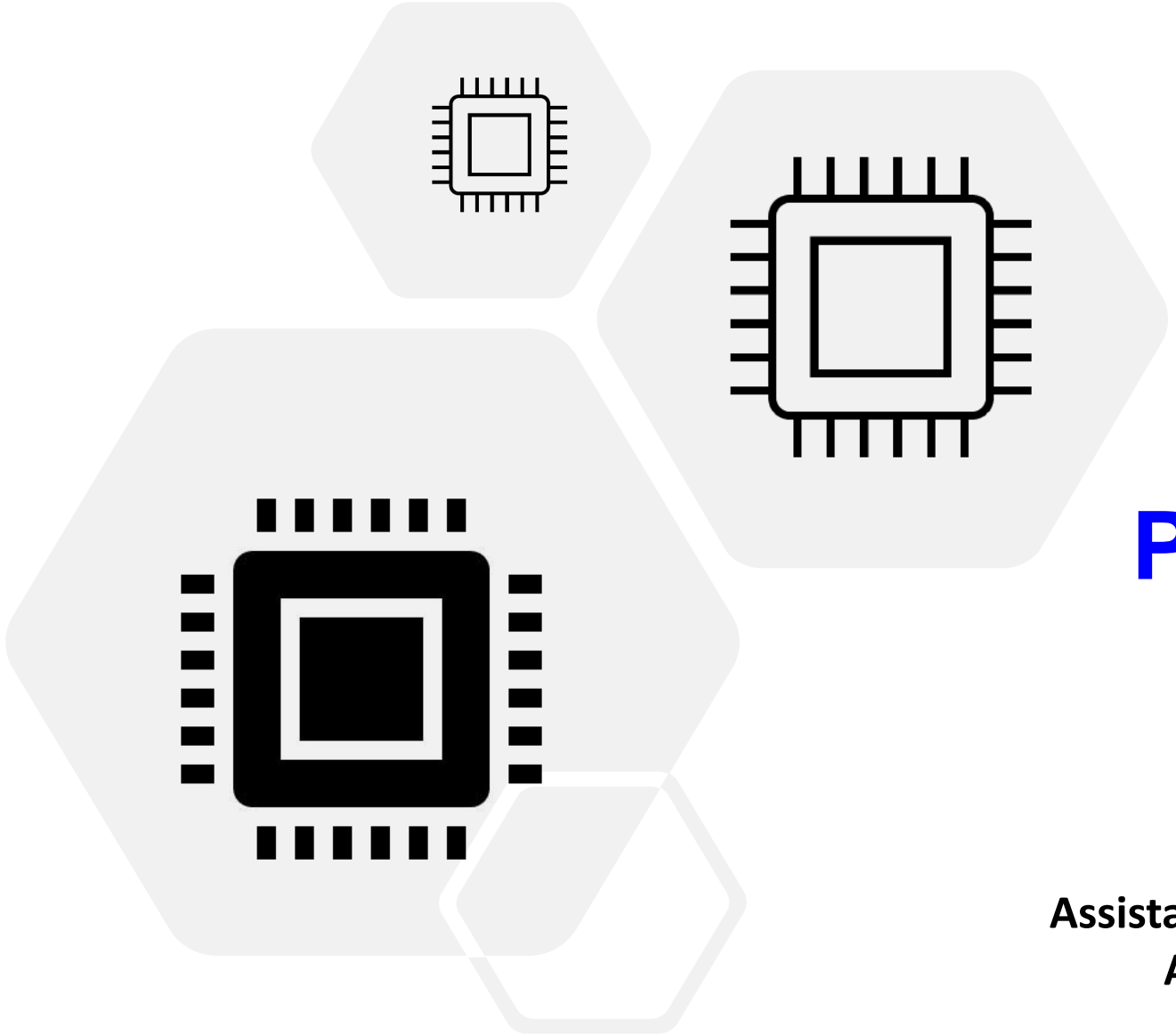
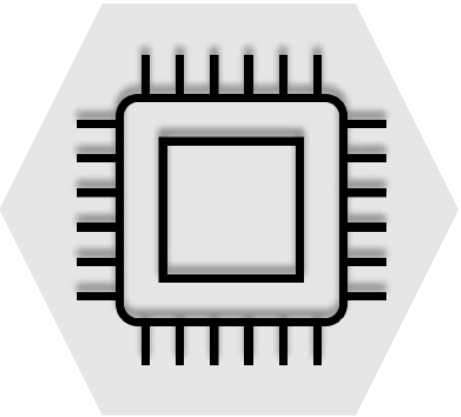
High Performance Computing



Module 2 - Parallel Programming Platforms

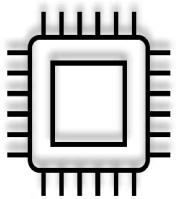
Shafaque Fatma Syed
Assistant Professor - Dept. of Information Technology
A P Shah Institute of Technology, Mumbai

Topics to be discussed



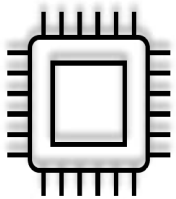
- Implicit Parallelism: Trends in Microprocessors and Architectures
- Dichotomy of Parallel Computing Platforms
- Physical Organization of Parallel Platforms
- Communication Costs in Parallel Machines

Scope of Parallelism



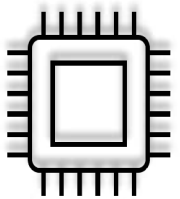
- Conventional architectures coarsely comprise of a processor, memory system, and the datapath.
- Each of these components present significant performance bottlenecks.
- Parallelism addresses each of these components in significant ways.
- One of the most important innovations is multiplicity – in processing units, datapaths, and memory units.
- Different applications utilize different aspects of parallelism - e.g., data intensive applications utilize high aggregate throughput, server applications utilize high aggregate network bandwidth, and scientific applications typically utilize high processing and memory system performance.
- It is important to understand each of these performance bottlenecks.

Implicit Parallelism: Trends in Microprocessor Architectures



- Microprocessor clock speeds have posted impressive gains over the past two decades (two to three orders of magnitude).
- Higher levels of device integration have made available a large number of transistors.
- The question of how best to utilize these resources is an important one.
- Current processors use these resources in multiple functional units and execute multiple instructions in the same cycle (such as the Itanium, Sparc Ultra, MIPS, and Power4).
- The precise manner in which these instructions are selected and executed (in order or out of order)provides impressive diversity in architectures.

Pipelining and Superscalar Execution



- Pipelining overlaps various stages of instruction execution to achieve performance.
- At a high level of abstraction, an instruction can be executed while the next one is being decoded and the next one is being fetched.
- Pipelining, however, has several limitations.
- The speed of a pipeline is eventually limited by the slowest stage.
- In typical program traces, every 5-6th instruction is a conditional jump! This requires very accurate branch prediction.
- The penalty of a misprediction grows with the depth of the pipeline, since a larger number of instructions will have to be flushed. (Imagine 20 stage pipelines in Pentium 4 processors!!).
- One simple way of alleviating these bottlenecks is to use multiple pipelines.
- The question then becomes one of selecting these instructions.

NOTE: Simple instructions means operation is completed in one clock cycle. -->

Instruction Issue Algorithm

- Decode the two consecutive instructions I1 and I2
- If the following are all true
  - I1 and I2 are simple instructions
  - I1 is not a jump instruction
  - Destination of I1 is not a source of I2
  - Destination of I1 is not a destination of I2
- Then issue I1 to u pipeline and I2 to v pipeline
- Else issue I1 to u pipeline

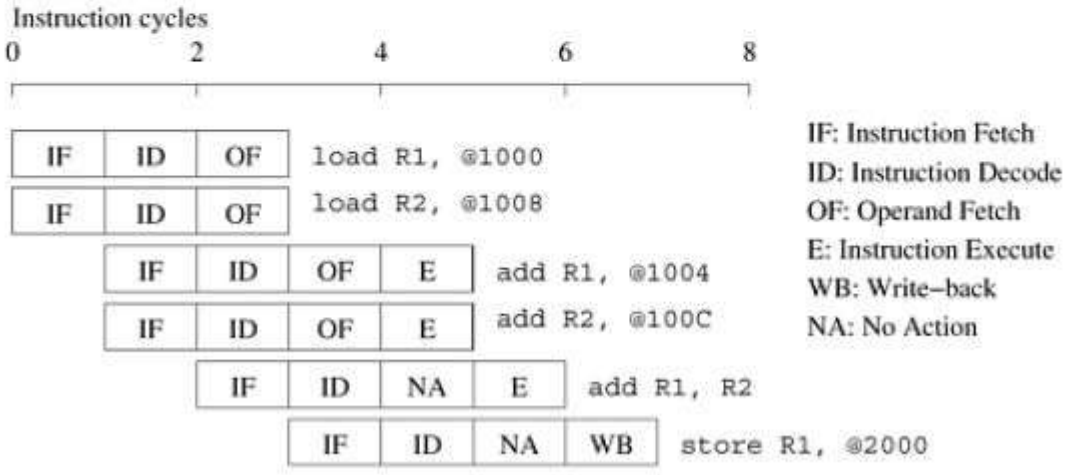
Superscalar Execution: An Example

Example of a two-way superscalar execution of instructions.

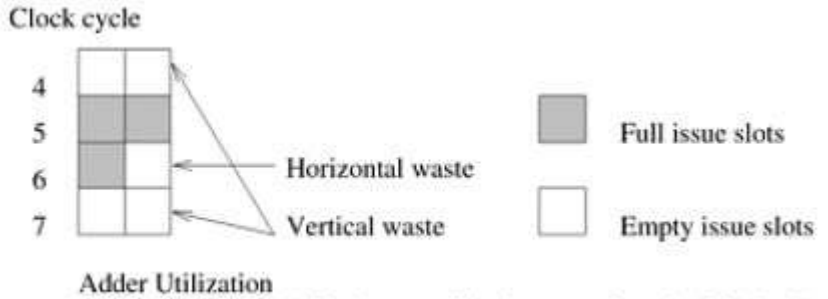
```
1. load R1, @1000      1. load R1, @1000      1. load R1, @1000
2. load R2, @1008      2. add R1, @1004       2. add R1, @1004
3. add R1, @1004       3. add R1, @1008       3. load R2, @1008
4. add R2, @100C       4. add R1, @100C       4. add R2, @100C
5. add R1, R2          5. store R1, @2000     5. add R1, R2
6. store R1, @2000                            6. store R1, @2000
```

(i) (ii) (iii)

(a) Three different code fragments for adding a list of four numbers.



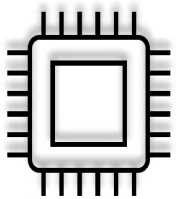
(b) Execution schedule for code fragment (i) above.



(c) Hardware utilization trace for schedule in (b).

- In the above example, there is some wastage of resources(execution unit) due to data dependencies.
- If, during a particular cycle, no instructions are issued on the execution units, it is referred to as vertical waste; if only part of the execution units are used during a cycle, it is termed horizontal waste.

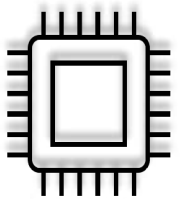
Superscalar Execution



Scheduling of instructions is determined by a number of factors:

- **True Data Dependency:** The result of one operation is an input to the next.
- **Resource Dependency:** Two operations require the same resource.
- **Branch Dependency:** Scheduling instructions across conditional branch statements cannot be done deterministically a-priori.
- The scheduler, a piece of hardware looks at a large number of instructions in an instruction queue and selects appropriate number of instructions to execute concurrently based on factors.
- The complexity of this hardware is an important constraint on superscalar processors.

