



CHAPTER-10

Parallelism in Uniprocessor System

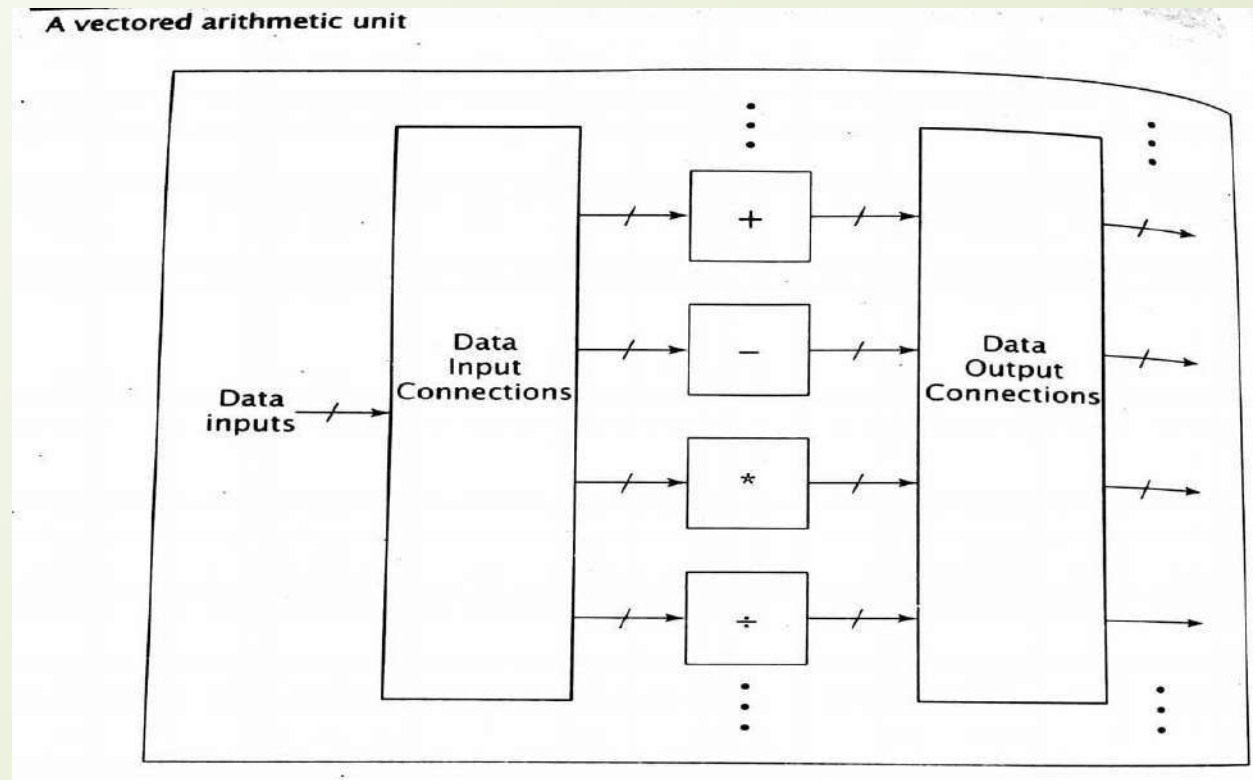
- A system that processes two different instructions simultaneously could be considered to perform parallel processing.
- A system that performs different operations on the same instruction usually would not.
- Example: consider the FETCH routine:
$$\text{FETCH2 : DR} \leftarrow \text{M}, \text{PC} \leftarrow \text{PC} + 1$$
- Although two micro-operations occur during this state, both are used to process same instruction. This is not considered parallel processing.
- Examples of parallelism in uniprocessor systems include:
 - => arithmetic pipeline
 - => instruction pipeline
 - => I/O processor

Contd...

- Consider a pipeline set to realize the function $A[i] \leftarrow B[i].C[i] + D[i]$.
- During the first clock cycle, the first stage receives the data inputs and calculates $B[i].C[i]$.
- Then in the second clock cycle, along with the value of $D[i]$, is sent to stage 3, which performs the addition, generating the desired result.
- Although arithmetic pipelines can perform many iterations of the same operation in parallel, they cannot perform different operations simultaneously.
- Vectored arithmetic unit performs different arithmetic operations in parallel.
- As shown in figure, vectored arithmetic unit contains multiple functional units; some perform addition, others subtraction, and other different function.
- Addition unit can input two numbers for addition and at the same time subtraction unit can input two numbers for subtraction, thus allowing CPU to execute both instructions simultaneously.

