NETWORK TOPOLOGIES:

Network topology refers to the ***arrangement of nodes and connections*** in a computer network. It defines how devices, ***such as computers, servers, and switches, are interconnected and communicate with each other.*** Here are some common network topologies:

1. **Bus Topology:** In a bus topology, all devices are ***connected to a central cable*** called a ***bus. Data is transmitted in both directions along the bus,*** and each device receives the transmitted data. However, if the bus cable fails, the entire network can be affected.

2. **Star Topology:** In a star topology, all ***devices are connected to a central device, usually a switch or hub.*** Each device has its ***own dedicated connection to the central device, forming a star-like shape.*** If one device fails, it does not affect the rest of the network, but the ***central device becomes a single point of failure.***

3. **Ring Topology:** In a ring topology, devices are connected in a ***closed loop,*** forming a ring. Each ***device is connected to two neighbouring devices,*** and ***data travels in a single direction around the ring. Token Ring is an example of a ring topology.*** If one device or connection fails, the entire network can be affected.

4. **Mesh Topology:** In a mesh topology, ***each device is connected to every other device in the network, creating a fully interconnected network.*** Mesh topologies ***provide redundancy and fault tolerance*** because ***multiple paths exist for data to travel.*** However, it requires a significant number of connections, making it ***costly and complex to implement.***

5. **Tree Topology:** Also known as a ***hierarchical topology,*** the tree topology resembles a hierarchical structure. It ***consists of multiple star topologies connected to a central bus.*** This topology is ***useful in large networks*** where ***scalability and organization*** are important.

6. **Hybrid Topology:** A hybrid topology is a combination of two or more different topologies. For example, a network could have a combination of star and bus topologies or a mix of star and ring topologies. Hybrid topologies are often ***used to take advantage of the benefits of multiple topologies*** and cater to specific network requirements.

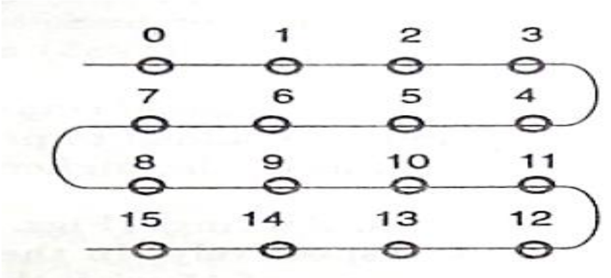
These are just a few examples of network topologies. There are other variations and hybrid forms that can be used based on specific network requirements, such as ***performance, scalability, fault tolerance,*** and ***cost considerations.***

1. STATIC CONNECTION NETWORKS

These networks are called static because their ***topology or structure remains fixed throughout the execution*** in parallel processing system. Once the network is established, the ***arrangement and connections between processors do not change dynamically during runtime.***

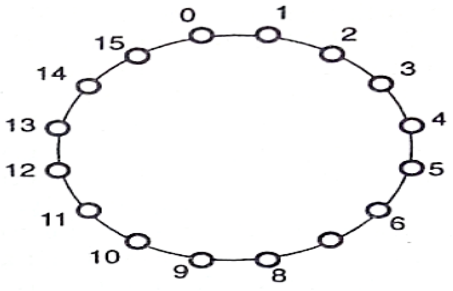
In these networks, the topology, is described using ***network parameters.*** **Here are some key static connection network topologies:**

1. **Linear Array:** It is a simple ***one-dimensional network where N nodes are connected in a line by N-1 links.*** The ***internal nodes have a degree of 2,*** while the ***terminal nodes have a degree of 1.*** Linear arrays are the most basic connection topology and are economical for small N values.

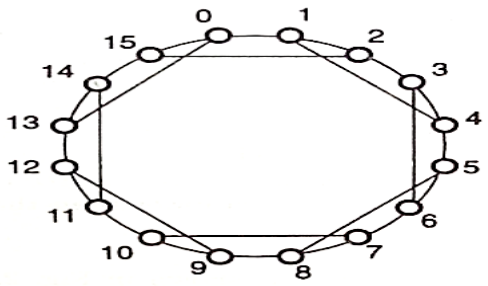


(a) Linear array

2. **Ring and Chordal Ring:** A ***ring is formed by connecting the two terminal nodes of a linear array with an extra link.*** It can be ***unidirectional or bidirectional. Increasing the node degree in a ring*** creates chordal rings, which have ***higher connectivity*** and ***shorter network diameter.***

