

## HPC – Unit 1 (Introduction to Parallel Computing) – IN-SEM PYQ Answers

Q1. What is Parallel Computing? Describe the scope of parallel computing. [4]

Parallel Computing is a computing paradigm in which **multiple processing elements** execute **multiple tasks simultaneously** to solve a computational problem faster.

In parallel computing:

- A large problem is divided into smaller sub-problems
- Sub-problems are executed concurrently
- Results are combined to produce final output

It aims to improve:

- **Execution time (Speedup)**
- **Throughput**
- **Resource utilization**

**Scope of Parallel Computing:**

1. **High Performance Computing (HPC):**  
Used in supercomputers for scientific simulations such as:
  - Weather forecasting
  - Climate modeling
  - Molecular dynamics
2. **Multi-core and Many-core Systems:**  
Modern processors contain multiple cores enabling thread-level parallelism.
3. **Artificial Intelligence and Machine Learning:**  
Training deep neural networks using GPU-based parallelism.
4. **Big Data and Distributed Systems:**  
Parallel processing of large datasets using clusters and distributed architectures.
5. **Real-time and Embedded Systems:**  
Used in image processing, signal processing, and multimedia applications.
6. **Engineering and Scientific Computation:**  
Matrix operations, numerical simulations, optimization problems.

Q2. What are the application of parallel computing? [4]

1. **Scientific Simulations:** Used for computationally intensive simulations such as climate modeling, astrophysics, and molecular dynamics.
  - Requires large-scale numerical computations
  - Implemented on supercomputers using MIMD architectures
2. **Artificial Intelligence and Machine Learning:** Parallel processing accelerates training of deep neural networks and large datasets
  - GPU-based parallelism (SIMT model)
  - Used in deep learning frameworks
3. **Big Data Analytics:** Enables processing of massive datasets in distributed computing environments.
  - MapReduce and cluster computing
  - Improves throughput and scalability
4. **Image and Signal Processing:** Used in medical imaging, video processing, and speech recognition.

- Data-level parallelism
  - Real-time processing requirements
5. **Engineering Design and CAD/CAM:** Applied in finite element analysis (FEA), structural analysis, and computational fluid dynamics (CFD).
    - Large matrix computations
    - High computational accuracy required
  6. **Cryptography and Security:** Parallel algorithms used for encryption, decryption, and brute-force key search.
    - Reduces computation time
    - Used in cybersecurity applications

Q3. Write a short note on Level of Parallelism. [4]

Levels of Parallelism refer to the different granularities at which parallel execution can occur in a computing system.

Levels of parallelism determine how parallel execution is exploited at hardware and software levels, influencing performance and scalability of parallel systems.

1. **Bit-Level Parallelism:** Achieved by increasing processor word size so that more bits are processed per instruction.
  - Example: 32-bit vs 64-bit processors
  - Improves arithmetic computation efficiency
2. **Instruction-Level Parallelism (ILP):** Multiple instructions are executed simultaneously within a single processor.
  - Achieved using **pipelining, superscalar architecture**
  - Exploits independence between instructions
3. **Data-Level Parallelism (DLP):** Same operation is performed on multiple data elements simultaneously.
  - Used in **SIMD architectures**
  - Common in image processing and vector operations
4. **Task-Level Parallelism (TLP):** Different tasks or threads are executed concurrently.
  - Implemented in **multi-core and multi-processor systems**
  - Suitable for independent program modules

Q4. Write note on Communication costs in parallel machines? [5]

Communication cost refers to the overhead incurred when processors exchange data in a parallel computing system. It significantly affects the performance and scalability of parallel programs.

Communication cost is a major source of overhead in parallel systems and must be minimized through efficient algorithm design and appropriate parallel architecture selection.

1. **Components of Communication Cost:** Communication time is commonly modeled as:
 
$$T_{\text{comm}} = t_s + t_w \times m$$
 where,  
 $t_s$  = Startup time (latency)  
 $t_w$  = Time per word (transfer time per data unit)  
 $m$  = Message size (number of words)
2. **Startup Time (Latency):** Time required to initiate communication between processors.