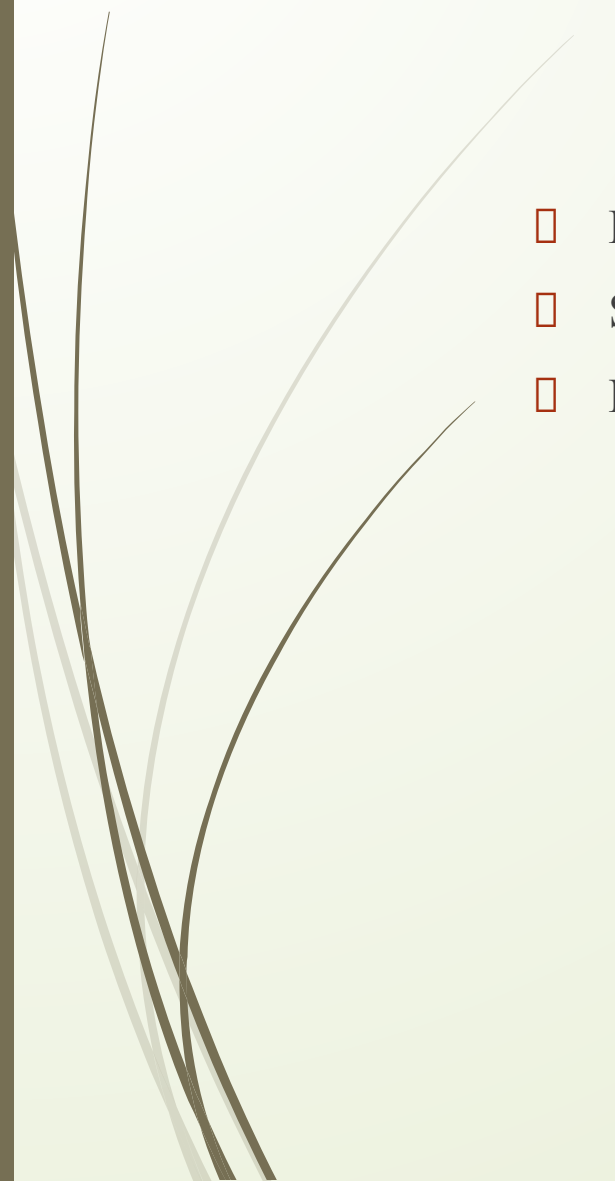
Contd...

- Input and output switches are needed to route the proper data to the correct arithmetic unit and to send the correct outputs to their proper destination.





Organization of Multi-Processor System:

- Flynn's Taxonomy
 - System Topologies
 - MIMD System Architecture
- 

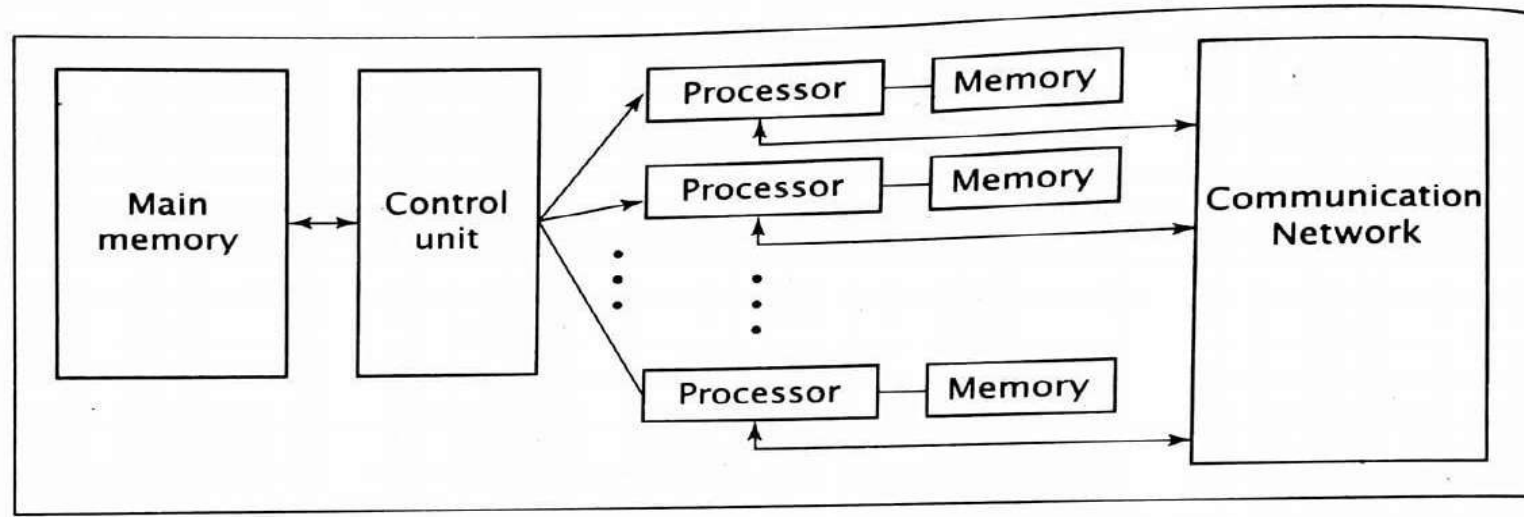


Flynn's Taxonomy:

- ❑ Based on instruction and data processing.
- ❑ Computer is classified by whether it processes a single instruction at a time or multiple instructions simultaneously, whether it operates on one or multiple data sets.
- ❑ Classified into four categories:
 - => SISD: Single Instruction Single Data
 - => SIMD: Single Instruction Multiple Data
 - => MISD: Multiple Instruction Single Data
 - => MIMD: Multiple Instruction Multiple Data
- ❑ SISD machine consists of a single CPU executing individual instructions on individual data values.
- ❑ MISD is not practical to implement. No significant MISD computers have even been built.

