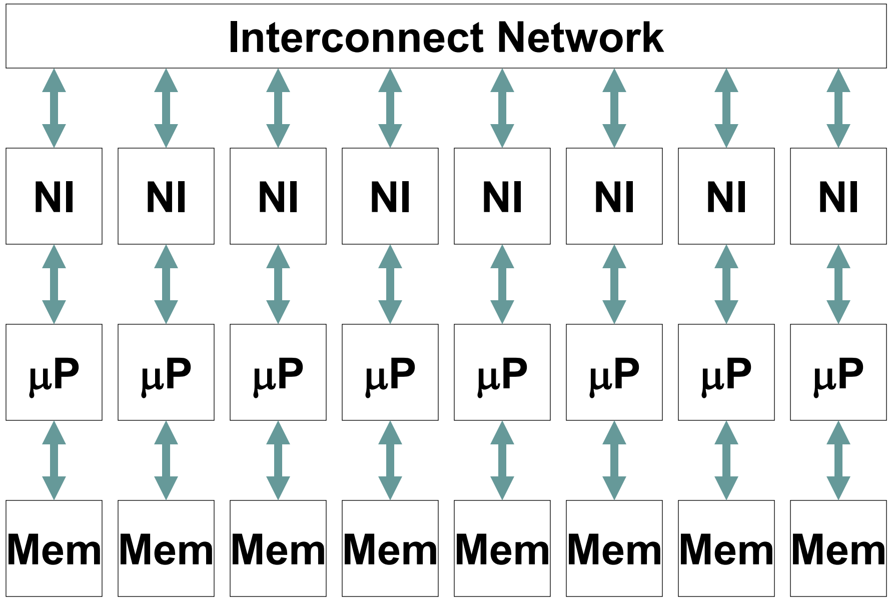
Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
  - Shared Instruction Processors
  - Shared Sequencer Processors
  - **Shared Network Processors**
  - Shared Memory Processors
  - Multicore Processors

Shared Network: Message Passing MPPs
(Massively Parallel Processors)

- Initial Research Projects
  - Caltech Cosmic Cube (early 1980s) using custom Mosaic processors
- Commercial Microprocessors including MPP Support
  - Transputer (1985)
  - nCube-1(1986) /nCube-2 (1990)
- Standard Microprocessors + Network Interfaces
  - Intel Paragon (i860)
  - TMC CM-5 (SPARC)
  - Meiko CS-2 (SPARC)
  - IBM SP-2 (RS/6000)
- MPP Vector Supers
  - Fujitsu VPP series