- Independent of message size
- Includes software overhead and network interface delay
- 3. **Data Transfer Time (Bandwidth Cost):** Time required to transmit data across the network.
  - Proportional to message size
  - Depends on network bandwidth
- 4. **Network Contention and Congestion:** Occurs when multiple processors attempt communication simultaneously.
  - Increases waiting time
  - Reduces overall system performance
- 5. **Synchronization Overhead:** Time spent coordinating processes (e.g., barriers, locks).
  - Idle waiting increases total execution time
  - Impacts parallel efficiency
- 6. **Impact on Performance:**
  - High communication cost reduces **speedup and efficiency**
  - Fine-grained parallelism may suffer due to frequent communication
  - Optimal granularity minimizes communication overhead

Q5. Explain Message Passing Costs in Parallel Computers in parallel machines. [5]

Message passing is a communication mechanism used in distributed memory parallel systems where processors exchange data explicitly through messages. The cost of message passing affects overall execution time.

#### 1. Communication Time Model:

The message passing time is modeled as:

$$T_{msg} = t_s + t_w \times m$$

where,

$t_s$  = Startup time (latency)

$t_w$  = Time per word (inverse of bandwidth)

$m$  = Message size (number of words)

2. **Startup Time (Latency):** Fixed overhead required to initiate communication.
  - Includes software overhead, buffer preparation, network interface delay
  - Independent of message length
3. **Per-word Transfer Time (Bandwidth Cost):** Time required to transmit each word of data.
  - Depends on network bandwidth
  - Proportional to message size
4. **Network Contention:** Occurs when multiple processors communicate simultaneously.
  - Causes delays due to shared communication links
  - Increases effective communication time
5. **Synchronization Overhead:** Processors may need to wait for message arrival.
  - Blocking communication increases idle time
  - Affects parallel efficiency
6. **Impact of Message Size:**
  - Small messages → latency dominates
  - Large messages → bandwidth dominates
  - Optimal performance achieved by balancing message granularity

Q6. Explain SIMD, MIMD and SIMT architectures. [5]

1. **SIMD (Single Instruction Multiple Data):**

A parallel architecture where a single control unit issues the same instruction to multiple processing elements operating on different data.

- All processors execute synchronously
- Suitable for data-level parallelism
- Efficient for vector and matrix operations
- Example: Vector processors

Structure:

Single Instruction Stream → Multiple Data Streams

2. **MIMD (Multiple Instruction Multiple Data):**

A parallel architecture where multiple processors execute different instructions on different data independently.

- Each processor has its own control unit
- Supports task-level parallelism
- Can be shared memory or distributed memory systems
- Example: Multi-core and cluster systems

Structure:

Multiple Instruction Streams → Multiple Data Streams

3. **SIMT (Single Instruction Multiple Threads):**

An execution model used in GPUs where multiple threads execute the same instruction but on different data elements.

- Threads are grouped into warps
- Each thread has its own program counter
- Efficient for massive data parallel applications
- Used in CUDA-based GPU architectures

Structure:

Single Instruction → Multiple Threads (each with separate data)

**Comparison:**

Architecture	Instruction Stream	Data Stream	Typical Use
SIMD	Single	Multiple	Vector processing
MIMD	Multiple	Multiple	General parallel systems
SIMT	Single (thread-based)	Multiple	GPU computing

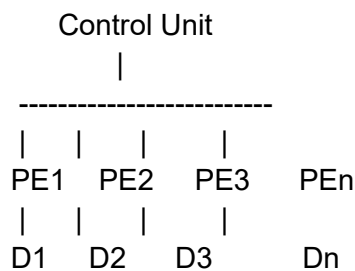
Q7. Explain with suitable diagrams, STMD, MIMD architecture? [4]

1. **SIMD (Single Instruction Multiple Data):**

In SIMD architecture, a **single control unit** broadcasts the same instruction to multiple processing elements. Each processing element operates on different data simultaneously.

- Suitable for **data-level parallelism**
- Processors execute synchronously
- Used in vector and array processors

**Block Diagram:**



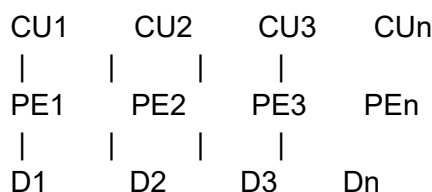
Single Instruction → Multiple Data elements

## 2. MIMD (Multiple Instruction Multiple Data):

In MIMD architecture, each processor has its own control unit and executes different instructions on different data independently.