Superscalar Execution



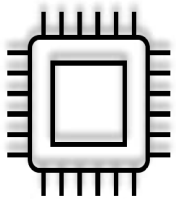
Issue Mechanism:

- In the simpler model, instructions can be issued only in the order in which they are encountered. That is, if the second instruction cannot be issued because it has a data dependency with the first, only one instruction is issued in the cycle. This is called *in-order* issue.
- In a more aggressive model, instructions can be issued out of order. In this case, if the second instruction has data dependencies with the first, but the third instruction does not, the first and third instructions can be co-scheduled. This is also called dynamic issue.
- Performance of in-order issue is generally limited.

Efficiency Considerations :

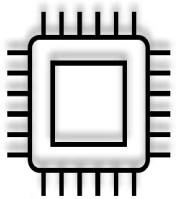
- Not all functional units can be kept busy at all times.
- If during a cycle, no functional units are utilized, this is referred to as vertical waste.
- If during a cycle, only some of the functional units are utilized, this is referred to as horizontal waste.
- Due to limited parallelism in typical instruction traces, dependencies, or the inability of the scheduler to extract parallelism, the performance of superscalar processors is eventually limited.
- Conventional microprocessors typically support four-way superscalar execution.

Very Long Instruction Word (VLIW) Processors

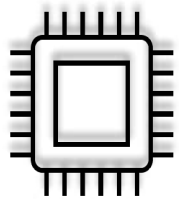


- The hardware cost and complexity of the superscalar scheduler is a major consideration in processor design.
- To address this issues, VLIW processors rely on compile time analysis to identify and bundle together instructions that can be executed concurrently.
- These instructions are packed and dispatched together, and thus the name very long instruction word.
- This concept was used with some commercial success in the Multiflow Trace machine (1984).
- Variants of this concept are employed in the Intel IA64 processors.

Very Long Instruction Word (VLIW) Processors: Considerations



- Issue hardware is simpler.
- Compiler has a bigger context from which to select co-scheduled instructions.
- Compilers, however, do not have runtime information such as cache misses. Scheduling is, therefore, inherently conservative.
- Branch and memory prediction is more difficult.
- VLIW performance is highly dependent on the compiler. A number of techniques such as loop unrolling, speculative execution, branch prediction are critical.
- Typical VLIW processors are limited to 4-way to 8-way parallelism.



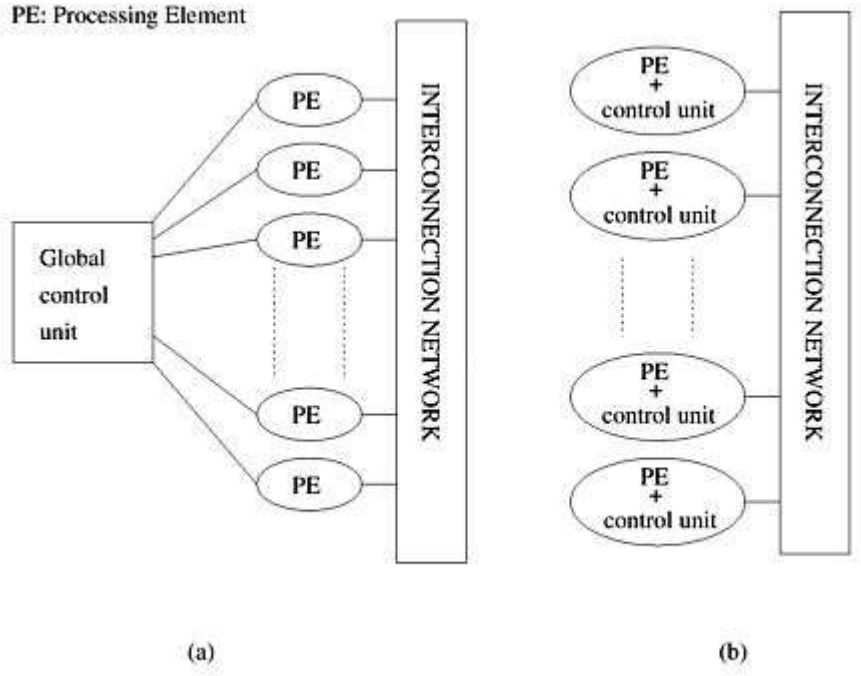
Dichotomy of Parallel Computing Platforms

Introduction to dichotomy of parallel computing platforms

- Parallel Computing platforms can be explored on the basis of logical and physical organization.
- The logical organization refers to a programmer's view of the platform(control structure)
- The physical organization refers to the actual hardware organization of the platform(communication model)
- The two critical components of parallel computing from a programmer's perspective are ways of expressing parallel tasks and mechanisms for specifying interaction between these tasks.

Control Structure of Parallel Programs

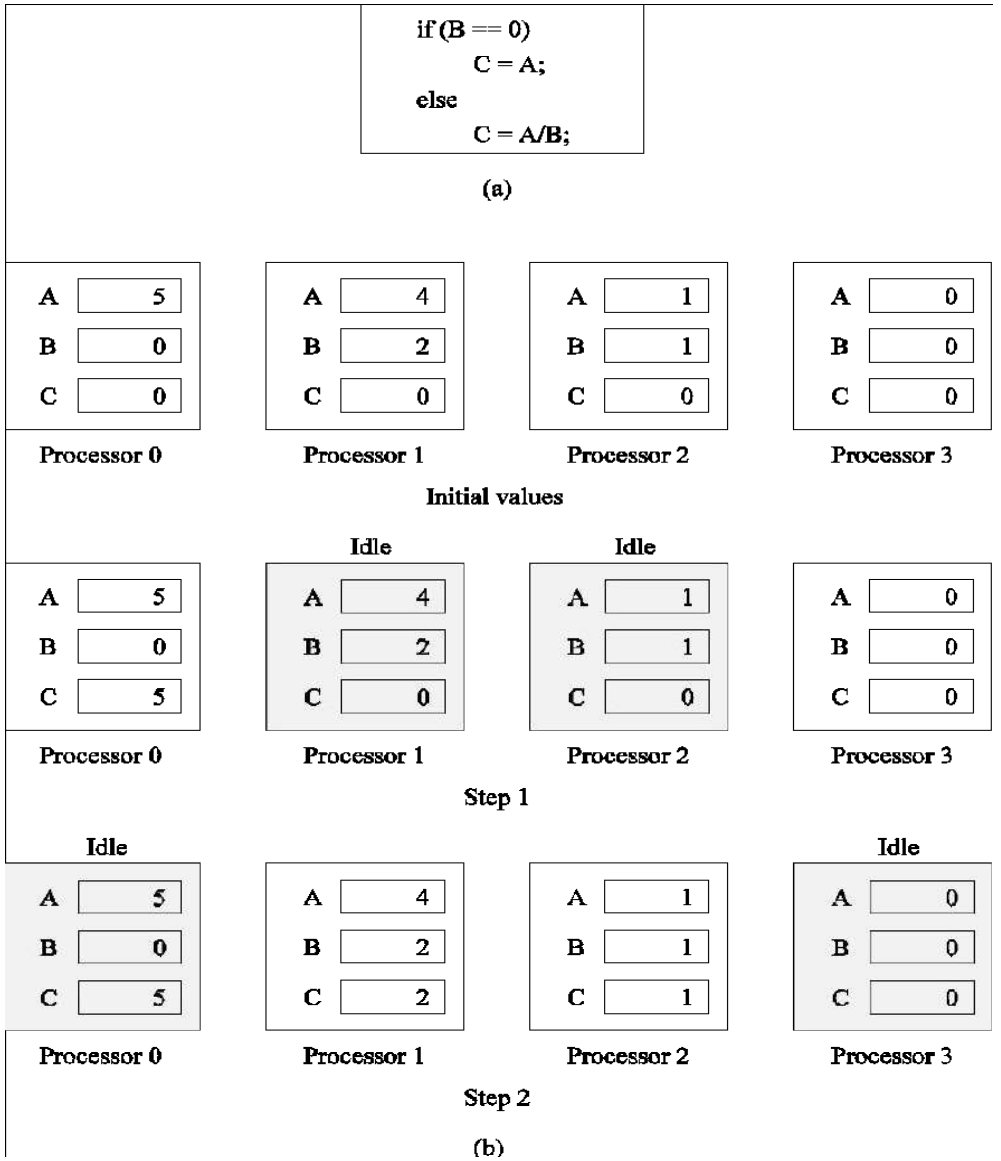
- Parallelism can be expressed at various levels of granularity - from instruction level to processes.
- Between these extremes exist a range of models, along with corresponding architectural support.
- Processing units in parallel computers either operate under the centralized control of a single control unit or work independently.
- If there is a single control unit that dispatches the same instruction to various processors (that work on different data), the model is referred to as single instruction stream, multiple data stream (SIMD).
- If each processor has its own control unit, each processor can execute different instructions on different data items. This model is called multiple instruction stream, multiple data stream (MIMD).



Disadvantages of SIMD

- Even if SIMD computers require less hardware (only one global control unit) and less memory(only one copy of the program needs to be stored),still SIMD is not popularly used.
- Since SIMD processors ae specially designed, they tend to be expensive and have long design cycles.
- Not all applications are naturally suited to SIMD processors.
- SIMD architectures yield poor resource utilization in the case of conditional execution.

Note:
Examples:Illiac IV, MPP, DAP, CM-2, and MasPar MP-1
It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an ``activity mask'', which determines if a processor should participate in a computation or not.



MIMD Processors

- In contrast to SIMD processors, MIMD processors can execute different programs on different processors.
- A variant of this, called single program multiple data streams (SPMD) executes the same program on different processors.
- It is easy to see that SPMD and MIMD are closely related in terms of programming flexibility and underlying architectural support.
- Examples of SPMD platforms include current generation Sun Ultra Servers, SGI Origin Servers, multiprocessor PCs, workstation clusters, and the IBM SP.