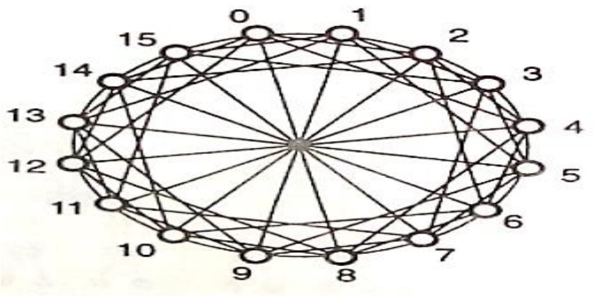


(b) Ring



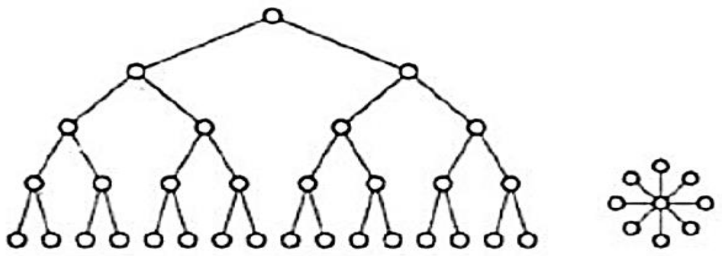
(c) Chordal ring of degree 3

3. **Barrel Shifter: It** is obtained from a ring by ***adding extra links from each node to other nodes at specific distances.*** Barrel shifters have increased connectivity compared to chordal rings of lower node degrees.



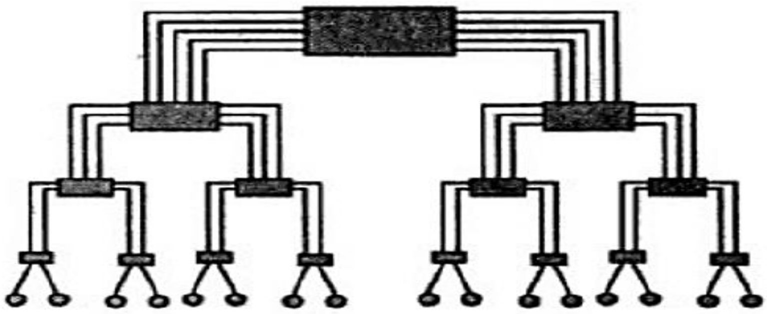
(e) Barrel shifter

4. **Tree and Star:** A ***binary tree*** is a ***scalable architecture*** with a ***constant node degree*** and ***hierarchical structure.*** The star architecture has a ***centralized supervisor node (Switch or Hub)*** and a high node degree. It is used in systems requiring ***centralized control.***



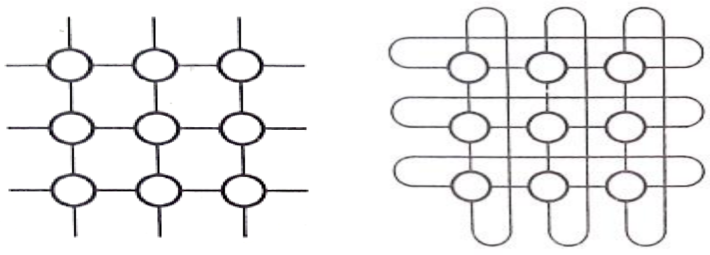
(a) Binary tree (b) Star

5. **Fat Tree:** In a fat tree, the ***channel width increases from the leaves to the root, resembling a real tree.*** It helps overcome the bottleneck problem seen in conventional binary trees. ***High Bandwidth and Low Latency, Scalability and Fault Tolerance, Uniform Traffic Distribution.***



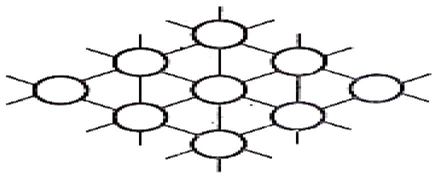
(c) Binary fat tree

6. **Mesh and Torus:** A mesh ***network consists of nodes arranged in a grid,*** where each node is connected to its adjacent nodes. ***Torus is a variant of mesh that combines ring and mesh topologies,*** extending to higher dimensions.