Contd...

A generic SIMD organization



- SIMD executes a single instruction on multiple data values simultaneously using many processors.
- Since, only one instruction is processed at any given time, it is not necessary for each processor to fetch and decode the instruction. Instead, a single control unit handles this task for all processors within the SIMD computer.
- Each processors may have local memory, but not mandatory.
- There is also a communication network that the processors use to communicate with each other.
- Although every processor receives the same control signals, they do not necessarily all execute the instruction; SIMD machine generally includes the capability to **mask**, or disable, individual processors



Contd...

- ❑ Systems referred to as **multiprocessors or multicomputer** are usually **MIMD**.
- ❑ Unlike SIMD machines, processors may execute different(i.e. multiple) instructions simultaneously.
- ❑ Therefore, each processor must include its own control unit.
- ❑ MIMD machines are well suited for general purpose use.

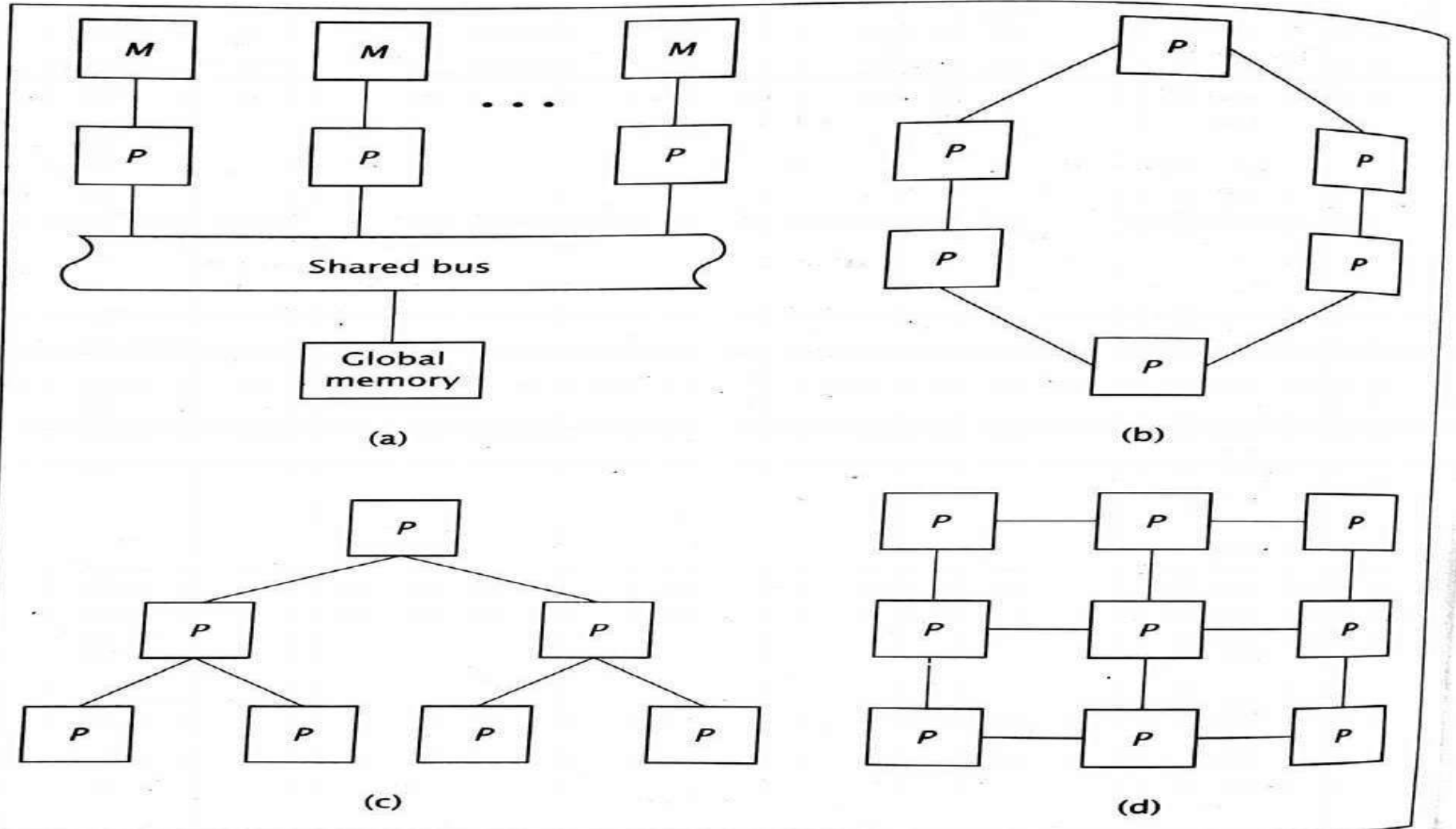


System Topologies/Interconnection Structures:

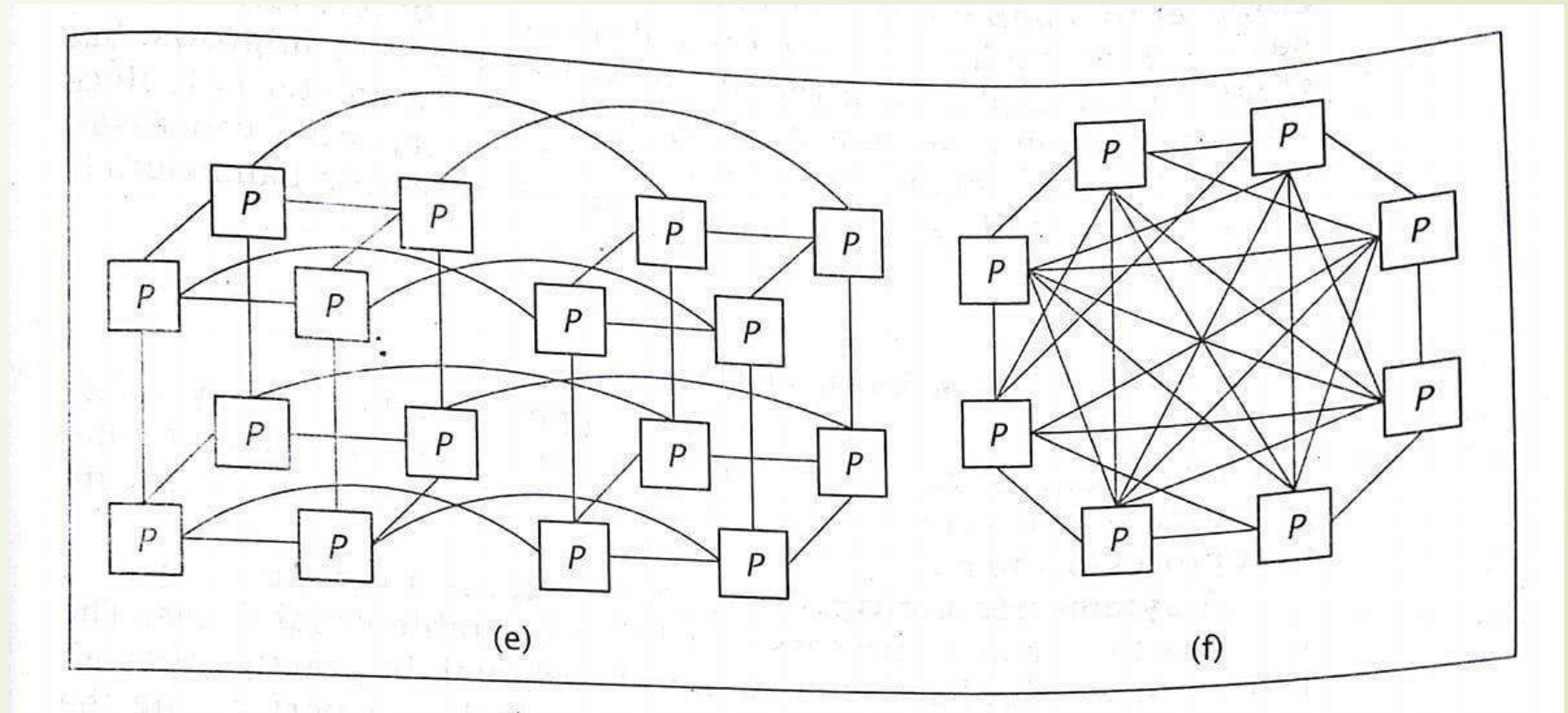
- Pattern of connections between the processors in a multiprocessor system.
- **Diameter-** maximum distance between 2 processors in the system.
- **Bandwidth-** capacity of communication link multiplied by the number of such links in the system.
- - A. Shared Bus
 - B. Ring Topology
 - C. Tree Topology
 - D. Mesh Topology
 - E. Hypercube Topology
 - F. Completely connected Topology

Contd...

MIMD system topologies: (a) shared bus, (b) ring, (c) tree, (d) mesh, (e) hypercube, and (f) completely connected



Contd...





Contd...

□ A. **Shared Bus**

- => Processor communicates each other via bus.
- => Bus can handle only one data at time.
- => Processors directly communicate with their own local memory to limit traffic on bus.
- => Easy to expand.
- => Diameter = 1 and Bandwidth = $1 \cdot L$

□ B. **Ring Topology**

- => Uses direct dedicated connections between processors unlike shared bus.
- => All communication line to be active simultaneously.
- => Data travels through several processors to reach final destination.
- => Diameter = $n/2$ (largest integer less than or equal to $n/2$)
- => Bandwidth = $n \cdot L$

Contd...

□ C. **Tree Topology**

=> Direct connections between processors; each processor has three connections.

=> Primary advantage is its low diameter.

=> Diameter = $2(\lg n)$

=> Bandwidth = $(n - 1).L$

□ D. **Mesh Topology**

=> Every processor connects to the processors above and below it, and to its right and left.

=> Without wraparound connection:

- Diameter = $2\sqrt{n}$ and Bandwidth = $(2n - 2\sqrt{n}) . L$

=> With wraparound condition

- Diameter = \sqrt{n} and Bandwidth = $2\sqrt{n} . L$



Contd...

□ E. **Hypercube**

=> Is a multidimensional mesh.