Designs can scale to 100s of 1000s of nodes…



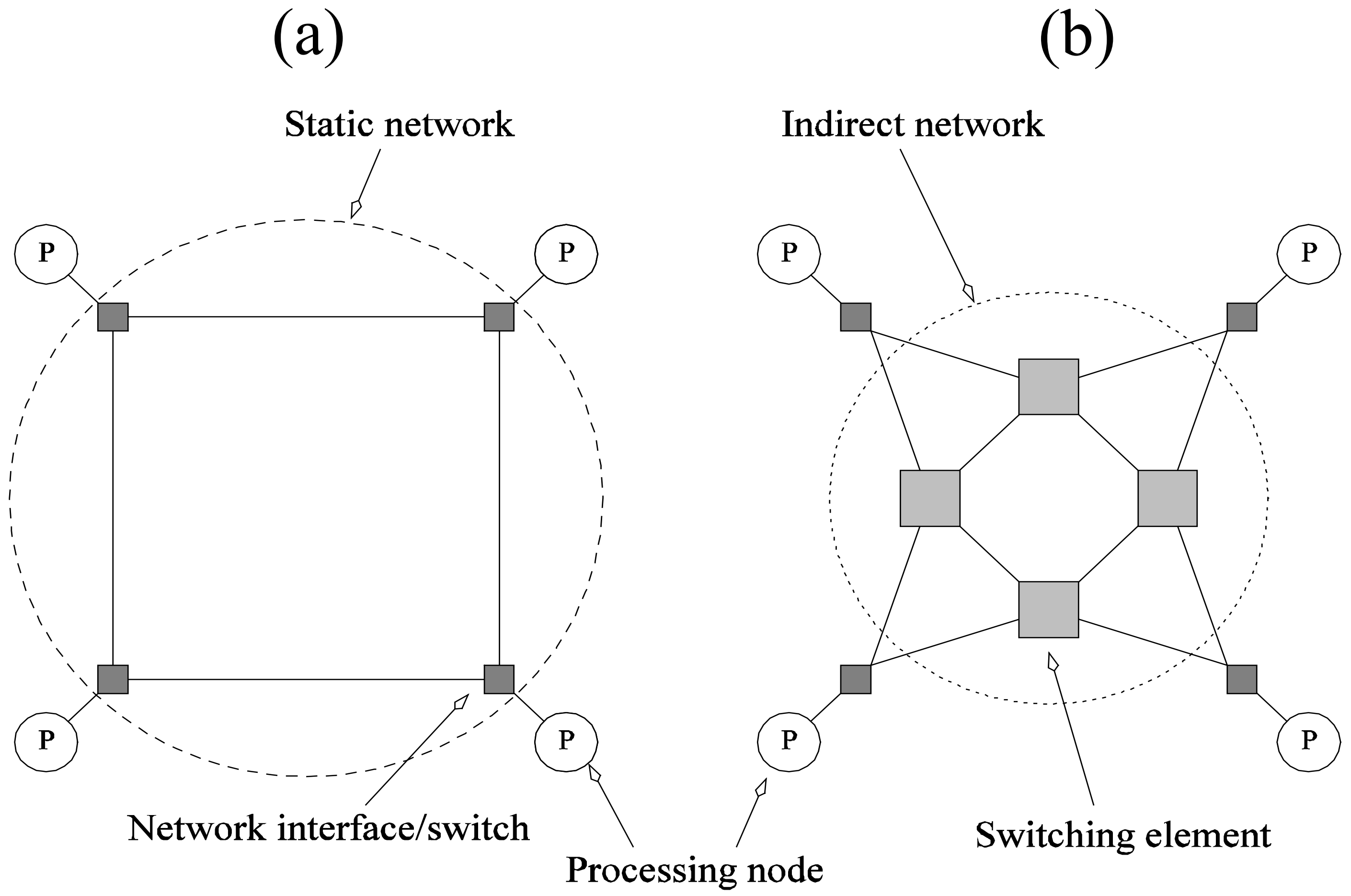
Message Passing MPP Problems

- All data layout must be handled by software
  - cannot retrieve remote data except with message request/reply
- Message passing has high software overhead
  - early machines had to invoke OS on each message
  - even user level access to network interface has dozens of cycles overhead (NI might be on I/O bus)
  - sending messages can be cheap (just like stores)
  - receiving messages is expensive, need to poll or interrupt

Interconnection Networks for Parallel Computers

- Interconnection networks carry data between processors and to memory.
- Interconnects are made of switches and links (wires, fiber).
- Interconnects are classified as static or dynamic.
- Static networks consist of point-to-point communication links among processing nodes and are also referred to as direct networks.
- Dynamic networks are built using switches and communication links. Dynamic networks are also referred to as indirect networks.

Static and Dynamic Interconnection Networks



Classification of interconnection networks: (a) a static network; and (b) a dynamic network.

Interconnection Networks

- Switches map a fixed number of inputs to outputs.
- The total number of ports on a switch is the degree of the switch.
- The cost of a switch grows as the *square* of the degree of the switch, the peripheral hardware linearly as the degree, and the packaging costs linearly as the number of pins.

Interconnection Networks: Network Interfaces

- Processors talk to the network via a network interface.
- The network interface may hang off the I/O bus or the memory bus.
- In a physical sense, this distinguishes a cluster from a tightly coupled multicomputer.
- The relative speeds of the I/O and memory buses impact the performance of the network.

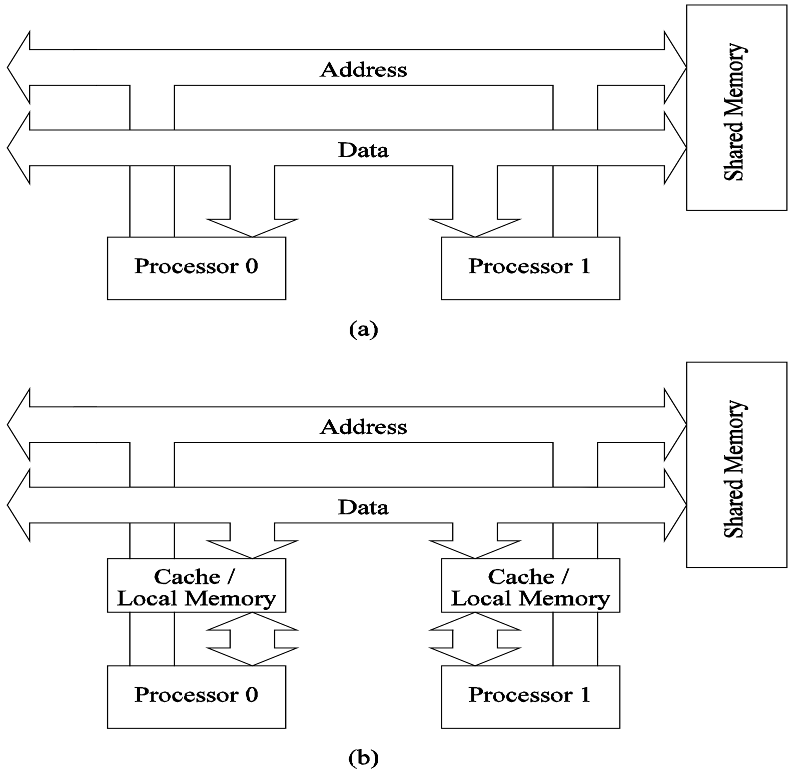
Network Topologies

- A variety of network topologies have been proposed and implemented.
- These topologies *trade off* performance for cost.
- Commercial machines often implement *hybrids* of multiple topologies for reasons of packaging, cost, and available components.