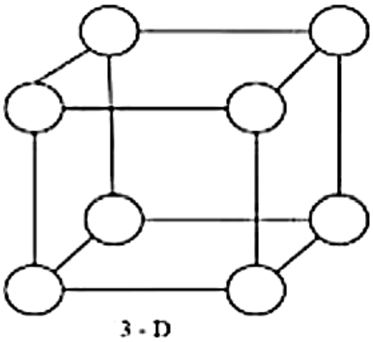
(a) Mesh (c) Torus

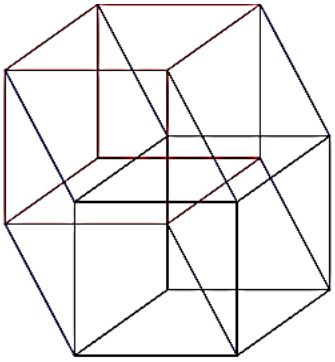
7. **Systolic Arrays:** These are ***multidimensional pipelined array architectures*** designed for implementing ***fixed algorithms.*** They ***match the communication structure of the algorithm*** and are used in applications like ***matrix multiplication.***



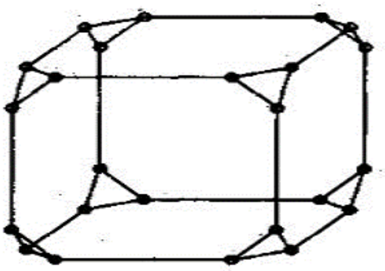
(d) Systolic array

8. **Hypercubes:** Hypercubes are ***multidimensional networks where each node is connected to its corresponding nodes in adjacent dimensions.*** The ***node degree increases linearly*** with the dimension, making it ***less scalable.***





9. **Cube-Connected Cycles:** This architecture is derived from hypercubes by ***replacing certain nodes with rings,*** forming cycles. Cube-connected cycles ***offer improved connectivity compared to hypercubes.***



In a static connection network, the ***links between nodes remain fixed, and data is routed along these predetermined connections.*** Each static connection network has its own advantages and limitations, the choice of network depends on the specific requirements of the application or system.

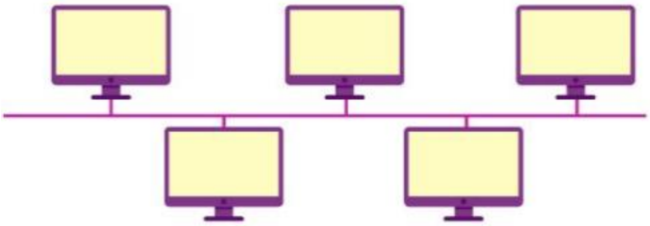
2. EXPLAIN DYNAMIC CONNECTION NETWORKS.

Dynamic connection networks, as the name suggests, are networks where the ***connections between processors can be dynamically established or changed during runtime.*** This flexibility allows for ***more adaptive and efficient communication patterns.*** Dynamic connection networks ***facilitate data transactions*** among ***processors, memory modules, and peripheral devices.*** Three main types are discussed: ***Digital Buses, Switch Modules,*** and ***Multistage Networks.***

1. **Digital Buses:**

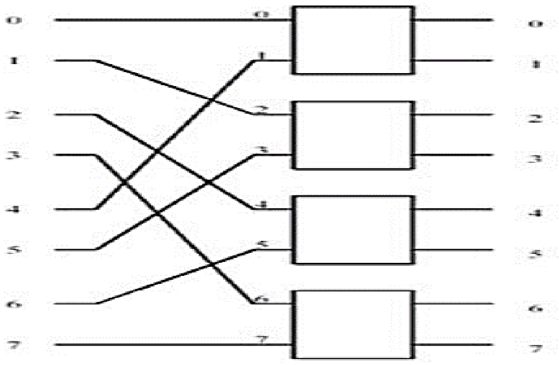
    - A bus system involves ***wires and connectors*** for data transactions between components.
    - ***Handles one transaction at a time*** between a ***source and destination.***
    - In case of multiple requests, ***bus logic allocates or deallocates the bus one at a time.***

- ***Lower cost and limited bandwidth*** compared to other dynamic networks.
- The system bus provides a ***common communication path*** on a PCB, connecting processors, memories, and peripherals.
- ***Master devices generate requests, and slave devices respond to these requests.***