=> It has n processors, each with $(\lg n)$ connections.

=> Diameter = $(\lg n)$ and Bandwidth = $(n/2) \cdot (\lg n) \cdot L$

□ F. **Completely Connected Topology**

=> Each processor has $(n-1)$ connections.

=> Increases the complexity of processor.

=> Diameter = 1 and Bandwidth = $(n/2) \cdot (n - 1) \cdot L$

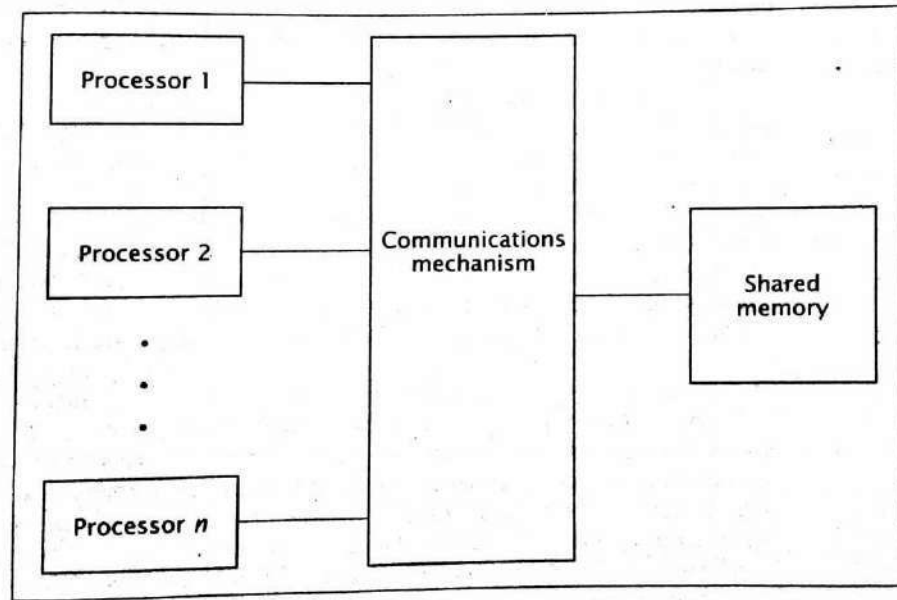


MIMD System Architectures:

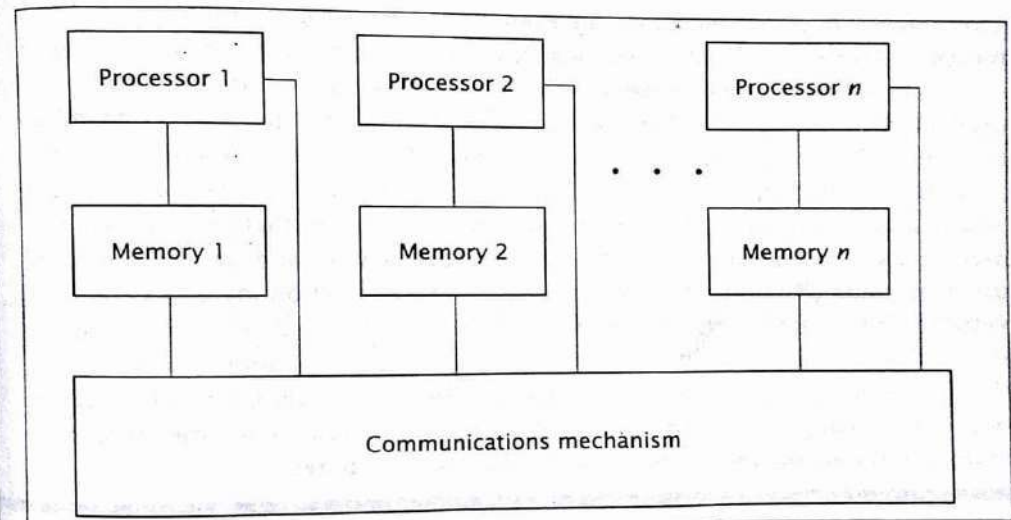
- ❑ Refers to the connection of the processors w.r.t system memory.
- ❑ A **Symmetric multiprocessor**, or **SMP**, is a computer system that has two or more processors with comparable capabilities(do not have to be identical).
 - => All processors must be capable of performing the same functions; this is the symmetry of SMPs.
 - => The processors all have access to the same I/O devices and memory modules.
- ❑ One type of SMP is the **Uniform Memory Access(UMA)** architecture.
- ❑ UMA gives all CPUs equal(uniform) access to all locations in shared memory.
- ❑ They interact with shared memory via some communication mechanism.
- ❑ Each processor may have its own cache memory, not directly accessible by the other processors.

Contd...

Uniform memory access (UMA) architecture



Nonuniform memory access (NUMA) architecture





Contd...

- ❑ **Non-uniform Memory Access(NUMA)** do not allow uniform access to all shared memory locations.
- ❑ The architecture still allows all processors to access all shared memory locations.
- ❑ However, each processor can access the memory module closest to it, its local shared memory, more quickly than the other modules; hence, the memory access times are non-uniform.
- ❑ Unlike UMA machines, NUMA computers may not be SMP.