Network Topologies: Buses

- Some of the simplest and earliest parallel machines used buses.
- All processors access a common bus for exchanging data.
- The distance between any two nodes is $O(1)$ in a bus. The bus also provides a convenient broadcast media.
- However, the bandwidth of the shared bus is a major bottleneck.
- Typical bus-based machines are limited to dozens of nodes.

Network Topologies: Buses

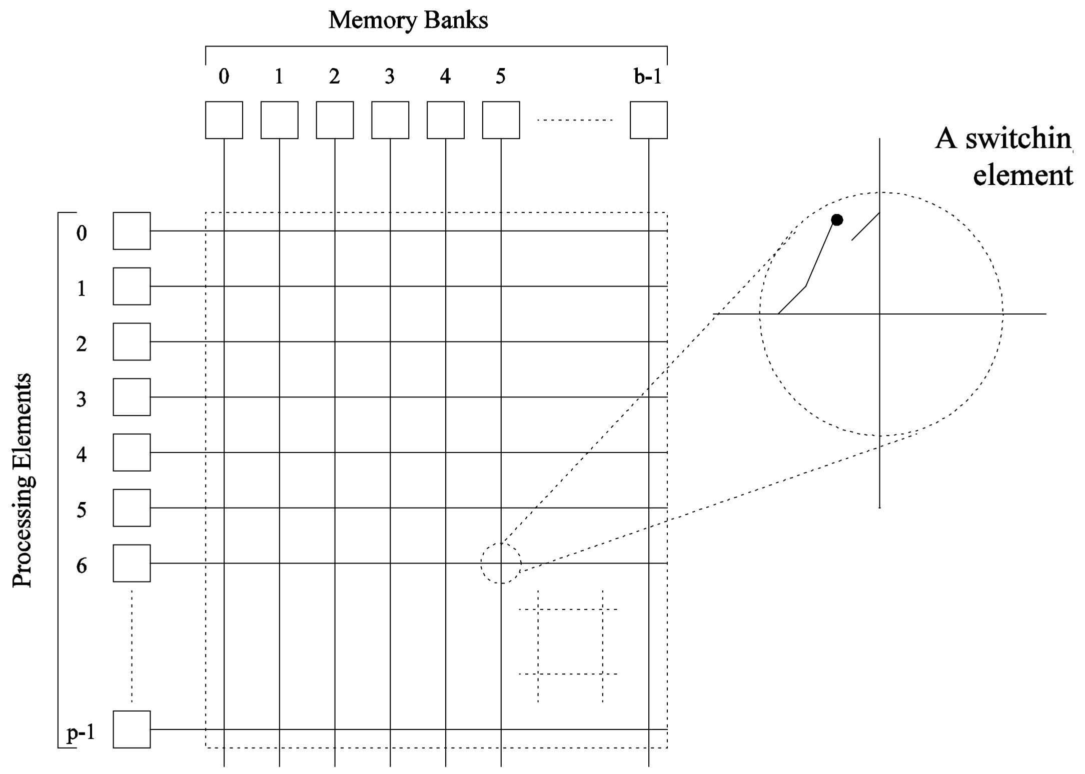


Bus-based interconnects (a) with no local caches; (b) with local memory/caches.

Since much of the data accessed by processors is local to the processor, a local memory can, at least in theory, improve the performance of bus-based machines.

Network Topologies: Crossbars

A crossbar network uses an $p \times m$ grid of switches to connect p inputs to m outputs in a non-blocking manner.



A completely non-blocking crossbar network connecting p processors to b memory banks.