Communication Model of Parallel Platforms

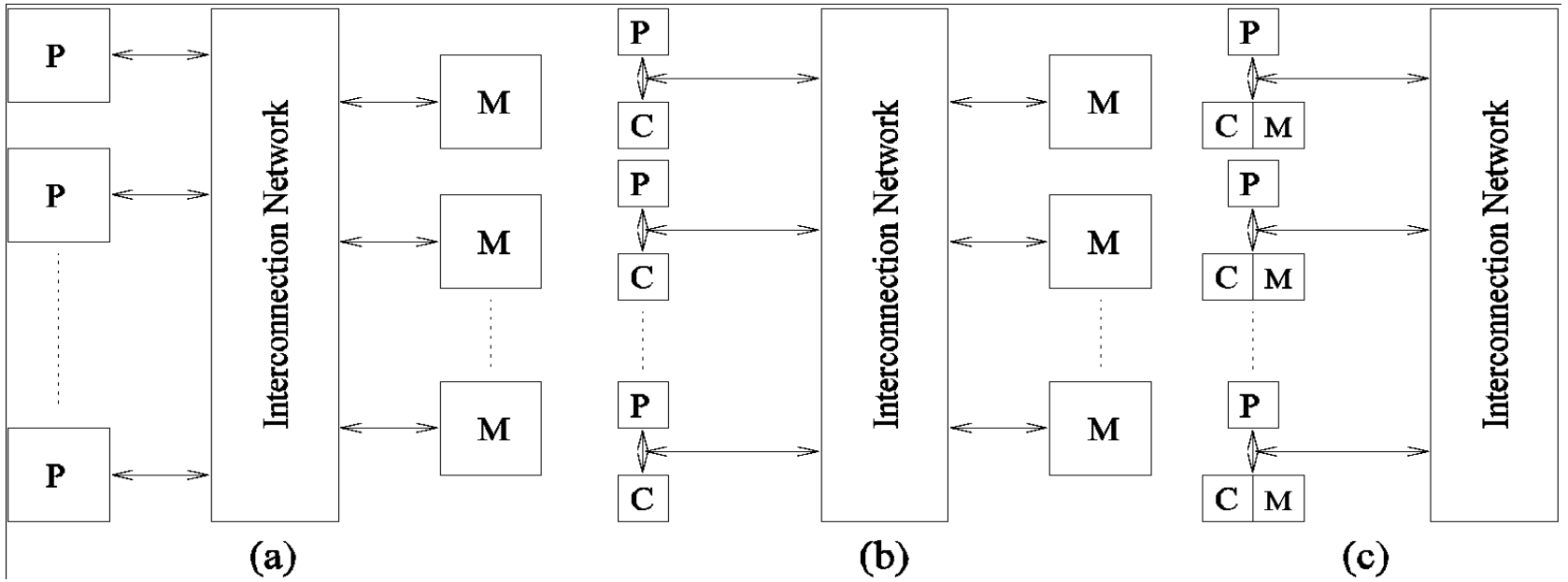
- There are two primary forms of data exchange between parallel tasks - accessing a shared data space and exchanging messages.
- Platforms that provide a shared data space are called shared-address-space machines or multiprocessors.
- Platforms that support messaging are also called message passing platforms or multicomputers.

Shared-address-space Platforms:

- Part (or all) of the memory is accessible to all processors. Memory in shared-address-space platforms can be local (exclusive to a processor) or global (common to all processors).
- Processors interact by modifying data objects stored in this shared-address-space.
- If the time taken by a processor to access any memory word in the system(global or local) is identical, the platform is classified as a uniform memory access (UMA), else, a non-uniform memory access (NUMA) machine.

NUMA and UMA Shared-Address-Space Platforms

- The distinction between NUMA and UMA platforms is important from the point of view of algorithm design. NUMA machines require locality from underlying algorithms for performance.
- Read-write data to shared data must be coordinated.
- Caches in such machines require coordinated access to multiple copies. This leads to the cache coherence problem.
- A weaker model of these machines provides an address map, but not coordinated access. These models are called non cache coherent shared address space machines.



Typical shared-address-space architectures: (a) Uniform-memory-access shared-address-space computer; (b) Uniform-memory-access shared-address-space computer with caches and memories; (c) Non-uniform-memory access shared-address-space computer with local memory only.

Important Viva Question

What is the difference between shared-address-space and shared-memory computers?

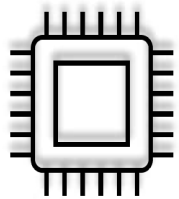
- The term shared-memory computer is historically used for architectures in which the memory is physically shared among various processors, i.e., each processor has equal access to any memory segment. This is identical to the UMA model.
- A shared-address-space computer is identical to a NUMA machine.

Message-Passing Platforms

- These platforms comprise of a set of processors and their own (exclusive) memory.
- Instances of such a view come naturally from clustered workstations and non-shared-address-space multicomputers.
- These platforms are programmed using (variants of) send and receive primitives.
- Libraries such as MPI and PVM provide such primitives.

Difference between message passing and shared address platforms

- Message passing requires little hardware support, other than a network.
- Shared address space platforms can easily emulate message passing. The reverse is more difficult to do (in an efficient manner).



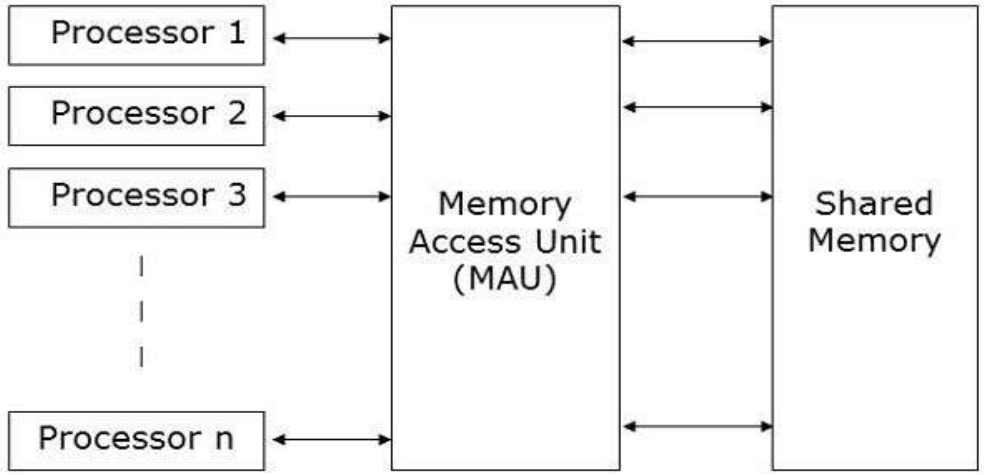
Physical Organization of Parallel Platforms

Architecture of an Ideal Parallel Computer

- A natural extension of the Random Access Machine (RAM) serial architecture is the Parallel Random Access Machine, or PRAM.
- PRAMs consist of *p* processors and a global memory of unbounded size that is uniformly accessible to all processors.
- Processors share a common clock but may execute different instructions in each cycle.

A PRAM model contains –

- A set of similar type of processors.
- All the processors share a common memory unit. Processors can communicate among themselves through the shared memory only.
- A memory access unit (MAU) connects the processors with the single shared memory.



Architecture of an Ideal Parallel Computer

Here, n number of processors can perform independent operations on n number of data in a particular unit of time. This may result in simultaneous access of same memory location by different processors.

To solve this problem, the following constraints have been enforced on PRAM model –

- Exclusive Read Exclusive Write (EREW) – Here no two processors are allowed to read from or write to the same memory location at the same time.
- Exclusive Read Concurrent Write (ERCW) – Here no two processors are allowed to read from the same memory location at the same time, but are allowed to write to the same memory location at the same time.
- Concurrent Read Exclusive Write (CREW) – Here all the processors are allowed to read from the same memory location at the same time, but are not allowed to write to the same memory location at the same time.
- Concurrent Read Concurrent Write (CRCW) – All the processors are allowed to read from or write to the same memory location at the same time.

Types of Concurrent writes

- Allowing concurrent read access does not create any semantic discrepancies in the program.
- However, concurrent write access to a memory location requires arbitration. Several protocols are used to resolve concurrent writes. The most frequently used protocols are as follows:

- Common, in which the concurrent write is allowed if all the values that the processors are attempting to write are identical. (p1 x=1, p2 x=1, p3 x=1, etc)
- Arbitrary, in which an arbitrary processor is allowed to proceed with the write operation and the rest fail.
- Priority, in which all processors are organized into a predefined prioritized list, and the processor with the highest priority succeeds and the rest fail.
- Sum, in which the sum of all the quantities is written (the sum-based write conflict resolution model can be extended to any associative operator defined on the quantities being written).

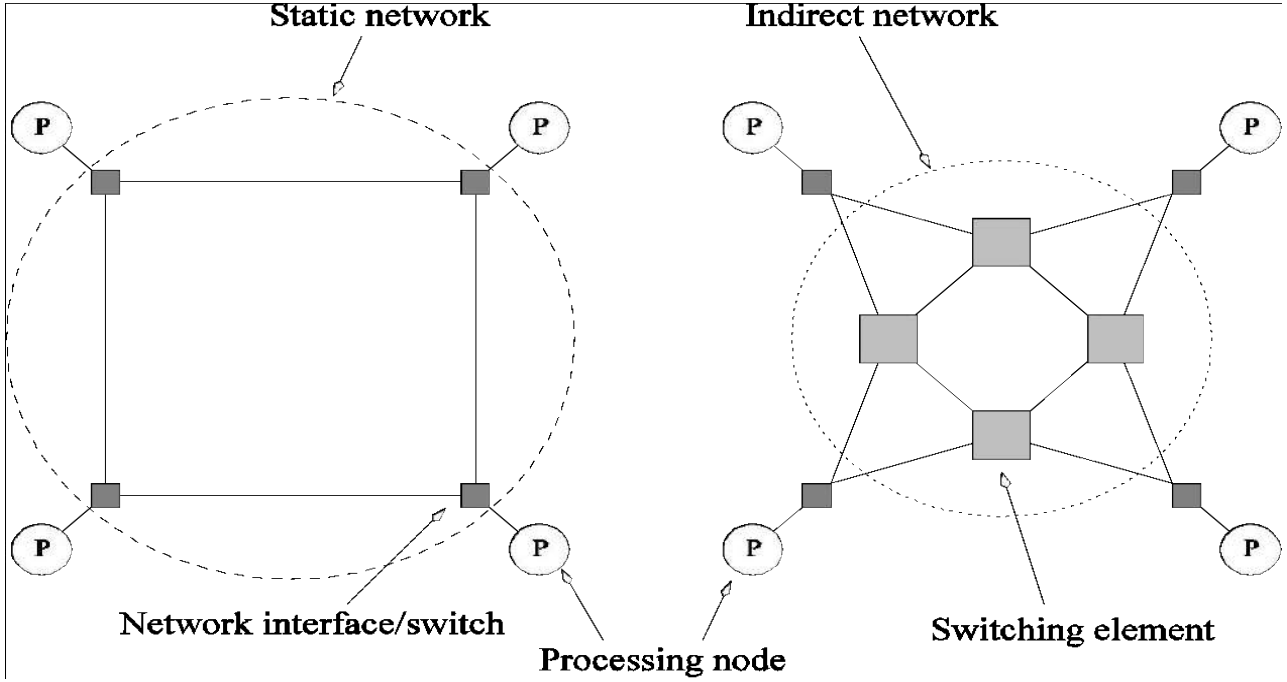
Physical Complexity of an Ideal Parallel Computer

- Consider the implementation of an EREW PRAM as a shared-memory computer with p processors and a global memory of m words.
- The processors are connected to the memory through a set of switches. These switches determine the memory word being accessed by each processor.
- In an EREW PRAM, each of the p processors in the ensemble can access any of the memory words, provided that a word is not accessed by more than one processor simultaneously.
- To ensure such connectivity, the total number of switches must be $\Theta(mp)$.
- For a reasonable memory size, constructing a switching network of this complexity is very expensive.
- Thus, PRAM models of computation are impossible to realize in practice.

Interconnection Networks for Parallel Computers

- Interconnection networks carry data between processors(processing nodes) and memory.
- Interconnects are made of switches and links (wires, fiber).
- For links based on conducting media, the capacitive coupling between wires limits the speed of signal propagation. This capacitive coupling and attenuation of signal strength are functions of the length of the link.