- Supports **task-level parallelism**
- Can be shared memory or distributed memory
- Used in multi-core systems and clusters

**Block Diagram:**



Multiple Instructions → Multiple Data elements

SIMD is efficient for structured, uniform computations, whereas MIMD is flexible and suitable for general-purpose parallel processing.

Q8. Explain basic working principle of Super scalar Architecture? [6]

A **Superscalar Architecture** is a processor design that can issue and execute **more than one instruction per clock cycle** by exploiting **Instruction-Level Parallelism (ILP)**.

### 1. Basic Principle:

- Multiple functional units are provided (ALU, FPU, Load/Store Unit).
- The processor fetches multiple instructions simultaneously.
- Independent instructions are dispatched to different execution units in parallel.

Goal: Increase **Instructions Per Cycle (IPC)**.

## 2. Main Stages of Working:

a) **Instruction Fetch:** Multiple instructions are fetched from instruction memory per cycle.

b) **Instruction Decode:** Instructions are decoded simultaneously.

c) **Dependency Checking:**

Hardware checks for:

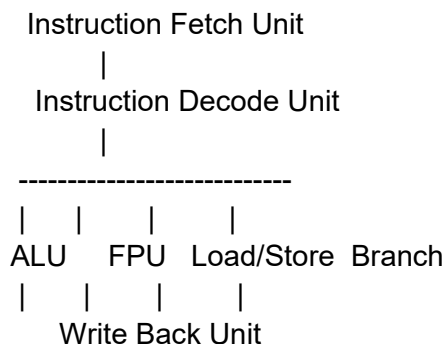
- Data hazards
- Structural hazards
- Control hazards

d) **Instruction Dispatch:** Independent instructions are issued to different execution units.

e) **Parallel Execution:** Functional units execute instructions concurrently.

f) **Write Back / Commit:** Results are written back in correct program order.

## 3. Block Diagram (Conceptual):



## 4. Key Features:

- Multiple execution units
- Dynamic scheduling
- Out-of-order execution (in advanced designs)
- Register renaming (to avoid WAR/WAW hazards)

## 5. Advantages:

- Improves throughput
- Increases processor performance without increasing clock frequency

Superscalar processors are widely used in modern general-purpose microprocessors to exploit instruction-level parallelism efficiently.

### Q9. Explain N-wide Superscalar Architectures. [6]

An **N-wide Superscalar Architecture** is a processor design capable of fetching, decoding, issuing, and executing up to **N instructions per clock cycle**, provided there are no dependencies.

Here, **N** denotes the maximum number of instructions that can be processed simultaneously in one cycle.

#### 1. Basic Concept:

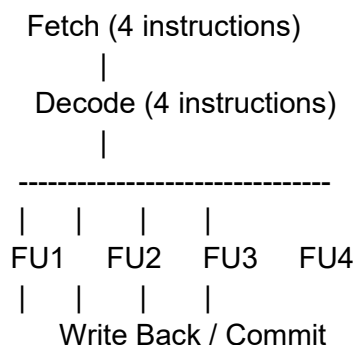
- Extends superscalar principle by increasing instruction issue width to N
- Exploits **Instruction-Level Parallelism (ILP)**
- Multiple functional units operate in parallel

If  $N = 4 \rightarrow$  processor can issue up to 4 instructions per cycle.

#### 2. Working Principle:

- Multiple Instruction Fetch:** Fetches N instructions in a single clock cycle.
- Parallel Decode:** All N instructions are decoded simultaneously.
- Dependency Checking:** Hardware checks for:
  - RAW (Read After Write)
  - WAR (Write After Read)
  - WAW (Write After Write) hazards
- Instruction Issue:** Independent instructions are dispatched to available execution units.
- Parallel Execution:** N functional units execute instructions concurrently.
- Commit Stage:** Results are written back in program order.

#### 3. Conceptual Block Diagram (4-wide example):



#### 4. Key Features:

- Multiple pipelines
- Dynamic scheduling
- Register renaming
- Out-of-order execution (advanced designs)

5. **Performance Consideration:** Maximum theoretical speedup  $\approx N$

Actual performance depends on:

- Availability of independent instructions
- Branch prediction accuracy
- Hardware complexity

N-wide superscalar architectures are used in modern high-performance processors to increase throughput without increasing clock frequency.