Contd...

- ❑ The **cache coherence NUMA** architecture, **CC-NUMA**, is similar to the NUMA architecture , except each processor includes cache memory.
- ❑ The cache can buffer data from memory modules that are not local to the processor, thus reducing the access time of the most lengthy memory transfers.
- ❑ However, this introduces a problem when two or more caches hold the same piece of data.
- ❑ When one processor changes the value of this data, the other caches' values are invalid.
- ❑ There are several methods to maintain consistency, or **coherence**.

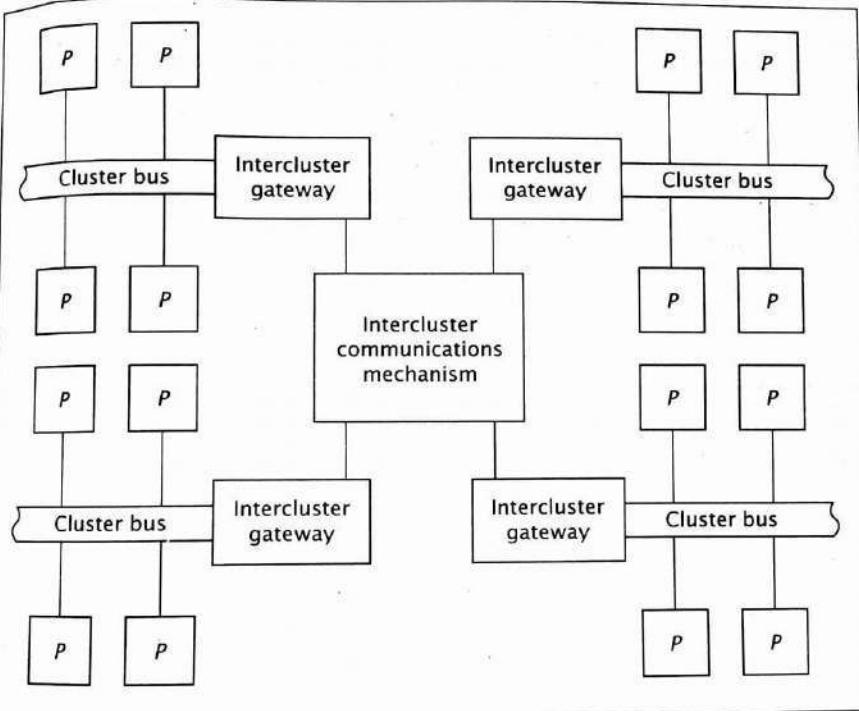


Communication in Multiprocessor Systems:

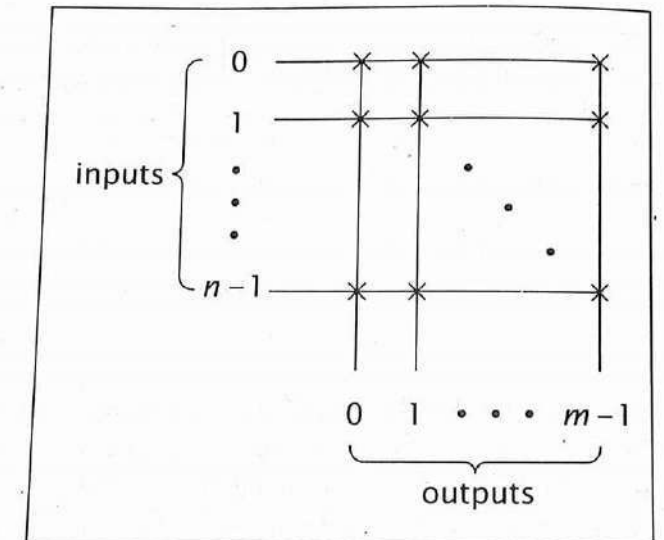
- Even the fastest processors cannot achieve high levels of efficiency if they spend too much of their time waiting to send and receive data.
- For this reason, multiprocessor systems use hardwired connections, rather than the data packets for communicating within the computer.
 1. Fixed connections
 2. Reconfigurable connections

Contd...