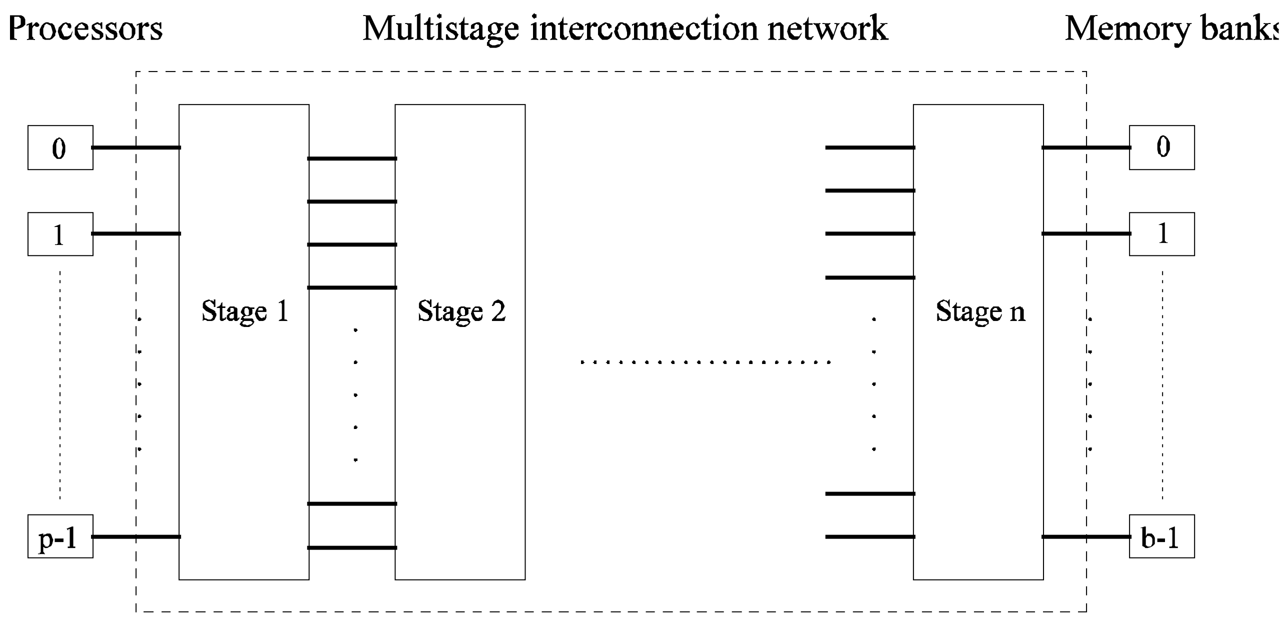
Network Topologies: Crossbars

- The cost of a crossbar of *p* processors grows as $O(p^2)$.
- This is generally difficult to scale for large values of *p*.

Network Topologies: Multistage Networks

- Crossbars have excellent performance scalability but poor cost scalability.
- Buses have excellent cost scalability, but poor performance scalability.
- Multistage interconnects strike a *compromise* between these extremes.

Network Topologies: Multistage Networks



The schematic of a typical multistage interconnection network.

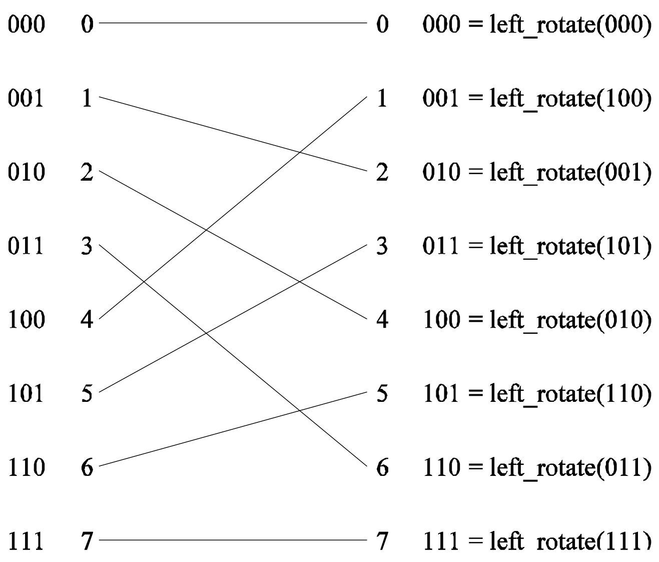
Network Topologies: Multistage Omega Network

- One of the most commonly used multistage interconnects is the Omega network.
- This network consists of *log p* stages, where *p* is the number of inputs/outputs.
- At each stage, input *i* is connected to output *j* if:

$$j = \begin{cases} 2i, & 0 \leq i \leq p/2 - 1 \\ 2i + 1 - p, & p/2 \leq i \leq p - 1 \end{cases}$$

Network Topologies: Multistage Omega Network

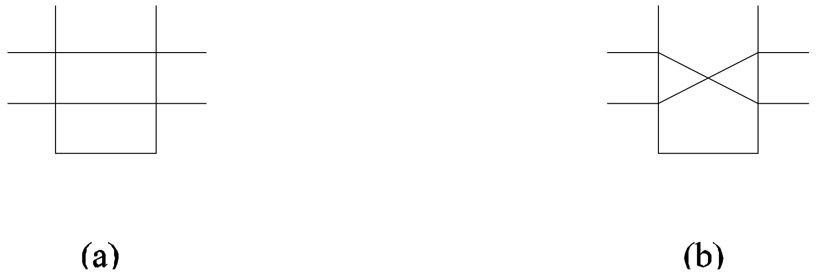
Each stage of the Omega network implements a perfect shuffle as follows:



A perfect shuffle interconnection for eight inputs and outputs.

Network Topologies: Multistage Omega Network

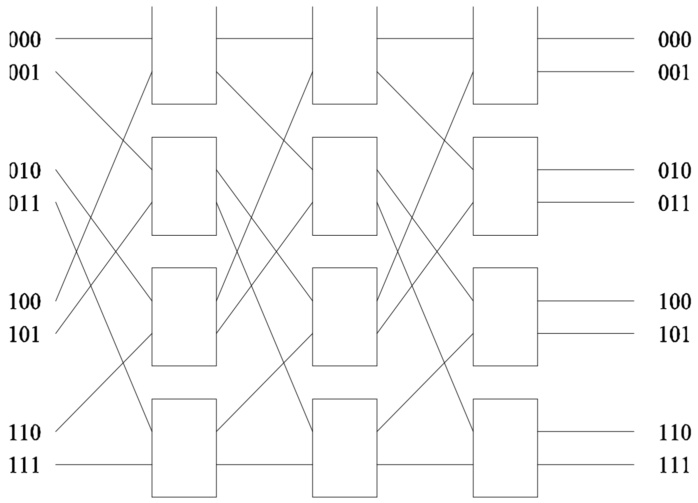
- The perfect shuffle patterns are connected using 2×2 switches.
- The switches operate in two modes – crossover or passthrough.