Q10. What are types of dataflow execution models? [6]

In the **Dataflow Execution Model**, program execution is driven by the availability of data rather than by sequential program counters. An instruction executes only when all its input operands are available.

The main types of Dataflow Execution Models are:

**1. Static Dataflow Model**

- Each instruction appears only once in the program graph.
- Data tokens travel along fixed arcs between nodes.
- No support for multiple simultaneous activations of the same instruction.

**Characteristics:**

- Simple implementation
- Limited support for loops and recursion
- Requires code replication for iterative constructs

**2. Dynamic Dataflow Model**

- Allows multiple instances of the same instruction to execute simultaneously.
- Uses **tags** to distinguish different activations.
- Supports loops and recursive procedures efficiently.

**Characteristics:**

- More flexible than static model
- Suitable for fine-grained parallelism
- Higher hardware complexity

**3. Demand-Driven (Lazy Evaluation) Model**

- Execution is initiated only when the result is required.
- Computation proceeds backward from output requirements.
- Often used in functional programming languages.

**Characteristics:**

- Avoids unnecessary computations
- Improves efficiency in some applications
- Used in dataflow-inspired systems

**4. Hybrid Dataflow Model**



- Combines features of control-driven (Von Neumann) and data-driven execution.
- Control flow manages coarse structure; dataflow enables parallelism within blocks.

#### Characteristics:

- Practical implementation in modern processors
- Supports parallel execution with manageable complexity

These dataflow models enable exploitation of parallelism by allowing instructions to execute as soon as their data dependencies are satisfied.

Q11. Explain Cache coherence in multiprocessor system. [5]

In a **shared memory multiprocessor system**, each processor has a private cache.

**Cache coherence** ensures that all processors have a **consistent view of shared memory** when copies of the same data exist in multiple caches.

#### 1. Need for Cache Coherence:

- Multiple processors may cache the same memory location.
- If one processor modifies data, others may read stale data.
- Coherence mechanism maintains data consistency.

#### 2. Cache Coherence Problem:

Example:

- P1 and P2 cache variable X
- P1 updates X
- P2 still holds old value → inconsistency

#### 3. Requirements of Cache Coherence:

##### a) Write Propagation:

A write to a memory location must be visible to all processors.

##### b) Write Serialization:

All processors must see writes to the same location in the same order.

#### 4. Cache Coherence Protocols:

##### a) Snooping Protocol:

- Used in bus-based shared memory systems
- Caches monitor (snoop) the shared bus
- Examples: MSI, MESI protocols

##### b) Directory-Based Protocol:

- Used in large-scale multiprocessors
- A directory keeps track of which caches hold copies
- Reduces bus traffic

**5. Impact:**

- Ensures correctness of parallel programs
- Involves communication overhead
- Essential for scalable shared-memory architectures

Q12. Explain the impact of Memory Latency & Memory Bandwidth on system performance? [6]

In parallel and high-performance systems, memory performance significantly affects overall execution time. Two critical parameters are **Memory Latency** and **Memory Bandwidth**.

**1. Memory Latency**

Memory latency is the time delay between issuing a memory request and receiving the data.

- Includes cache miss time, memory access time, and network delay (in distributed systems).
- Measured in nanoseconds or clock cycles.

**Impact on Performance:**

- High latency increases processor stall cycles.
- Reduces CPU utilization.
- Affects performance of fine-grained parallel programs.
- Dominates when memory accesses are frequent and irregular.

Latency effect is critical in pointer-based and random-access workloads.

**2. Memory Bandwidth**

Memory bandwidth is the rate at which data can be transferred between memory and processor.

Bandwidth = Data transferred / Time

- Measured in bytes/sec or GB/s.
- Depends on bus width, memory frequency, and architecture.

**Impact on Performance:**

- Low bandwidth limits throughput.
- Becomes bottleneck in data-intensive applications (e.g., matrix operations).
- Affects large-scale parallel and GPU computations.

Bandwidth dominates in streaming and bulk data transfer workloads.

**3. Latency vs Bandwidth Comparison**

Parameter	Memory Latency	Memory Bandwidth
Definition	Delay to access data	Rate of data transfer
Affects	Response time	Throughput

<b>Critical for</b>	Random access workloads	Large data processing
<b>Performance Limitation</b>	Processor stalls	Data transfer bottleneck

#### 4. Overall Impact on Parallel Systems

- High latency reduces speedup and efficiency.
- Insufficient bandwidth causes communication bottlenecks.
- Modern architectures use caching, prefetching, pipelining, and multithreading to hide latency and improve bandwidth utilization.