A 16-processor multiprocessor that uses clustering



An $n \times m$ crossbar switch



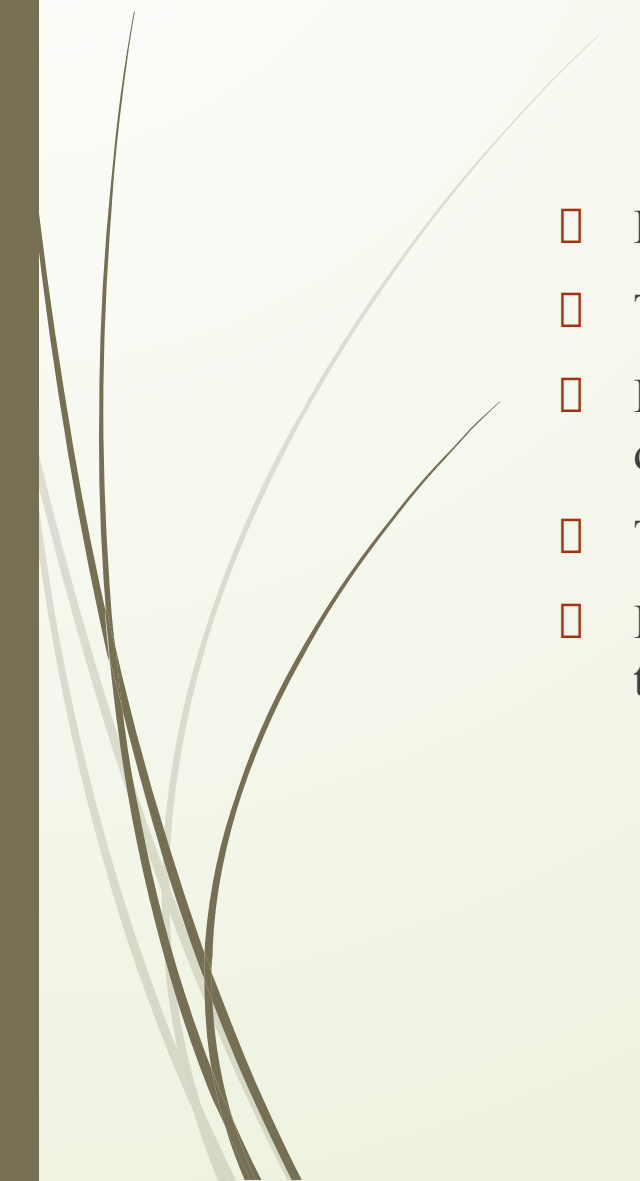



Fixed Connection:

- ❑ Inflexible system and is less costly than reconfigurable communications mechanism.
- ❑ A 16 processor multiprocessor system is divided into 4 clusters, each with 4 processors.
- ❑ Processors in each cluster are connected via their own shared bus, called the cluster bus.
- ❑ Two processors in the same cluster communicate via their cluster bus, leaving the intercluster communication mechanism and other cluster buses free for other communication.
- ❑ All cluster buses could transmit data within their clusters simultaneously, thus maximizing dataflow and minimizing processor delay.



Contd...

- Each cluster includes intercluster gateway, which handles data transfers between clusters.
 - These gateways are connected by an intercluster communication mechanism.
 - If a processor in one cluster must send data to a processor in a different cluster, it sends the data and destination processor information to its intercluster gateway.
 - The gateway examines the destination processor information to determine its cluster.
 - It then sends the data and destination through the intercluster communication mechanism, to the intercluster gateway of the destination processor's cluster.
- 



Reconfigurable connections: (crossbar switch)

- ❑ A crossbar switch has n inputs and m outputs; size = $n \times m$
- ❑ In practice, n and m usually have the same value.
- ❑ Each crosspoint within the switch is a connection point that can be closed to connect the input and output, or open to break the connection.
- ❑ There is one crosspoint for every possible input-output combination.
- ❑ In a multiprocessor system, the inputs are connected to the processors, and the outputs are connected to the memory modules or the I/O devices, or back to processors for interprocess communication.
- ❑ As the number of inputs and outputs increases, the size and hardware complexity of the crossbar switches increase rapidly.



Memory Organization in Multiprocessor Systems:

❑ **SHARED MEMORY:**

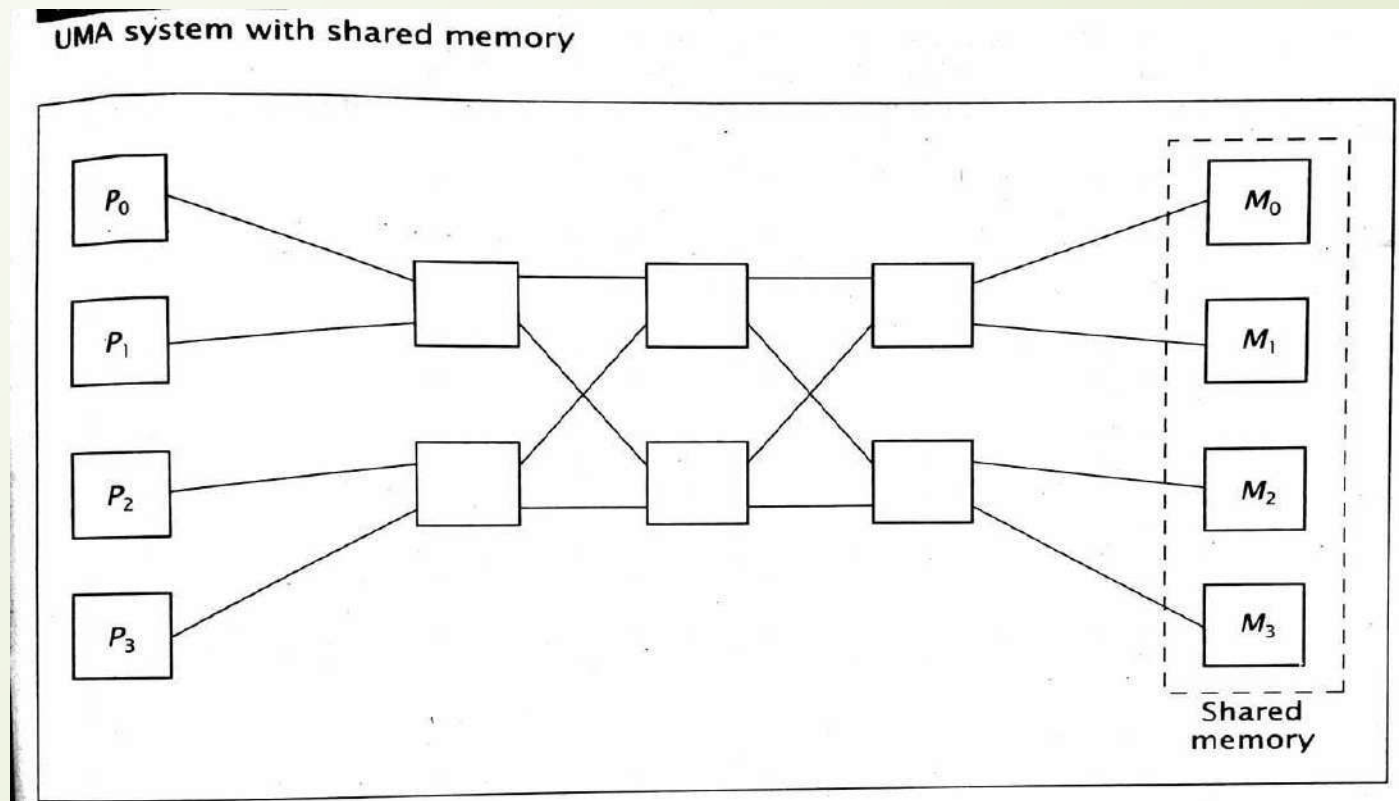
- ❑ The UMA and NUMA architecture both use shared memory.
- ❑ Through shared memory, the processors can access shared programs and data.
- ❑ They can also use the shared memory to communicate with each other via message passing.
- ❑ In direct message passing, without the use of shared memory, one processor sends a message directly to another processor, usually in a data packet.
- ❑ This requires some synchronization between the two processors, or some sort of buffer between the two.
- ❑ Instead of a separate buffer, a multiprocessor system can use shared memory



Contd...

- ❑ The first processor writes its message to the shared memory and signals the second processor that it has a waiting message.
- ❑ When the second processor is ready, it reads the message from shared memory.
- ❑ The location of the message in shared memory is either known beforehand or sent with the message waiting signal.
- ❑ In addition to message passing, the OS uses shared memory to store information about its current states.
- ❑ In UMA architecture of the following figure, it might first appear that all processors must try to access a single shared memory module, and that only one can be successful at any given time.
- ❑ In practice, the shared memory is partitioned into several modules, all of which can be accesses simultaneously.

Contd...





Contd...

(most significant problem in multiprocessor memory system design)

- ❑ Unlike uniprocessor systems, however, multiprocessors have individual caches for each processor.
- ❑ This can lead to problems when two or more caches hold the value of the same memory location simultaneously.
- ❑ As one processor stores a value to that location in its cache, the other cache will have an invalid value in its location.
- ❑ Using a write-through cache will not resolve this problem, since it would update main memory but not the other caches.
- ❑ In addition, the extra writes to main memory would decrease system performance.
- ❑ This is the **cache coherence** problem.

Contd...

- To illustrate this cache coherence problem, consider a multiprocessor system with four processors, each of which has a write-back cache.

Cache values illustrating the cache coherence problem

Action	Cache 0	Cache 1	Cache 2	Cache 3
Initial	1234: 56	1234: 56	1234: 56	1234: 56
Processor 0 updates 1234H	1234: 78	1234: 56	1234: 56	1234: 56
Processor 3 reads 1234H				Reads 56H instead of 78H



Contd...

- It is possible to resolve the cache coherence problem during program compilation.
- The compiler can mark all shared data as **non-cacheable**, thus forcing all accesses to this data to be from shared memory.
- Although, this resolves the problem, it lowers the cache hit ratio and reduces overall system performance.
- To reduce these effects and improve the hit ratio, a compiler may mark data as non-cacheable only at specified, critical parts of code.



Contd...

- ❑ Hardware solution scheme is the **cache directory**.
- ❑ A directory controller is integrated with the main memory; it maintains a cache directory in main memory, which contains information on the contents of local caches.
- ❑ All cache writes are also sent to the directory controller so that it can update the cache directory.
- ❑ When the processor writes data to its cache, the directory controller checks to see which other caches also have that data.
- ❑ It invalidates that data in those caches by marking its locations as empty.



Contd...

- One popular solution to the cache coherence problem is called **snooping**.
- In snooping, each cache(sometimes called snoopy cache) monitors memory activity on the system bus.
- Whenever it encounters a memory access(by another processor) to a location that it currently holds, it takes appropriate action.
 - => If the request is memory read, and the contents in its cache are the same as those in main memory, it simply notes that another cache also contains this data.
 - => If the request is memory read, and the contents in its cache are different than those in main memory, the processor must have written data to the cache that has not yet been written back to main memory.

The cache intercepts the memory read request, sending its data to both main memory and the cache which requested the data.
 - => If the request is the memory write, it simply marks its own data as invalid, essentially removing it from the cache.