Two switching configurations of the 2 × 2 switch:
(a) Pass-through; (b) Cross-over.

Network Topologies: Multistage Omega Network

A complete Omega network with the perfect shuffle interconnects and switches can now be illustrated:



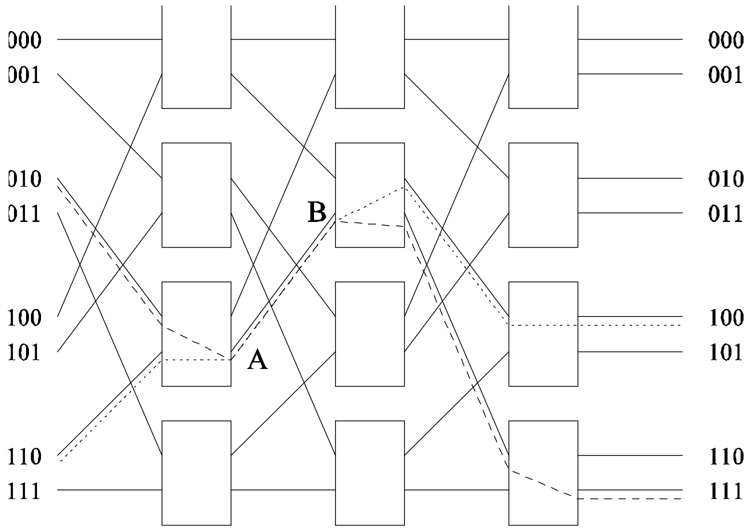
A complete omega network connecting eight inputs and eight outputs.

An omega network has *p/2* × *log p* switching nodes, and the cost of such a network grows as *(p log p)*.

Network Topologies: Multistage Omega Network – Routing

- Let *s* be the binary representation of the source and *d* be that of the destination processor.
- The data traverses the link to the first switching node. If the most significant bits of *s* and *d* are the same, then the data is routed in pass-through mode by the switch else, it switches to crossover.
- This process is repeated for each of the *log p* switching stages.
- Note that this is not a non-blocking switch.

Network Topologies: Multistage Omega Network – Routing



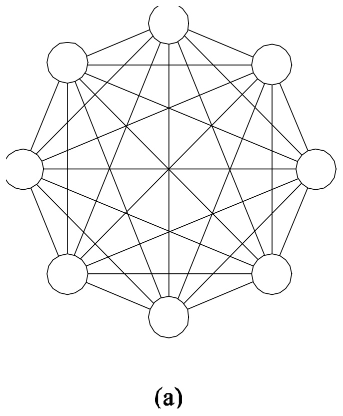
An example of blocking in omega network: one of the messages (010 to 111 or 110 to 100) is blocked at link AB.

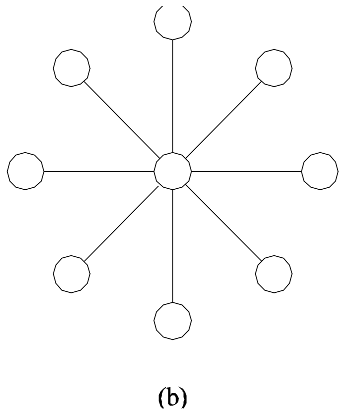
Network Topologies: Completely Connected Network

- Each processor is connected to every other processor.
- The number of links in the network scales as $O(p^2)$.
- While the performance scales very well, the hardware complexity is not realizable for large values of p.
- In this sense, these networks are static counterparts of crossbars.

Network Topologies: Completely Connected and Star Connected Networks

Example of an 8-node completely connected network.





(a) A completely-connected network of eight nodes;
(b) a star connected network of nine nodes.

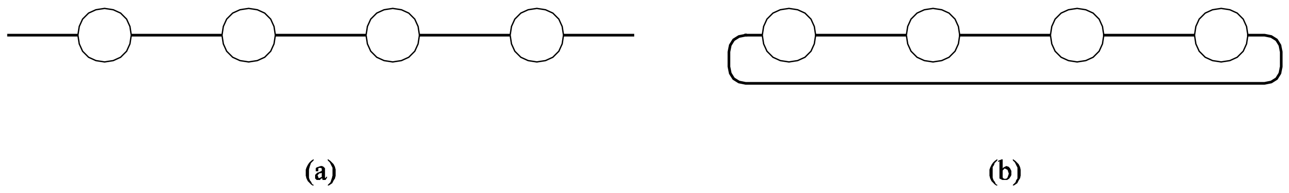
Network Topologies:
Star Connected Network

- Every node is connected only to a common node at the center.
- Distance between any pair of nodes is *O(1).* However, the central node becomes a bottleneck.
- In this sense, star connected networks are static counterparts of buses.