- 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.



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

Interconnection Networks for Parallel Computers

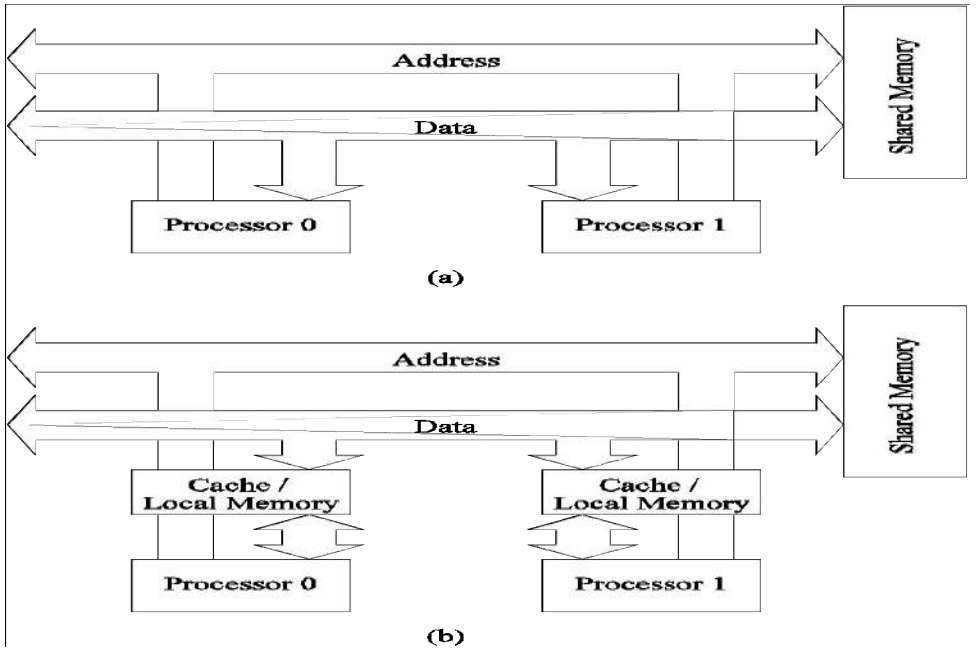
- Switches map a fixed number of inputs to outputs.
- The total number of ports on a switch is the *degree* of the switch.
- Switches may also provide support for internal buffering (when the requested output port is busy), routing (to alleviate congestion on the network), and multicast (same output on multiple ports).
- The mapping from input to output ports can be provided using a variety of mechanisms based on physical crossbars, multi-ported memories, multiplexers-demultiplexers, and multiplexed buses.
- 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.
- It typically has the responsibility of packetizing data, computing routing information, buffering incoming and outgoing data for matching speeds of network and processing elements, and error checking.
- The network interface may hang off the I/O bus or the memory bus.
- The relative speeds of the I/O and memory buses(can support higher bandwidth) impact the performance of the network.
- A variety of network topologies have been proposed and implemented having tradeoff between performance and 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 cost of the network scales linearly as the number of nodes, p.
- 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. Sun Enterprise servers and Intel Pentium based shared-bus multiprocessors are examples of such architectures.
- Since much of the data accessed by processors is local to the processor, a local memory can improve the performance of bus-based machines.

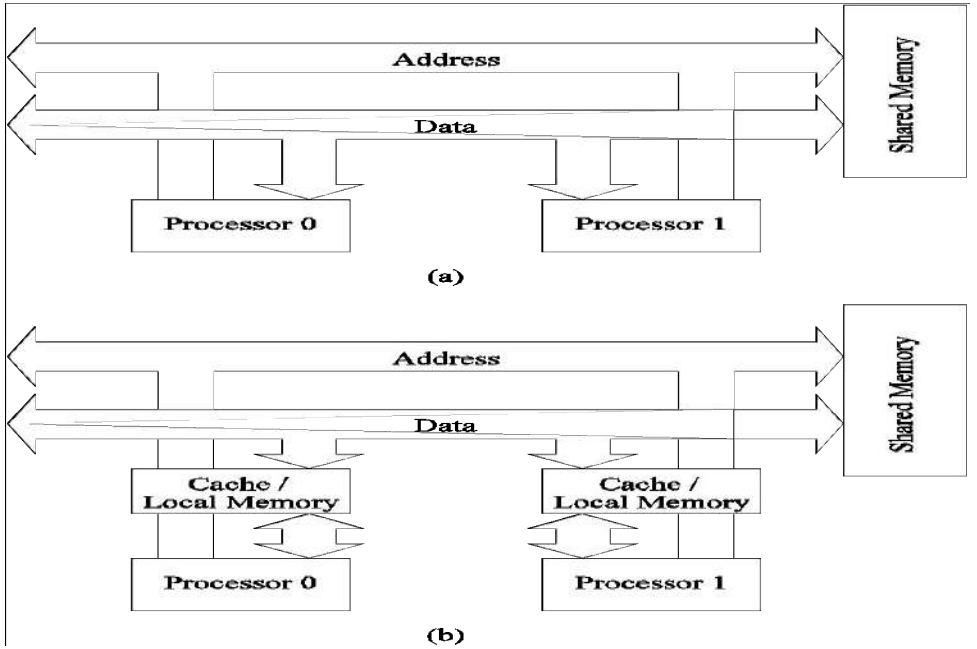


Bus-based interconnects (a) with no local caches; (b) with local memory/caches.

Network Topologies: Buses

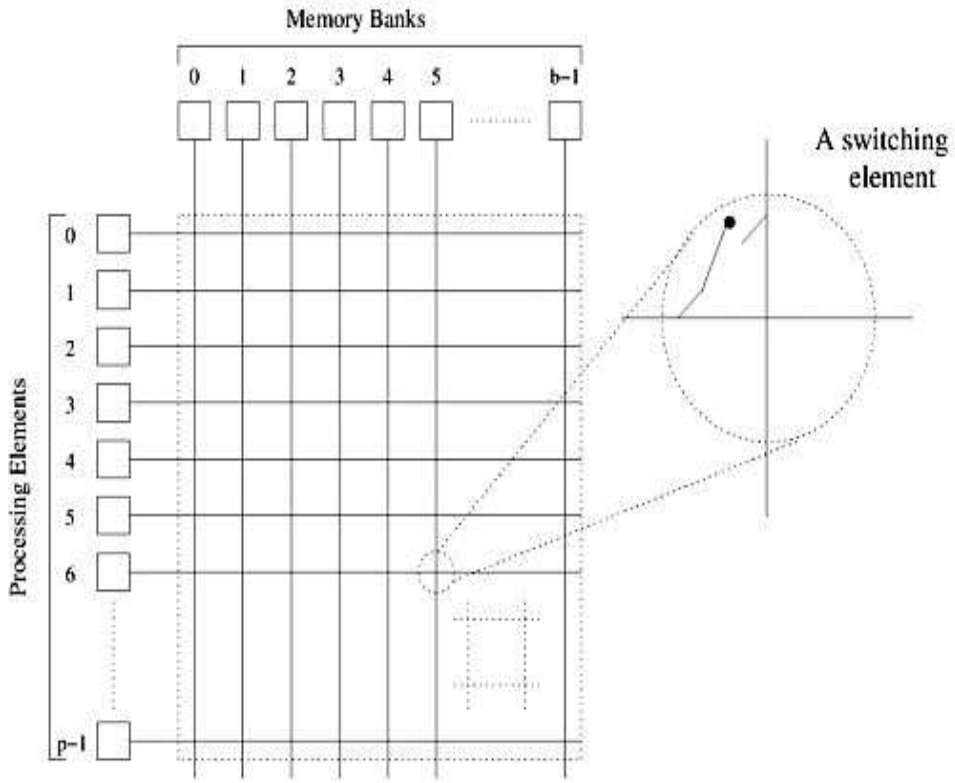
- Consider p processors sharing a bus to the memory.
- Assuming that each processor accesses k data items, and each data access takes time tcycle.
- The execution time is lower bounded by tcycle x kp seconds
- Let us assume that 50% of the memory accesses (0.5k) are made to local data.
- The total execution time is lower bounded by 0.5 x tcycle x k + 0.5 x tcycle x kp.
- As p becomes large, the organization of Figure (b) results in a lower bound that approaches 0.5 x tcycle x kp.
- This time is a 50% improvement in lower bound on execution time.

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



Network Topologies: Crossbars

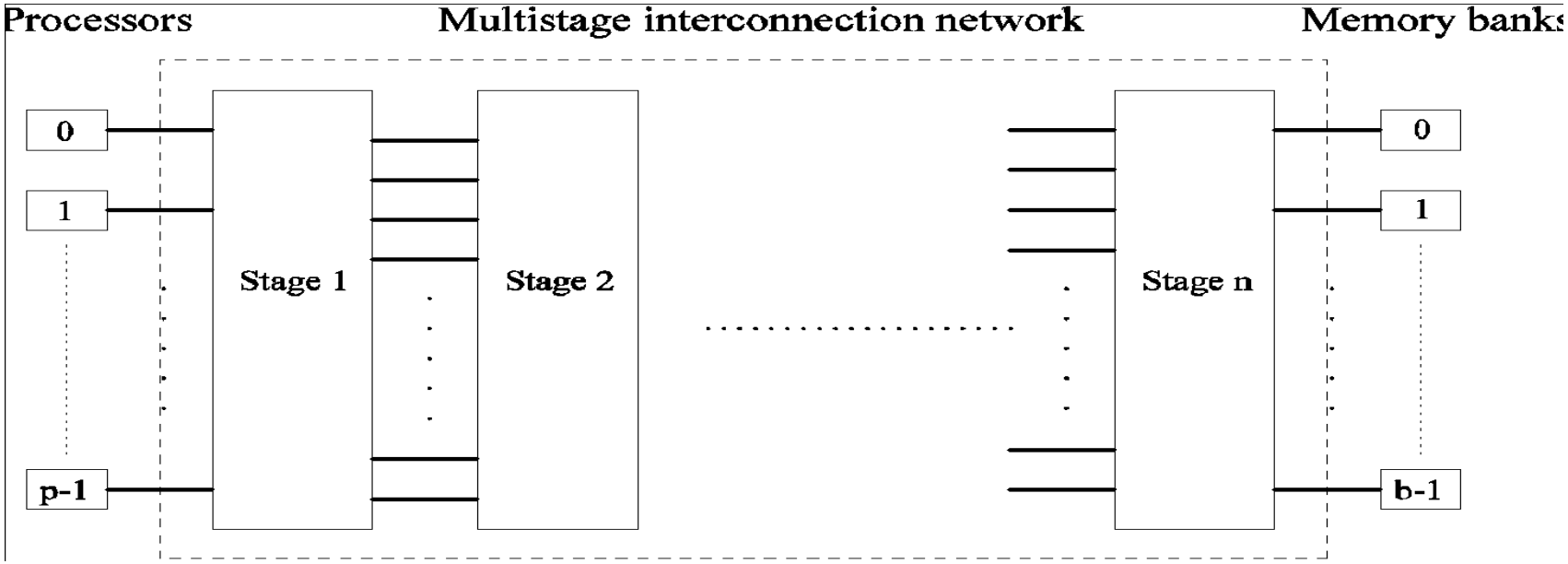
- A crossbar network uses an $p \times m$ grid of switches to connect p inputs to m outputs in a non-blocking manner.
- The crossbar network is a non-blocking network in the sense that the connection of a processing node to a memory bank does not block the connection of any other processing nodes to other memory banks
- The cost of a crossbar of p processors grows as $O(p^2)$.
- This is generally difficult to scale for large values of p.
- Examples of machines that employ crossbars include the Sun Ultra HPC 10000 and the Fujitsu VPP500.