2. **Switch Modules:**

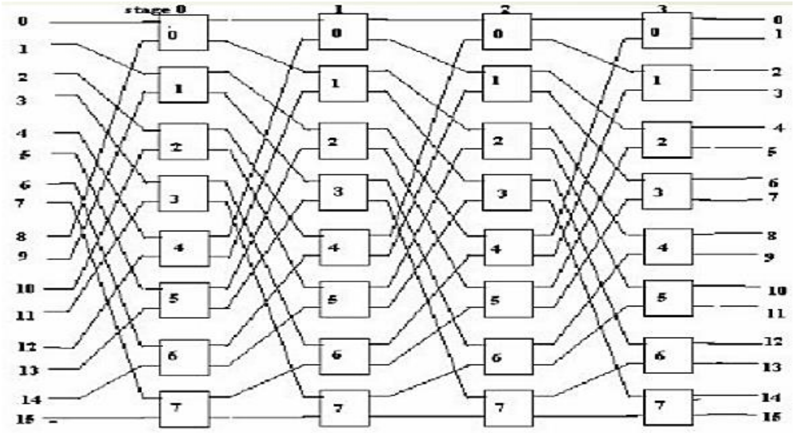
- A switch module, like an *a*b switch module, has ***a*b inputs and b outputs.***
- ***Binary switch*** corresponds to a 2*2 switch module.
- Allows ***one-to-one and one-to-many mappings;*** conflicts arise in ***many-to-one mappings.***
- N*N crossbar switch is ***formed when one-to-one mapping is allowed.***
- N*N crossbar can ***achieve n! permutations*** in general.



3. **Multistage Networks (MIN):**

- ***Utilize a number of a*b switches*** in each stage.
- ***Fixed interstage connections*** between switches, ***dynamically set to establish desired connections.***
- Different classes of MINs differ in ***switch modules used and interstage connection (ISC) patterns.***
- Examples include the ***Omega network:***
    - A ***16*16 Omega network requires four stages*** each of 16 switches.

- Each ***stage needs n/2 switch modules.*** (i.e., 8 Switch modules)
- Provides a ***structured way to connect 16 inputs to 16 outputs*** with ***dynamic configurations.***



In summary, dynamic connection networks enable ***flexible data transactions in parallel processing systems. Digital Buses are cost-effective*** but have ***limited bandwidth.*** Switch Modules offer ***various mappings and permutations.*** Multistage Networks ***use a series of switches with dynamically set connections***, with examples like the Omega network providing structured interconnections.

3. COMPARE STATIC CONNECTION NETWORK AND DYNAMIC CONNECTION NETWORKS.

Aspect	Static Connection Networks	Dynamic Connection Networks
Topology	Fixed topology	Dynamic topology
Connections	***Pre-established*** and ***remain static***	Can be ***established, modified, or terminated during runtime***
Flexibility	***Limited flexibilit***y and ***adaptability***	High flexibility and adaptability
Application Suitability	Suitable for applications with ***stable and predictable communication patterns***	Suitable for applications with ***dynamic and changing communication requirements***
Connection Management	***Less overhead*** in managing and maintaining connections	***Higher overhead*** in managing and maintaining connections due to runtime changes
Scalability	***Lower scalability*** as adding or removing nodes requires changes in topology	***Higher scalability*** as nodes can be added or removed without significant changes in topology

Aspect	Static Connection Networks	Dynamic Connection Networks
Fault Tolerance	Lower fault tolerance	Higher fault tolerance as ***dynamic connections can be rerouted in case of failures***
Usage	used in ***traditional wired networks***	used in ***wireless networks,distributed systems***
Examples	***Buses, star networks, ring networks***	***Mesh networks, peer-to-peer networks, mobile networks***

4. DISCUSS NETWORK PROPERTIES AND ROUTING

The network properties and routing are essential factors in determining the ***efficiency and performance of an interconnection network.*** There are several key aspects to consider such as:

1. Topology:

- Interconnection networks can be categorized as static or dynamic.
- Static networks consist of ***point-to-point direct connections*** that do not change during the execution.
- Dynamic networks are ***implemented with switched channels,*** such as ***digital buses, crossbar switches, and multistage networks.***

2. Node Degree & Network Diameter:

- **Node Degree:** Refers to the ***number of edges connected to a node.*** In ***unidirectional channels,*** it's divided into ***in-degree and out-degree.***
- **Network Diameter:** Represents the ***maximum shortest path*** between any two nodes. ***Minimizing diameter is crucial for efficient communication.***