Network Topologies: Linear Arrays, Meshes, and *k-d* Meshes

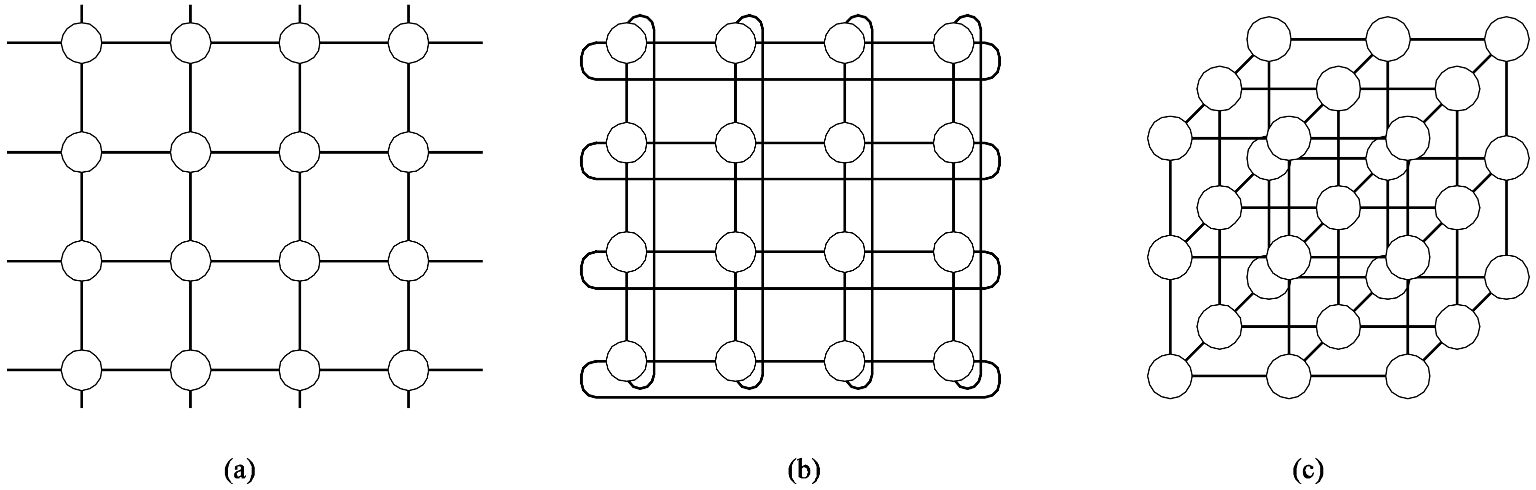
- In a linear array, each node has two neighbors, one to its left and one to its right. If the nodes at either end are connected, we refer to it as a 1-D torus or a ring.
- A generalization to 2 dimensions has nodes with 4 neighbors, to the north, south, east, and west.
- A further generalization to d dimensions has nodes with $2d$ neighbors.
- A special case of a d-dimensional mesh is a hypercube. Here, $d = \log p$, where p is the total number of nodes.

Network Topologies: Linear Arrays



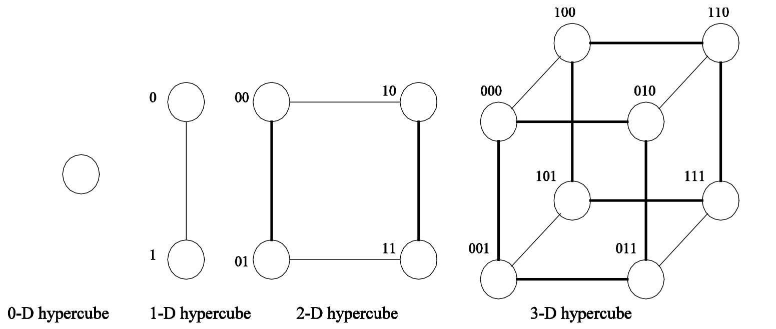
Linear arrays: (a) with no wraparound links; (b) with wraparound link.

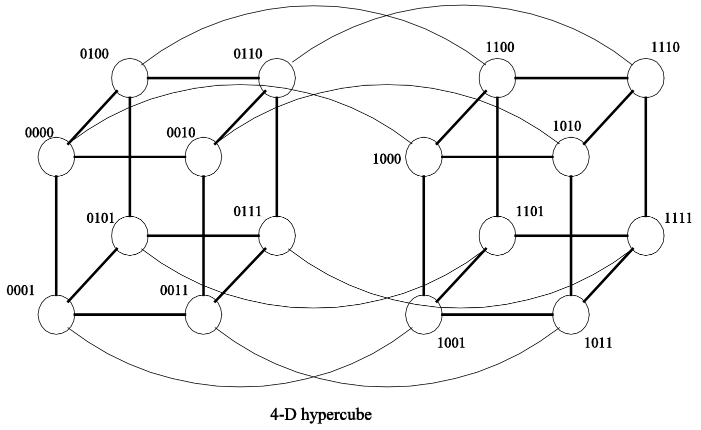
Network Topologies: Two- and Three Dimensional Meshes



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.