Q13. Describe UMA and NUMA multicomputer platforms? [5]

Q14. Describe Uniform-memory-access and Non-uniform-memory-access with diagrammatic representation. [6]

In shared-memory multiprocessor systems, memory organization is classified as **Uniform Memory Access (UMA)** and **Non-Uniform Memory Access (NUMA)** based on memory access time.

##### 1. UMA (Uniform Memory Access):

In UMA architecture, all processors share a common physical memory and the access time to any memory location is the same for all processors.

##### Characteristics:

- Single shared memory
- Equal memory access latency
- Typically bus-based or crossbar interconnect
- Easier to implement cache coherence

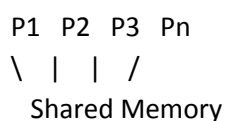
##### Advantages:

- Simpler programming model
- Uniform performance characteristics

##### Limitations:

- Scalability limited due to bus contention
- Suitable for small to medium-scale systems

##### Block Diagram:



##### 2. NUMA (Non-Uniform Memory Access):

In NUMA architecture, each processor has local memory, but the entire memory space is shared logically. Access time depends on whether memory is local or remote.

**Characteristics:**

- Distributed physical memory
- Faster local memory access
- Slower remote memory access
- Requires directory-based cache coherence

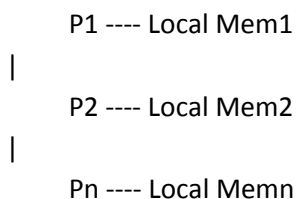
**Advantages:**

- Better scalability
- Suitable for large multiprocessor systems

**Limitations:**

- Complex memory management
- Performance depends on data locality

**Block Diagram:**



**Comparison:**

Feature	UMA	NUMA
Memory Access Time	Uniform	Non-uniform
Scalability	Limited	High
Memory Organization	Centralized	Distributed
Coherence Method	Snooping	Directory-based

UMA provides uniform access time, while NUMA improves scalability by distributing memory and optimizing local memory access.

Q15. What is VLIW processor? Write any two advantages of VLIW? [4]

**VLIW (Very Long Instruction Word) Processor:**

A **VLIW processor** is an architecture that executes multiple operations simultaneously using a single long instruction word.

- A single instruction contains multiple independent operations.
- The compiler performs instruction scheduling.
- Hardware executes all operations in parallel without complex dynamic scheduling.

Structure:

```
| OP1 | OP2 | OP3 | OP4 |
|   |   |   |   |
FU1  FU2  FU3  FU4
```

Each field controls a separate functional unit.

### Advantages of VLIW:

#### a) **Simpler Hardware Design:**

- No need for complex dynamic scheduling hardware.
- Reduces control logic complexity.

#### b) **High Instruction-Level Parallelism (ILP):**

- Multiple operations executed in one cycle.
- Improves throughput when sufficient independent instructions exist.

Q16. Write a short note on: 1) Dataflow Models, 2) Demand Driven Computation, 3) Cache Memory. [5]

### 1) Dataflow Models [3]

A **Dataflow Model** is an execution model in which an instruction executes only when all its input operands are available. Execution is driven by data availability rather than program counter control.

#### **Key Features:**

- Program represented as a **dataflow graph** (nodes = operations, edges = data dependencies)
- Enables fine-grained parallelism
- Eliminates global program counter

#### **Types:**

- Static Dataflow
- Dynamic Dataflow

Used in parallel architectures to exploit inherent parallelism.

### 2) Demand Driven Computation [3]

Demand-driven computation is a dataflow-based model where execution begins only when the output value is required.

- Also called **lazy evaluation**
- Computation proceeds backward from required result
- Avoids unnecessary calculations

#### **Characteristics:**

- Efficient for functional programming

- Reduces redundant computations
- Improves resource utilization in some applications

### 3) Cache Memory [3]

**Cache Memory** is a small, high-speed memory placed between CPU and main memory to reduce average memory access time.

- Stores frequently accessed data and instructions
- Exploits **temporal and spatial locality**

**Levels:**

- L1 (fastest, smallest)
- L2
- L3

**Advantages:**

- Reduces memory latency
- Improves processor performance
- Minimizes processor stall cycles