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

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.



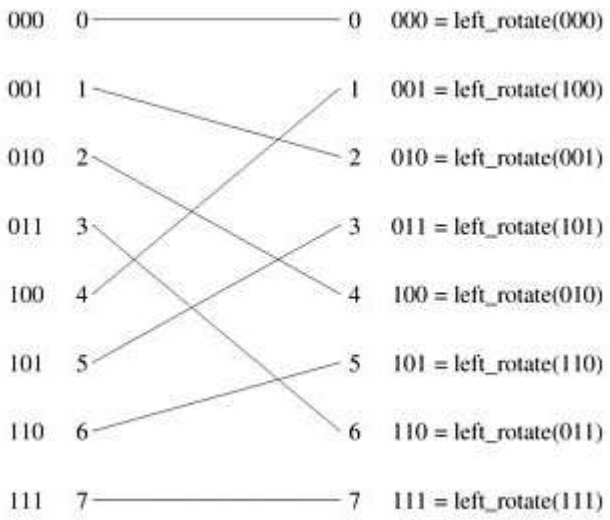
The schematic of a typical multistage interconnection network.

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:

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

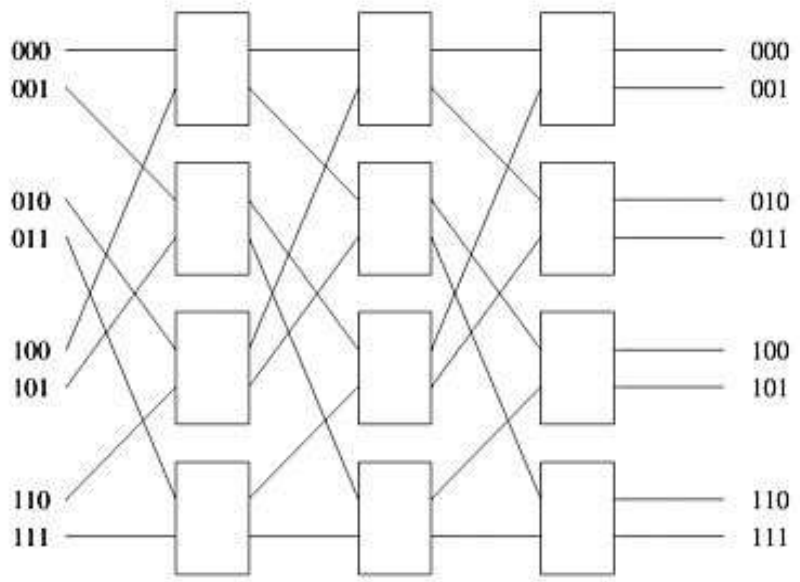
- Each stage of the Omega network implements a perfect shuffle as follows:



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

Network Topologies: Multistage Omega Network

- An omega network has $p/2 \times \log p$ switching nodes, and the cost of such a network grows as $(p \log p)$.
- A complete Omega network with the perfect shuffle interconnects and switches can now be illustrated:

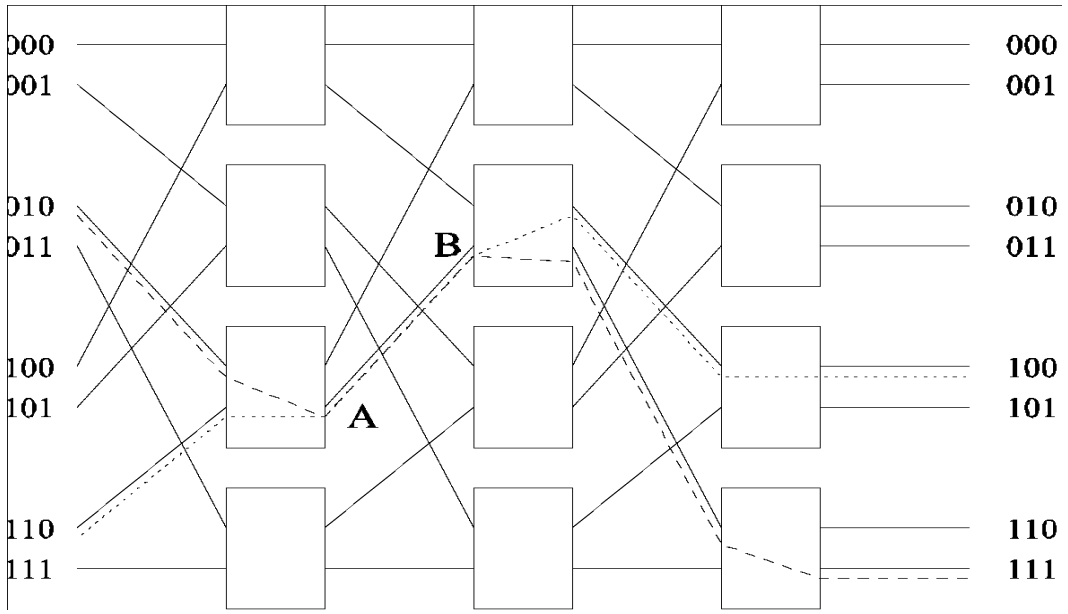


A complete omega network connecting eight inputs and eight outputs.

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.

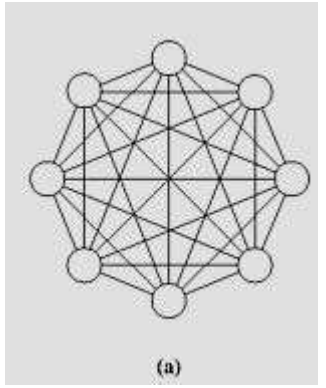
An example of blocking in omega network: one of the messages (010 to 111 or 110 to 100) is blocked at link AB.



Network Topologies: Completely Connected Network

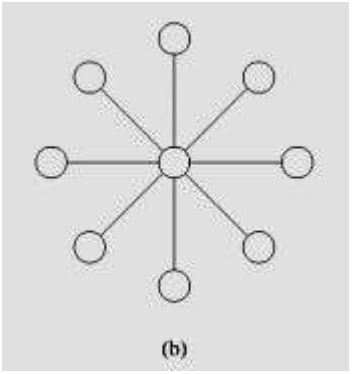
Completely Connected Network:

- 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 values of p.
- In this sense, these networks are static counterparts of crossbars.



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.

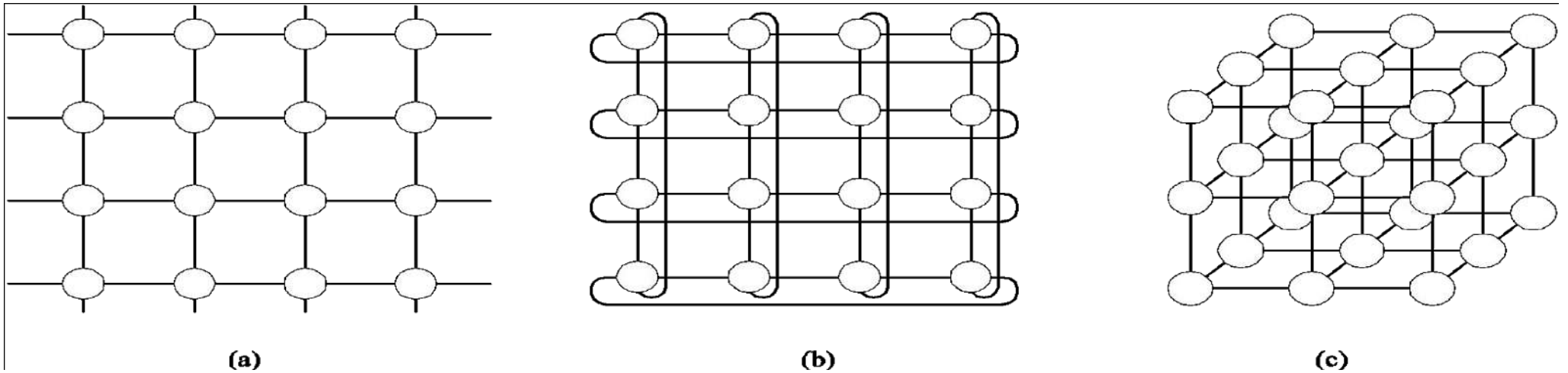


Network Topologies: Linear Arrays, Meshes, and *k-d* Meshes

- 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.

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

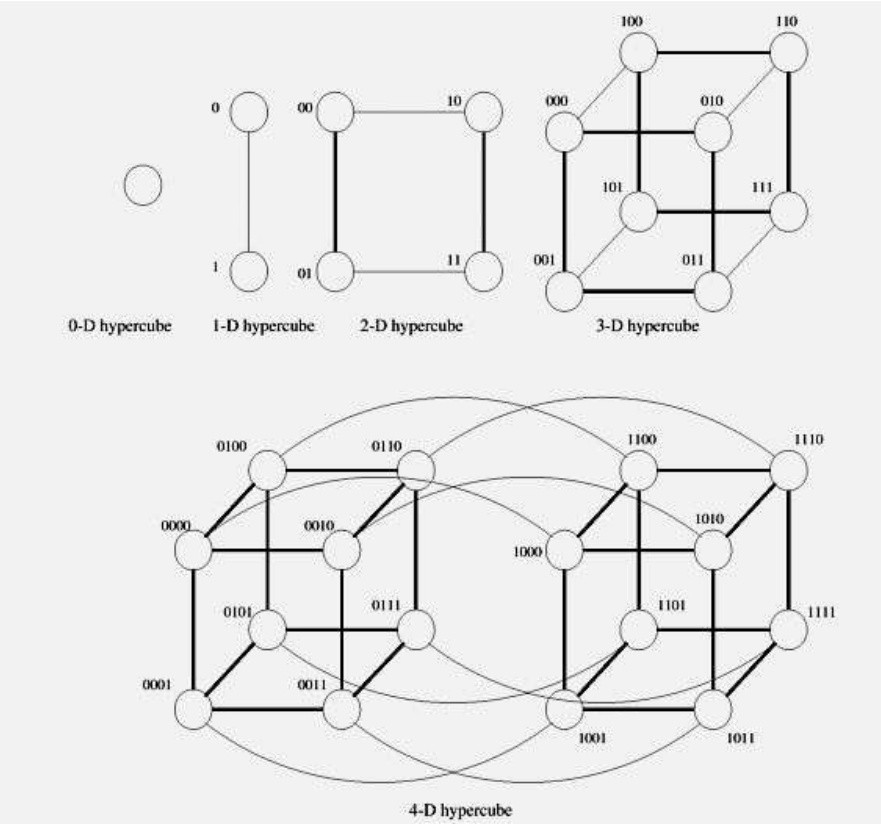
Network Topologies: Two- and Three Dimensional Meshes



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

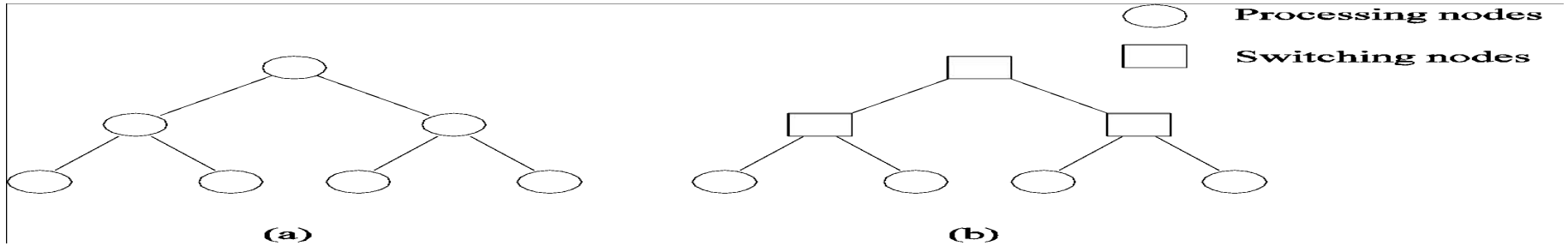
- The distance between any two nodes is at most *log p*.
- Each node has *log p* neighbors.
- The distance between two nodes is given by the number of bit positions at which the two nodes differ.