Network Topologies: Hypercubes and their Construction



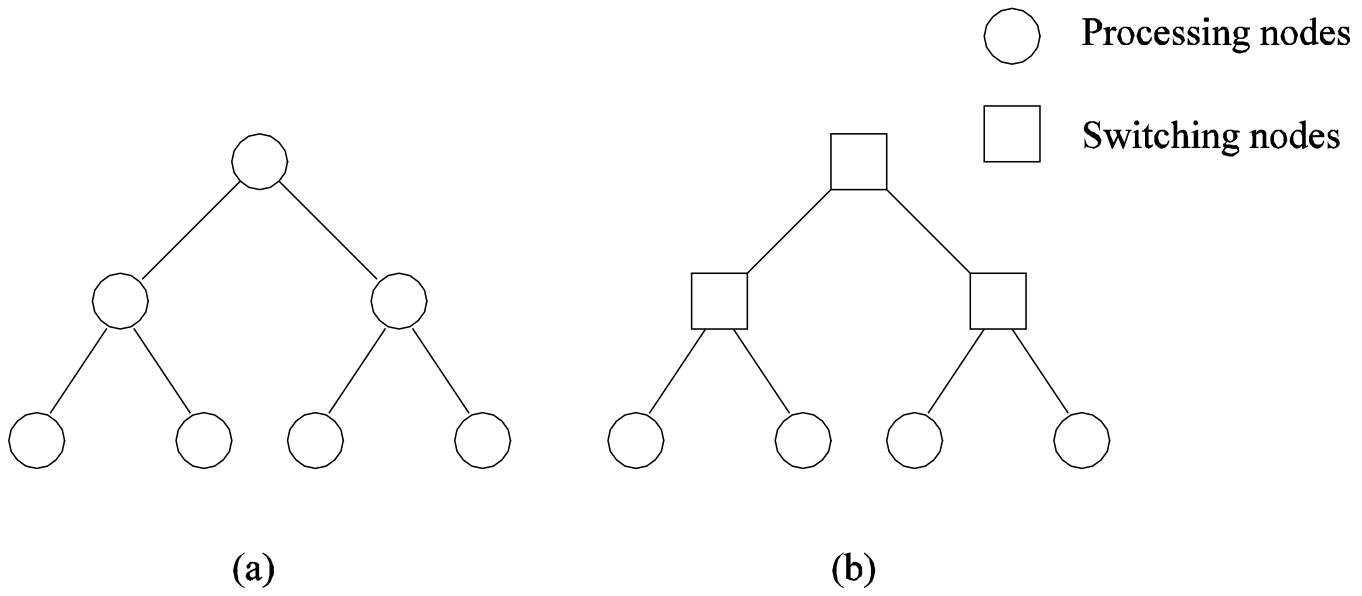


Construction of hypercubes from hypercubes of lower dimension.

Network Topologies: Properties of Hypercubes

- The distance between any two nodes is at most *log p*.
- Each node has *log p* neighbors.
- The distance between two nodes is given by the number of bit positions at which the two nodes differ.

Network Topologies: Tree-Based Networks

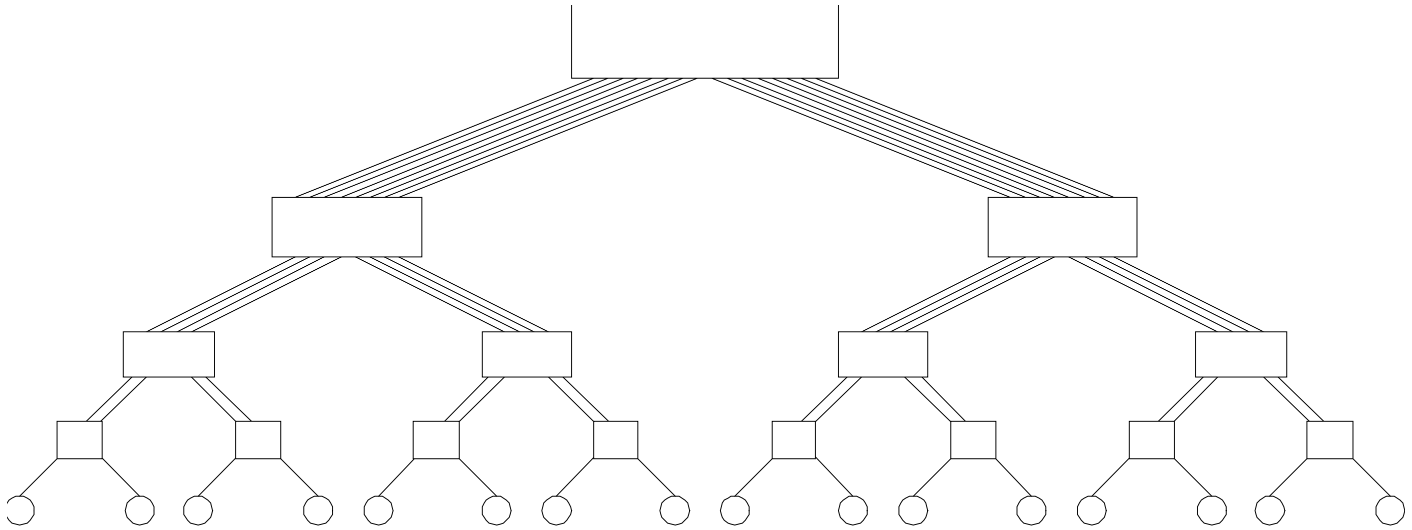


Complete binary tree networks: (a) a static tree network; and (b) a dynamic tree network.