3. Bisection Width:

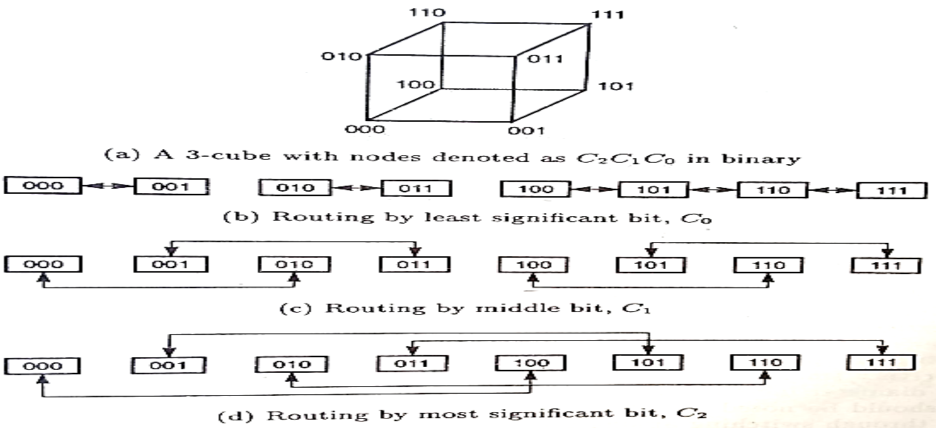
- Refers to the ***minimum number of edges*** along the cut when a network is divided into two equal halves. A ***smaller bisection width indicates better communication efficiency.***

4. Data-Routing Functions:

- **Data-Routing Concept**: Involves the ***movement of data*** among Processing Elements (PEs) within a parallel processing system.
- **Common Functions Include:**

- **Shifting:** Moving data elements in a ***specified direction,*** useful for ***reordering or aligning data.***
- **Rotation:** Circular movement of data elements, often used for ***circular buffers.***
- **Permutation:** Arrangements of objects, with cyclic notation specifying the permutation function. Understanding permutations is crucial for developing ***efficient routing algorithms.***
- **Shuffle:** ***Redistributing data elements,*** typically for ***specific patterns or configurations.***
- **Exchange:** ***Swapping data*** between two PEs, facilitating ***direct communication.***
- **Perfect Shuffle:** Involves ***shifting 1 bit to the left*** and ***wrapping around the most to least significant position.*** Fundamental for many interconnection networks.
- **Broadcast:** ***One-to-all mapping.*** Commonly treated as a ***global operation in a multicomputer.***
- **Multicast:** ***Many-to-many mapping.*** Involves ***sending messages to selected receivers.*** Personalized broadcast sends personalized messages to specific destinations.

5. Hypercube Routing Functions: Hypercube networks use ***binary cube structures.*** Specific routing functions are defined by ***different bits present in the node address,*** allowing ***data exchange based on specific bit differences.***



(a) A 3-cube with nodes denoted as $C_2C_1C_0$ in binary

(b) Routing by least significant bit, C_0

(c) Routing by middle bit, C_1

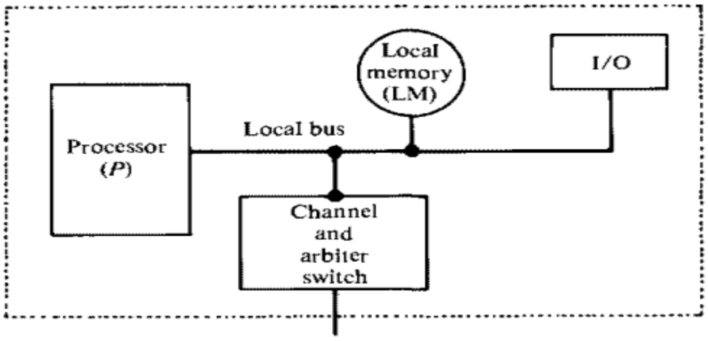
(d) Routing by most significant bit, C_2

Figure 2.15 Three routing functions defined by a binary 3-cube.

6. Network Performance:

- Influenced by ***functionality, network latency, bandwidth, hardware complexity,*** and ***scalability.***
- Considerations of these factors are vital in ***designing and evaluating the effectiveness*** of interconnection networks.

5. LOOSELY COUPLED ARCHITECTURE IN MULTIPROCESSOR SYSTEMS:



(*a*) A computer module

In a loosely coupled multiprocessor system, ***independent modules*** are connected through a ***Message Transfer System (MTS) network.*** This architecture ***enhances fault tolerance and independence*** among processors. **Let's delve into the key components and workings of a loosely coupled system.**

1. **Modules and Components:**
   - Each processor in the system is associated with its own ***local memory, a set of input-output devices, and a Channel and Arbiter Switch (CAS).*** This combination of processor, local memory, I/O devices, and CAS is referred to as a ***computer module.***
   - The ***presence of local memories allows processors to operate independently,*** and ***communication between them is facilitated by the MTS network.***
2. **Communication in Loosely Coupled Systems:**
   - Processes executing on different modules communicate by ***exchanging messages through physical segments of the MTS network.***
   - This type of architecture is often known as a ***distributed system*** because the modules are ***not tightly interconnected*** and may be ***geographically dispersed.***
   - Loosely coupled systems are efficient, ***when processes running on different modules require minimal interaction.*** Each ***module can operate independently most of the time.***
3. **Arbiter Switch (CAS):**
   - The Channel and Arbiter Switch (CAS) plays a crucial role in ***managing access to the MTS.***
   - If multiple modules request access to the MTS simultaneously, the ***CAS must choose one of the requests and delay the others*** until the selected request is serviced completely(responded).