Construction of hypercubes from hypercubes of lower dimension.

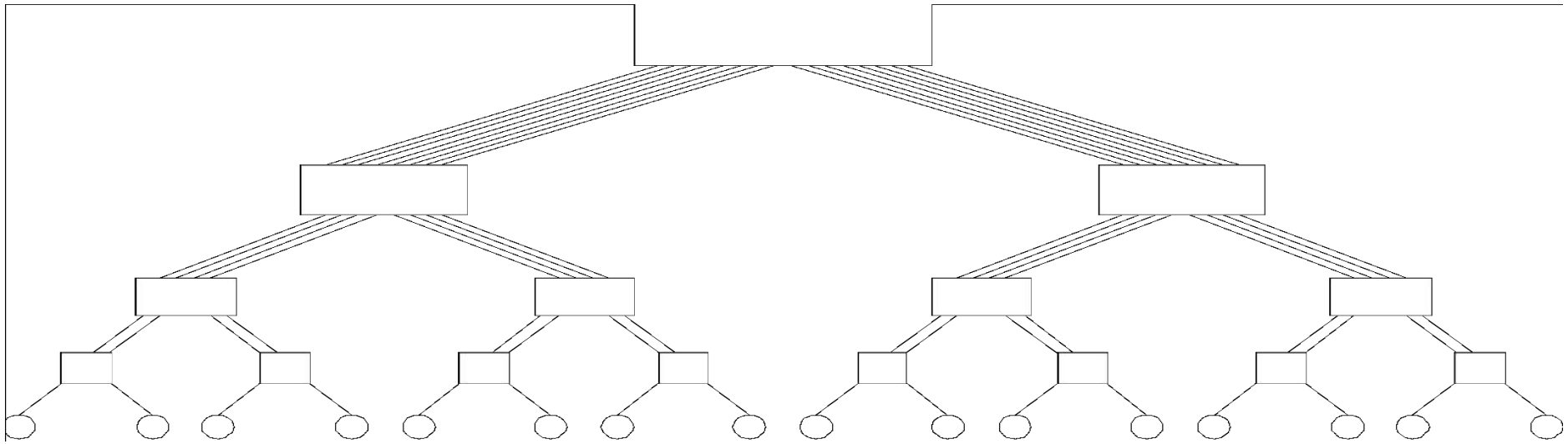
Network Topologies: Tree-Based Networks

- A tree network is one in which there is only one path between any pair of nodes.
- Both linear arrays and star-connected networks are special cases of tree networks.
- The distance between any two nodes is no more than $2 \log p$.
- Links higher up the tree potentially carry more traffic than those at the lower levels.
- For example,when many nodes in the left subtree of a node communicate with nodes in the right subtree, the root node must handle all the messages
- This problem can be alleviated in dynamic tree networks by increasing the number of communication links and switching nodes closer to the root.
- For this reason, a variant called a 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.



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

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 in to the cost.**

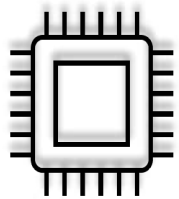
Evaluating Static Interconnection Networks

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	$p - 1$

Cache Coherence in Multiprocessor Systems

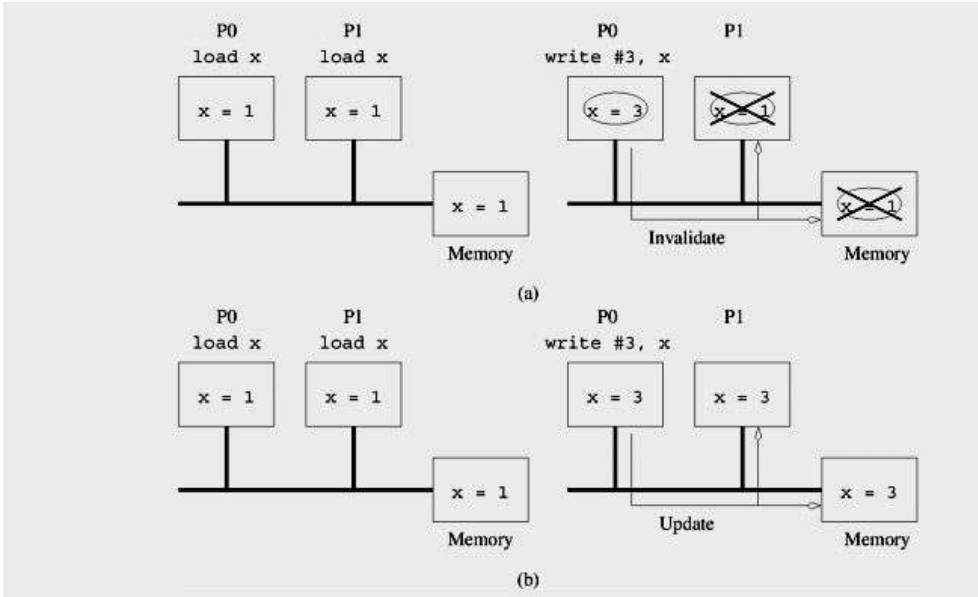


Topics to be discussed

- Cache Coherence
- Update and Invalidate protocols
- Snoopy Cache Systems
- Directory Based Systems

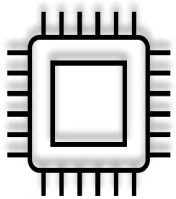
Cache Coherence in Multiprocessor Systems

- Interconnects provide basic mechanisms for data transfer.
- In the case of shared address space machines, additional hardware is required to coordinate access to data that might have multiple copies in the network.
  - When the value of a variable is changed, all its copies must either be invalidated or updated.
  - Failing this, other processors may potentially work with incorrect (stale) values of the variable.
- In an invalidate protocol, when a processor updates its part of the cache-line, the other copies of this line are invalidated.
- In an update protocol, whenever a data item is written, all of its copies in the system are updated.



Cache coherence in multiprocessor systems: (a) Invalidate protocol; (b) Update protocol for shared variables.

Cache Coherence: Update and Invalidate Protocols



- If a processor just reads a value once and does not need it again, an update protocol may generate significant overhead (latency at source and bandwidth on the network)
- When other processors try to update their parts of the cache-line, the line must actually be fetched from the remote processor.
- Both protocols suffer from false sharing overheads (False sharing refers to the situation in which different processors update different parts of of the same cache-line)
- The tradeoff between invalidate and update schemes is the classic tradeoff between communication overhead (updates) and idling (stalling in invalidates).
- Most current machines use invalidate protocols.

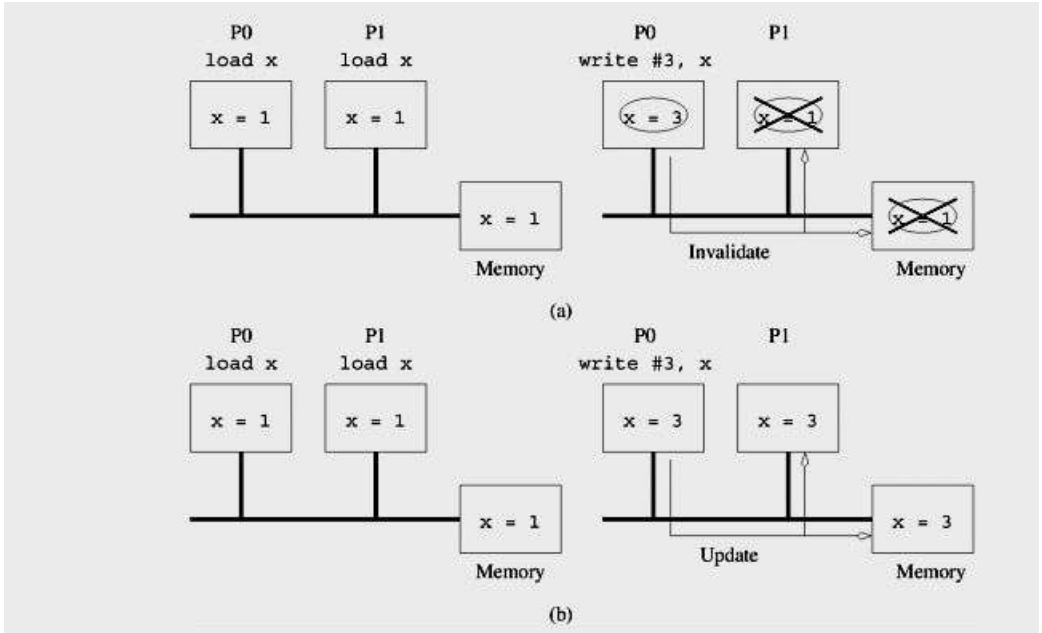
Maintaining Coherence Using Invalidate Protocols

- Initially the variable x resides in the global memory.
- The first step executed by both processors is a load operation on this variable.At this point, the state of the variable is said to be shared, since it is shared by multiple processors.
- When processor P0 executes a store on this variable, it marks all other copies of this variable as invalid. It must also mark its own copy as modified or dirty.
- This is done to ensure that all subsequent accesses to this variable at other processors will be serviced by processor P0 and not from the memory.
- At this point, say, processor P1 executes another load operation on x .
- Processor P1 attempts to fetch this variable and,

since the variable was marked dirty by processor P0, processor P0 services the request.