Network Topologies: Tree Properties

- The distance between any two nodes is no more than *2logp*.
- Links higher up the tree potentially carry more traffic than those at the lower levels.
- For this reason, a variant called fat-tree, *fattens* the links as we go up the tree.
- Trees can be laid out in 2D with *no* wire crossings. This is an attractive property of trees.

Network Topologies: Fat Trees



A fat tree network of 16 processing nodes.

Evaluating Static Interconnection Networks

- Diameter: The distance between the farthest two nodes in the network. The diameter of a linear array is $p - 1$, that of a mesh is $2(\sqrt{p} - 1)$, that of a tree and hypercube is $\log p$, and that of a completely connected network is $O(1)$.
- Bisection Width: The minimum number of wires you must cut to divide the network into two equal parts. The bisection width of a linear array and tree is *1*, that of a mesh is $\sqrt{p}$, that of a hypercube is $p/2$ and that of a completely connected network is $p^2/4$.
- Cost: The number of links or switches (whichever is asymptotically higher) is a meaningful measure of the cost. However, a number of other factors, such as the ability to layout the network, the length of wires, etc., also factor into the cost.

Evaluating Static Interconnection Networks

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

Evaluating Dynamic Interconnection Networks

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Crossbar	1	p	1	p^2
Omega Network	$\log p$	$p/2$	2	$p/2$
Dynamic Tree	$2 \log p$	1	2	$p - 1$

Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
  - Shared Instruction Processors
  - Shared Sequencer Processors
  - Shared Network Processors
  - **Shared Memory Processors**
  - Multicore Processors

Shared Memory: Shared Memory Multiprocessors

- Will work with any data placement (but might be slow)
  - can choose to optimize only critical portions of code
- Load and store instructions used to communicate data between processes
  - no OS involvement
  - low software overhead
- Usually some special synchronization primitives
  - fetch&op
  - load linked/store conditional (LL/SC)
- **In large scale systems, the logically shared memory is implemented as physically distributed memory modules**
- Two main categories
  - non cache coherent
  - hardware cache coherent

Shared Memory: Shared Memory Multiprocessors

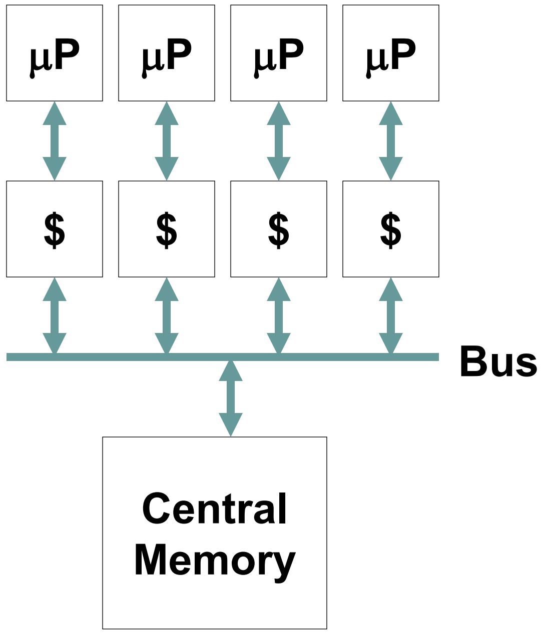
- No hardware cache coherence
  - IBM RP3
  - BBN Butterfly
  - Cray T3D/T3E
  - Parallel vector supercomputers (Cray T90, NEC SX-5)
- Hardware cache coherence
  - many small-scale SMPs (e.g. Quad Pentium Xeon systems and most Intel/AMD machines today)
  - large scale bus/crossbar-based SMPs (Sun Starfire)
  - large scale directory-based SMPs (SGI Origin)

HW Cache Coherency

- Bus-based Snooping Solution
  - Send all requests for data to all processors
  - Processors snoop to see if they have a copy and respond accordingly
  - Requires broadcast, since caching information is at processors
  - Works well with bus (natural broadcast medium)
  - Dominates for small scale machines (most of the market)

- Directory-Based Schemes
  - Keep track of what is being shared in 1 centralized place (logically)
  - Distributed memory => distributed directory for scalability (avoids bottlenecks)
  - Send point-to-point requests to processors via network
  - Scales better than Snooping
  - Actually existed BEFORE Snooping-based schemes

Bus-Based Cache-Coherent SMPs



- Small scale (<= 8 processors) bus-based SMPs by far the most common parallel processing platform today
- Bus provides broadcast and serialization point for simple snooping cache coherence protocol
- Many modern microprocessors integrate support for this protocol

Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
    - Shared Instruction Processors
    - Shared Sequencer Processors
    - Shared Network Processors
    - Shared Memory Processors
    - **Multicore Processors**

Multicore Processors

- Multiple cores on the same die
- Extensive on-chip resource sharing
  - Caches, NoC, memory controllers
- Increasing diversity in topology
  - Write-once-run-everywhere will not work!

- From power wall and complexity wall to programmability wall