- The CAS has a ***high-speed communication memory*** accessible by all modules in the system. This communication memory serves as a ***buffer*** for message transfers.

4. **Collisions and CAS Handling:**

   - Collisions occur when multiple modules request access to the MTS at the same time. The CAS is responsible for resolving these collisions.
   - The CAS's decision on which request to prioritize can be based on various criteria, such as a ***predefined priority scheme or a first-come-first-served*** basis.
   - Once a request is chosen, the ***CAS ensures that the corresponding message transfer is completed before allowing other pending requests to proceed.***

5. **Efficiency and Independence:**

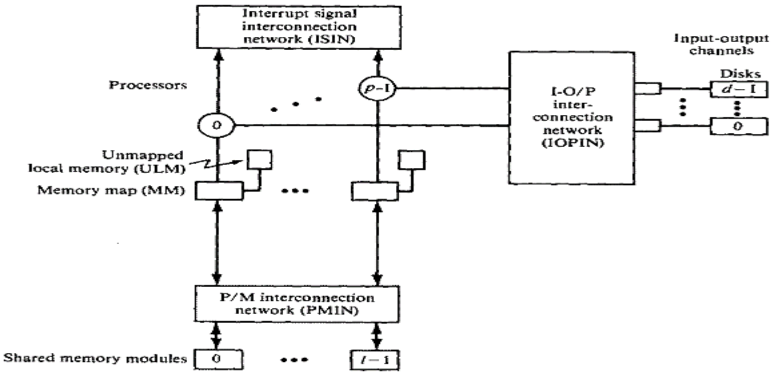
   - Loosely coupled architectures provide a ***high degree of independence between modules,*** allowing them to operate ***autonomously.***
   - Communication between modules is achieved through message passing, and the CAS efficiently manages the access to the shared MTS network.

Here is a list of applications that commonly utilize loosely coupled architecture:

1. Cloud computing platforms
2. ***Microservices*** architectures
3. Distributed systems
4. Event-driven systems

In summary, a ***loosely coupled multiprocessor architecture*** is ***characterized by independent modules connected through an MTS network.*** Communication is achieved through message passing, and the CAS plays a crucial role in managing access to the shared MTS, handling collisions, and ensuring efficient communication between modules.

6. TIGHTLY COUPLED ARCHITECTURE


Interrupt signal interconnection network (ISIN)
Input-output channels
Disks
Processors
p−1
0
I-O/P inter-connection network (IOPIN)
d−1
0
Unmapped local memory (ULM)
Memory map (MM)
P/M interconnection network (PMIN)
Shared memory modules
0
l−1

(a) Without private cache

Tightly coupled multiprocessors architecture refers to a design where the ***components of a system are highly dependent on each other and tightly integrated.*** In this architecture, the ***components share a significant amount of resources*** and rely heavily on ***direct communication and synchronization.***

Components of Tightly Coupled Multiprocessor System without Private Cache:

1. **Processors:** The system comprises ***multiple processors*** that are interconnected and capable of concurrently executing tasks.
2. **Main Memory:** A ***shared main memory*** is accessible to all processors in the system. The processors can read from and write to this common memory.
3. **Interconnection Networks:** The system includes three interconnected networks for facilitating ***communication and coordination*** among the units:
    - **Processor-Memory Interconnection Network**: This network enables processors to access the shared main memory for reading and writing data.
    - **Input-Output Processor Interconnection Network:** It facilitates communication between ***processors and input-output devices*** for ***data transfer and control.***
    - **Interrupt-Signal Interconnection Network:** This network is responsible for ***handling interrupt signals*** and ***managing the interrupt-driven operations*** within the system.