- Copies of this variable at processor P1 and the

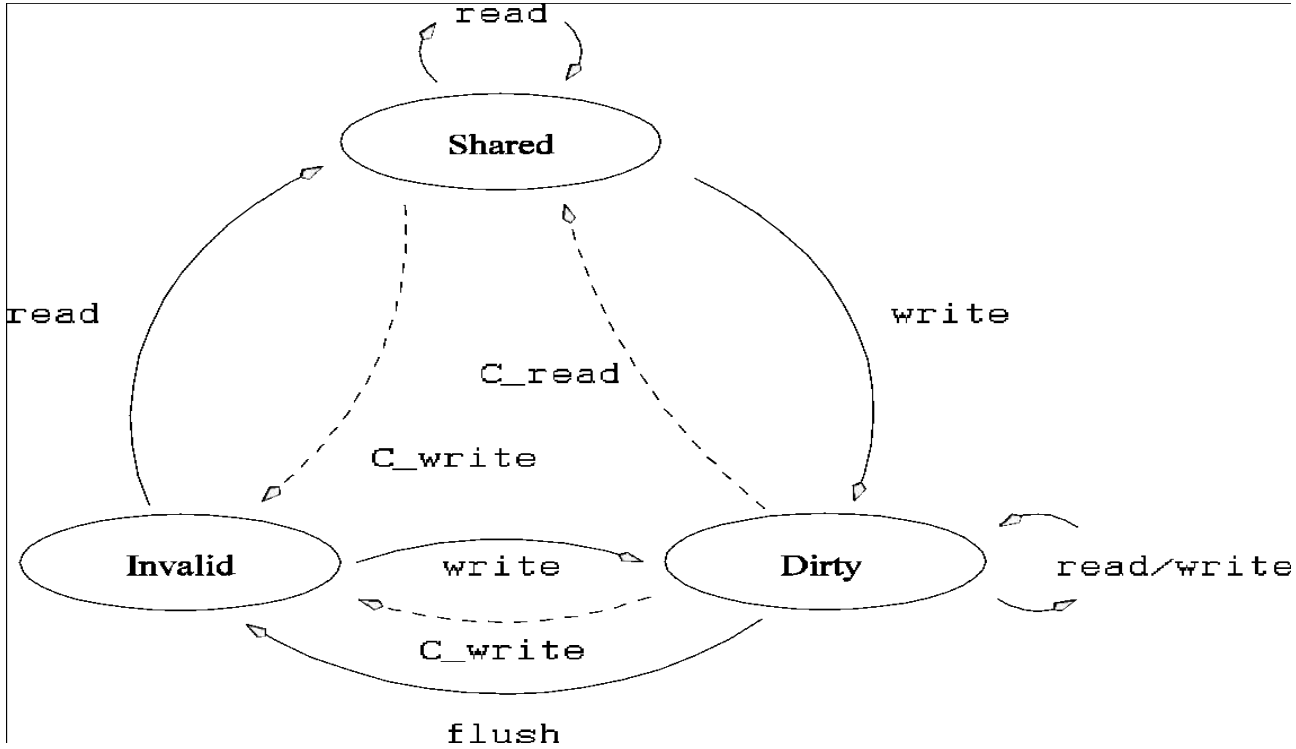
global memory are updated and the variable re-enters the shared state.



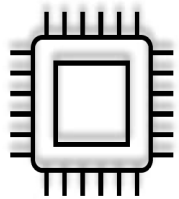
Maintaining Coherence Using Invalidate Protocols

- Each copy of a data item is associated with a state.
- One example of such a set of states is, shared, invalid or dirty.
- In shared state, there are multiple valid copies of the data item (and therefore, an invalidate would have to be generated on an update).
- In dirty state, only one copy exists and therefore, no invalidates need to be generated.
- In invalid state, the data copy is invalid, therefore, a read generates a data request (and associated state changes).

State diagram of a simple three-state coherence protocol.



Maintaining Coherence Using Invalidate Protocols



Time ↓	Instruction at Processor 0	Instruction at Processor 1	Variables and their states at Processor 0	Variables and their states at Processor 1	Variables and their states in Global mem.
					x = 5, D y = 12, D
	read x		x = 5, S		x = 5, S
		read y		y = 12, S	y = 12, S
	x = x + 1		x = 6, D		x = 5, I
		y = y + 1		y = 13, D	y = 12, I
	read y		y = 13, S	y = 13, S	y = 13, S
		read x	x = 6, S	x = 6, S	x = 6, S
	x = x + y		x = 19, D	x = 6, I	x = 6, I
	x = x + 1		x = 20, D		x = 6, I

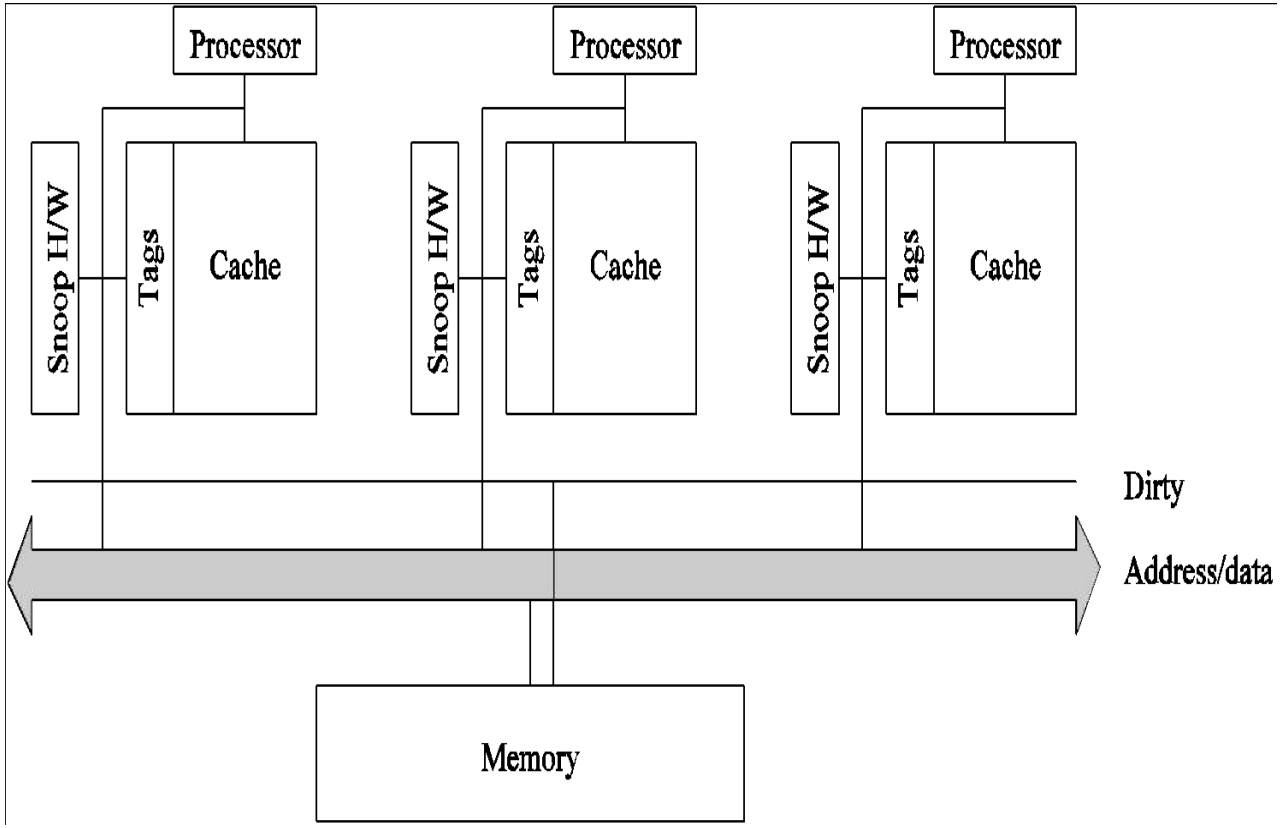
Example of parallel program execution with the simple three-state coherence protocol.

Snoopy Cache Systems

How are invalidates sent to the right processors?

In snoopy caches, there is a broadcast media(bus or ring) that listens to all invalidates and read requests and performs appropriate coherence operations locally.

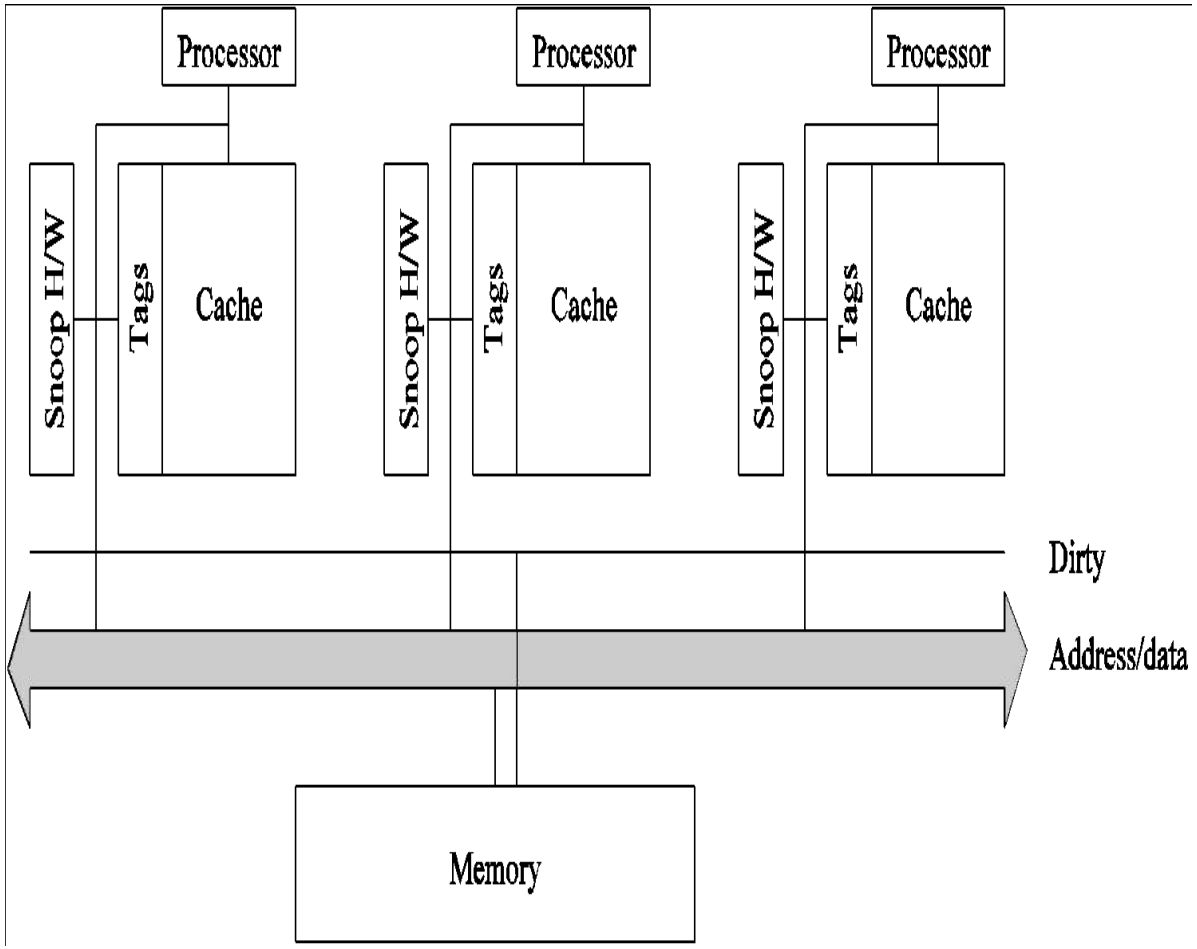
- Each processor's cache has a set of tag bits (which would be updated) associated with it that determine the state of the cache blocks.
- For instance, when the snoop hardware detects that a read has been issued to a cache block which was marked as dirty, it asserts control of the bus and puts the data out.
- Similarly, when the snoop hardware detects that a write operation has been issued on a cache block that it has a copy of, it invalidates the block.



A simple snoopy bus based cache coherence system.