Working of Tightly Coupled Multiprocessor System without Private Cache:

1. **Task Distribution:** When a ***task*** or program is initiated, it can be ***distributed across the multiple processors for parallel execution.*** Each processor can work on a specific portion of the task, leading to ***enhanced performance and reduced execution time.***
2. **Shared Memory Access:** Processors can access the shared main memory to ***read data, write results, and share information.*** This shared memory model allows for ***efficient data sharing and communication*** among the processors.
3. **Inter-Processor Communication:** The interconnected networks enable seamless communication among the processors, facilitating ***data exchange, synchronization, and coordination of tasks.***
4. **Scalability and Performance:** Tightly Coupled Multiprocessor Systems are designed to scale efficiently, allowing for the ***addition of more processors to enhance computational power and overall system performance.***
5. **Task Synchronization:** Mechanisms for synchronization, such as ***mutual exclusion and interprocess interrupts,*** ensure that the ***processors can access shared resources effectively.***

Overall, the tightly coupled multiprocessor system architecture without private cache offers a powerful platform for ***parallel processing, efficient memory access, and inter-processor***

communication, making it suitable for ***high-performance computing*** tasks and ***large-scale data processing*** applications.

7. DIFFERENTIATE BETWEEN LOOSELY AND TIGHTLY COUPLED ARCHITECTURES.

Aspect	Loosely Coupled Architecture	Tightly Coupled Architecture
Components Dependency	***Components are independent*** and have ***minimal dependencies***	Components are ***highly dependent*** on each other
Communication	Communication through ***message passing or indirect methods***	***Direct communication*** through shared memory
Flexibility and Scalability	Highly flexible and scalable as its modules are distributed at different locations.	Less flexible and scalable than the LCMS
System Modifications	***Easier to modify and update*** individual components	***Complex and time-consuming*** modifications to interconnected units
Resource Sharing	Components may ***have their own local resources and memory***	Components ***share a significant amount of resources and memory***
Performance	***Lower performance*** due to ***indirect communication***	***Higher performance*** due to direct communication and resource sharing
Fault tolerance	Better fault tolerant as failures in one component don't affect others significantly	System-wide failures can occur if one component fails
Suitability	Suitable for ***distributed systems and modular architectures***	Suitable for ***high-performance computing and real-time systems***

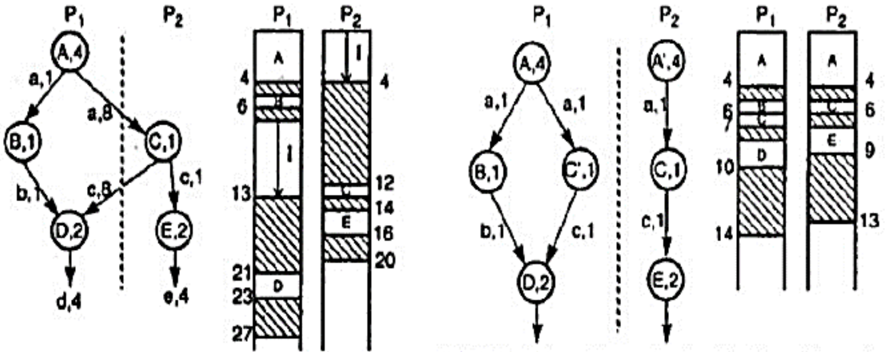
8. WHY SCHEDULING IS REQUIRED FOR PARALLELISM. EXPLAIN STATIC MULTIPROCESSOR SCHEDULING WITH EXAMPLE.

Why Scheduling is Required for Parallelism:

Parallelism involves the simultaneous execution of multiple tasks to improve overall system efficiency. Scheduling is crucial in a parallel computing environment to ***coordinate and optimize the execution of tasks across multiple processors.*** Here's why scheduling is essential:

1. **Resource Utilization:** Scheduling ensures efficient ***utilization of available resources, preventing idle time in processors*** and ***maximizing their potential.***

2. **Load Balancing:** It ***distributes the workload evenly among processors,*** preventing some processors from being underutilized while others are overloaded. This enhances overall system performance.
3. **Minimizing Delays:** Efficient scheduling ***minimizes communication delays and idle time between processors,*** reducing the overall execution time of parallel tasks.
4. **Improved Throughput:** Proper ***scheduling enhances the system's throughput by allowing tasks to progress concurrently*** without unnecessary delays.
5. **Optimizing Parallel Execution:** ***Scheduling algorithms aim to minimize the completion time of parallel tasks,*** optimizing the overall efficiency of the parallel system.