Snoopy Cache Systems

- Once copies of data are tagged dirty, all subsequent operations can be performed locally on the cache without generating external traffic.
- If a data item is read by a number of processors, it transitions to the shared state in the cache and all subsequent read operations become local.
- If processors read and update data at the same time, they generate coherence requests on the bus .
- Bottleneck: Since a shared bus has a finite bandwidth, only a constant number of such coherence operations can execute in unit time.

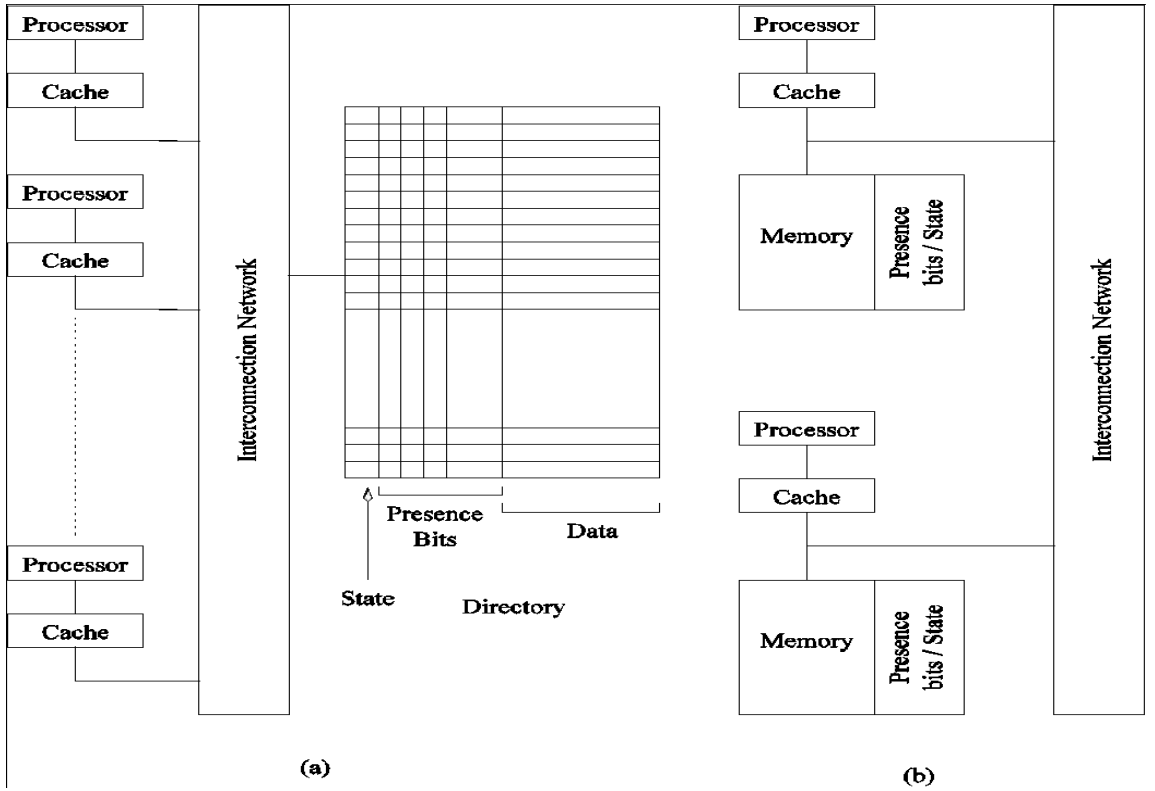


A simple snoopy bus based cache coherence system.

Directory Based Systems

- In snoopy caches, each coherence operation is sent to all processors. This is an inherent limitation.
- Why not send coherence requests to only those processors that need to be notified?
- This is done using a directory, which maintains a presence vector for each data item (cache line) along with its global state.

- The need for a broadcast media is replaced by the directory.
- The additional bits to store the directory may add significant overhead.
- The underlying network must be able to carry all the coherence requests.
- The directory is a point of contention, therefore, distributed directory schemes must be used.



Architecture of typical directory based systems: (a) a centralized directory; and (b) a distributed directory.

Communication Costs in Parallel Machines

- Along with idling and contention, communication is a major overhead in parallel programs.
- The cost of communication is dependent on a variety of features including the programming model semantics, the network topology, data handling and routing, and associated software protocols.

Message Passing Costs in Parallel Computers

- **The total time to transfer a message over a network comprises of the following:**
  - ***Startup time*** **(t_s):** **Time spent at sending and receiving nodes (executing the routing algorithm, programming routers, etc.).**
  - ***Per-hop time*** **(t_h):** **This time is a function of number of hops and includes factors such as switch latencies, network delays, etc.**
  - ***Per-word transfer time*** **(t_w):** **This time includes all overheads that are determined by the length of the message. This includes bandwidth of links, error checking and correction, etc.**

Store-and-Forward Routing

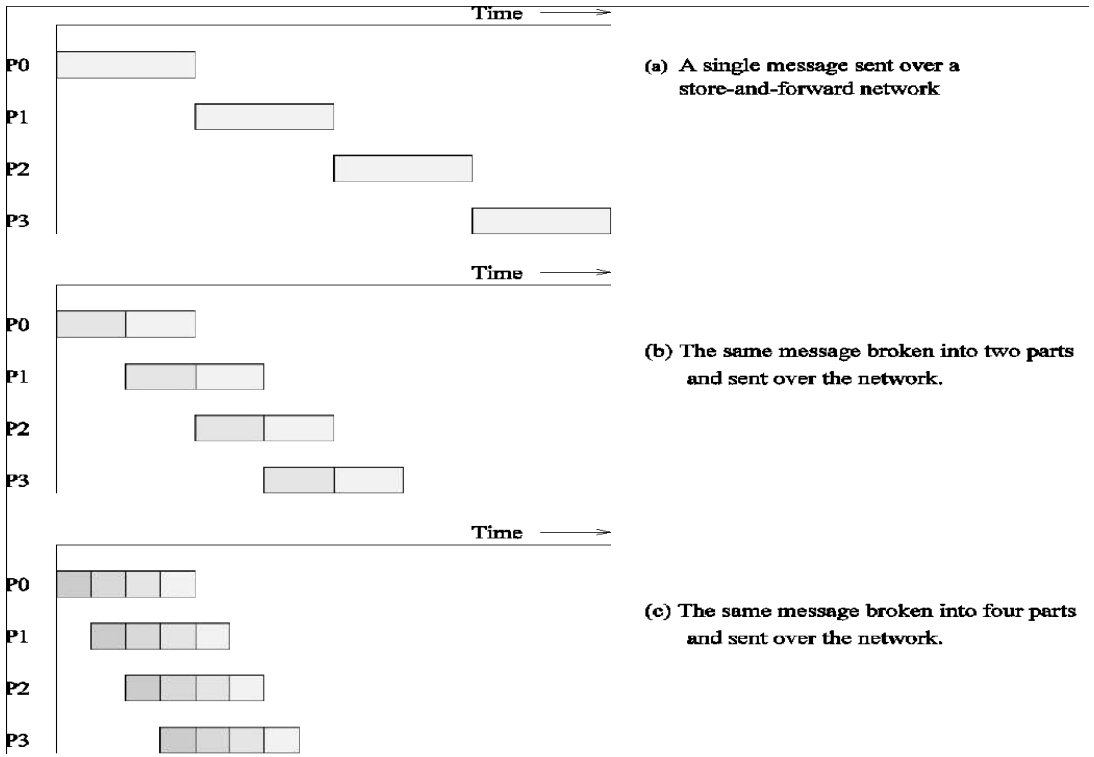
- A message traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop.
- The total communication cost for a message of size *m* words to traverse *l* communication links is

$$t_{comm} = t_s + (mt_w + t_h)l.$$

- In most platforms, t_h is small and the above expression can be approximated by

$$t_{comm} = t_s + mlt_w.$$

Routing Techniques



Passing a message from node P_0 to P_3 (a) through a store-and-forward communication network; (b) and (c) extending the concept to cut-through routing. The shaded regions represent the time that the message is in transit. The startup time associated with this message transfer is assumed to be zero.

Packet Routing

- Store-and-forward makes poor use of communication resources.
- Packet routing breaks messages into packets and pipelines them through the network.
- Since packets may take different paths, each packet must carry routing information, error checking, sequencing, and other related header information.
- The total communication time for packet routing is approximated by:

- The factor t_w accounts for overheads in packet headers.

$$t_{comm} = t_s + t_h l + t_w m.$$

Cut-Through Routing

- Takes the concept of packet routing to an extreme by further dividing messages into basic units called flits.
- Since flits are typically small, the header information must be minimized.
- This is done by forcing all flits to take the same path, in sequence.
- A tracer message first programs all intermediate routers. All flits then take the same route.
- Error checks are performed on the entire message, as opposed to flits.
- No sequence numbers are needed.

Cut-Through Routing

- **The total communication time for cut-through routing is approximated by:**

$$t_{comm} = t_s + t_h l + t_w m.$$

- **This is identical to packet routing, however, t_w is typically much smaller.**

Simplified Cost Model for Communicating Messages

- The cost of communicating a message between two nodes *l* hops away using cut-through routing is given by

$$t_{comm} = t_s + lt_h + t_w m.$$

- In this expression, t_h is typically smaller than t_s and t_w. For this reason, the second term in the RHS does not show, particularly, when *m* is large.
- Furthermore, it is often not possible to control routing and placement of tasks.
- For these reasons, we can approximate the cost of message transfer by

$$t_{comm} = t_s + t_w m.$$

Simplified Cost Model for Communicating Messages

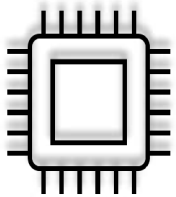
- It is important to note that the original expression for communication time is valid for only uncongested networks.
- If a link takes multiple messages, the corresponding t_w term must be scaled up by the number of messages.
- Different communication patterns congest different networks to varying extents.
- It is important to understand and account for this in the communication time accordingly.

Cost Models for Shared Address Space Machines

- While the basic messaging cost applies to these machines as well, a number of other factors make accurate cost modeling more difficult:
  - Memory layout is typically determined by the system.
  - Finite cache sizes can result in cache thrashing.
  - Overheads associated with invalidate and update operations are difficult to quantify.
  - Spatial locality is difficult to model.
  - Prefetching can play a role in reducing the overhead associated with data access.
  - False sharing and contention are difficult to model.
- We can use the same expression $t_{comm} = t_s + t_w m.$

to account for the cost of sharing a single chunk of m words between a pair of processors in both shared-memory and message passing paradigms with the difference that the value of the constant t_s relative to t_w is likely much smaller on a shared-memory machine than on a distributed memory machine (tw is likely 0 for UMA machine)

You have completed this topic, you should be able to:



- Discuss the trends in microprocessor architectures to use parallelism?
- Explain dichotomy of parallel computing platforms?
- Explain PRAM model?
- Discuss various Interconnection networks used in parallel machines?
- Evaluate static and dynamic interconnection networks?
- Explain Cache Coherence in Multiprocessor Systems?
- Explain the cost model for message passing and shared address pace paradigms?

References Used

- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar , —Introduction to Parallel Computing,Pearson Education, Second Edition, 2007.
- https://www.tutorialspoint.com/parallel_algorithm/parallel_random_access_machines.htm