(a) Schedule without node duplication

(b) Schedule with node duplication (A → A and A'; C → C and C')

Figure 2.8 Node-duplication scheduling to eliminate communication delays between processors. (I: idle time; shaded areas: communication delays)

Static Multiprocessor Scheduling - Node Duplication Example:

In static multiprocessor scheduling, one approach to improving performance is ***node duplication.*** This involves ***replicating certain nodes across processors*** to ***eliminate idle time and communication delays.*** Let's break down the example:

1. **Node Duplication Concept:**
   - ***To reduce idle time and communication delays,*** nodes are duplicated and assigned to different processors.
   - Original nodes (A, B, C, D, E) are initially scheduled (Fig. 2.8a), resulting in idle time and delays between processors.
2. **Duplicated Nodes:**
   - Nodes are duplicated to create additional copies (A', C') assigned to different processors.
   - Schedule with duplicated nodes (Fig. 2.8b) is almost 50% shorter than the original, eliminating delays between processors.

Example Outcome:

- By following these steps and introducing node duplication, the parallel schedule becomes more efficient, minimizing delays, and utilizing processor resources effectively.

In summary, scheduling in parallel computing, especially in static multiprocessor scheduling with techniques like node duplication, plays a vital role in ***optimizing resource usage, minimizing delays,*** and ***improving overall system performance.***

9. EXPLAIN GRAIN PACKING AND SCHEDULING WITH SUITABLE EXAMPLE.

Grain packing refers to a technique in parallel computing where smaller computational tasks, known as "grains," are initially defined at a finer level. These fine-grain tasks are then strategically combined or packed together to form larger tasks, called "coarse-grain" tasks. The process involves optimizing the size and grouping of these tasks to enhance parallel execution efficiency.

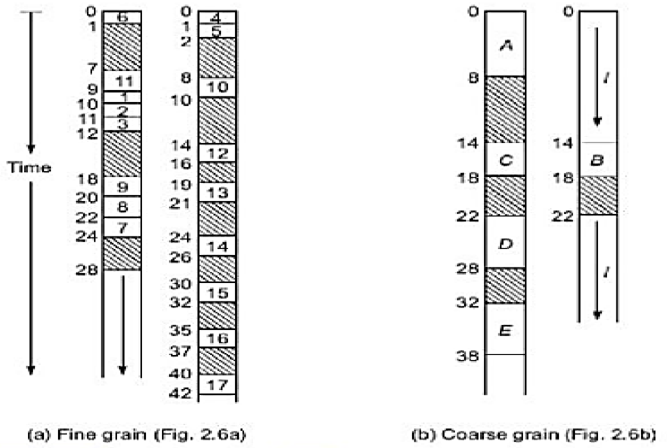
Here's a breakdown:

1. **Fine-Grain Tasks:** These are small, individual computational units or operations within a program. Examples could be simple arithmetic operations or small sections of a program.
2. **Coarse-Grain Tasks:** These are larger tasks formed by combining multiple fine-grain tasks. The goal is to create a balance – large enough to exploit parallelism effectively but not so large that parallelism is underutilized.
3. **Optimizing Parallel Execution:** By combining fine-grain tasks into larger, coherent units (coarse-grain), the scheduling of these tasks for parallel execution can be more efficient. This can lead to better utilization of computational resources and improved overall system performance.
4. **Trade-Off:** There's a trade-off involved in determining the optimal size of the coarse-grain tasks. If they are too fine, the system might spend more time managing parallelism than actually performing computations. On the other hand, if they are too coarse, some processors may remain idle, reducing the benefits of parallel processing.

In summary, grain packing is a strategy used in parallel computing to enhance the efficiency of parallel execution by carefully combining smaller tasks into larger, manageable units, striking a balance between fine-grain and coarse-grain parallelism.

6. Comparison of Schedules:

- Two multiprocessor schedules are compared in Fig 2.7:



(a) Fine grain (Fig. 2.6a) (b) Coarse grain (Fig. 2.6b)

Fig. 2.7 Scheduling of the fine-grain and coarse-grain programs (arrows: idle time; shaded areas: communication delays)

- Fine-Grain Schedule (42-time units): Longer due to more included communication delays (shaded area).
- Coarse-Grain Schedule (38-time units): Shorter because communication delays among nodes 12, 13, and 14 are eliminated after grain packing.

Grain packing and scheduling involve the strategic subdivision of program modules into fine-grain and coarse-grain nodes. The process aims to balance parallelism and scheduling overhead. By applying fine-grain operations first and then combining them into larger coarse-grain nodes, the schedule can be optimized for efficient parallel execution, as illustrated by the example program graphs and schedules. The goal is to minimize communication delays and achieve a shorter overall execution time.