USN										

Internal Assessment Test 1 – Sept. 2025

	Subject/Code						
	Date:	Duration: 90 mins	Max. Marks: 50	Semester: 07		Branch:	

Sl.	Answer any FIVE FULL Questions	Marks	CO	RBT
1	Describe the different types of interconnection networks (bus, crossbar, mesh, torus). Compare their performance. **Answer:** Interconnection networks connect processors and memory modules in a parallel computer system. Their performance is critical for efficient communication. Types: 1. Bus: o Description: A single communication path (a set of wires) shared by all processors and memory modules. Only one device can transmit data at a time, requiring an arbitration mechanism. o Performance: Low cost and simple to implement. However, it has high contention (traffic conflict) and limited bandwidth. Performance degrades significantly as the number of processors increases. 2. Crossbar: o Description: A non-blocking network with a grid of switches. It provides a dedicated path for every possible pair of processors and memory modules. o Performance: Offers the highest bandwidth and low latency, as multiple communications can happen simultaneously without interference. The major drawback is its high cost and complexity, as the number of switches required is O(N2) for N processors. 3. Mesh: o Description: Processors are arranged in a k-dimensional grid (commonly 2D). Each processor connects to its north, south, east, and west neighbours. Processors on the edges have fewer connections. o Performance: More scalable than a bus or crossbar. Cost is O(N)for N processors. Latency depends on the distance (number of hops) between communicating nodes. A significant drawback is that the diameter (longest shortest path) is large for non-adjacent nodes. 4. Torus: o Description: A variant of the mesh where the nodes on the edges are connected to the nodes on the opposite edges, forming a ring in each	10	CO1	L1

	dimension. This eliminates the edge nodes of a simple mesh. o Performance: Has a smaller diameter and better average latency compared to a mesh of the same size due to its wraparound connections. The cost and complexity are slightly higher than a mesh but offer better performance and load balancing. Comparison of Performance: **Network** | **Scalability** | **Latency (Avg.)** | **Bandwidth** | **Contention** | **Cost & Complexity** **Bus** | Very Poor | Low (if idle) | Very Low | Very High | Very Low **Crossbar** | Good | Very Low | Very High | Very Low | Very High (N2*N*2) **Mesh** | Good | Medium-High | Medium | Medium | Medium (*NN*) **Torus** | Good | Medium | Medium | Low | Medium-High (*NN*)			
2	a) Explain data parallelism? Give one example. **Answer:** Data parallelism is a parallel computing model where the same operation is performed concurrently (simultaneously) on different elements of a large dataset. The key idea is to distribute the data across multiple processing units, and each unit executes the same function on its portion of the data. • Example: Adding two large arrays, A and B, to produce a result array C. If you have 4 processor cores, you can split the arrays into 4 chunks. Each core independently computes C[i] = A[i] + B[i] for its assigned chunk of indices i. b) Mention any two drawbacks of distributed-memory programming. **Answer:** 1. Explicit Communication Overhead: The programmer is responsible for explicitly sending and receiving messages between processes. This adds complexity to the code and introduces communication latency, which can become a performance bottleneck. 2. Data Partitioning and Load Balancing: It can be challenging to divide the data and	5+5	CO1	L2

	workload evenly among all processors. Poor load balancing leads to some processes finishing early and sitting idle while others are still working, reducing overall efficiency.			
3	A serial program runs for 480 seconds. Its parallel MPI version runs for 100 seconds using 5 processes. a) Calculate the speedup and efficiency. **Answer:** • **Speedup (S):** $S = \frac{T_{serial}}{T_{parallel}} = \frac{480}{100} = 4.8$ • **Efficiency (E):** $E = \frac{S}{N} = \frac{4.8}{5} = 0.96$ or **96%** (Where $N = 5$ is the number of processes) b) The parallel time can be broken down: T_parallel = T_comp + T_comm + T_idle. If the total computation time (sum across all processes) is 450 seconds, and the total communication time is 50 seconds, calculate the average utilization (computation time/parallel time) for a process. • Total parallel time for the application, $T_{parallel} = 100$ seconds. • Total computation time *across all processes* = 450 seconds. • Since there are 5 processes, the **average computation time per process** is $\frac{450}{5} = 90$ seconds. • **Average Utilization** for a process is the fraction of time it spends computing. Utilization $= \frac{\text{Average Computation Time per Process}}{T_{parallel}} = \frac{90}{100} = 0.90$ or **90%**	5+5	CO2	L3

4	Write pseudocode for shared-memory vector addition using OpenMP parallel for.
	
	Answer:
	
	The problem is to compute C[i] = A[i] + B[i] for all i in parallel using OpenMP.
	Pseudocode:
	
	Program VectorAddition
	Declare arrays A[0..N-1], B[0..N-1], C[0..N-1]
	Initialize arrays A and B with values
	
	#pragma omp parallel for
	for i = 0 to N-1 do
	C[i] = A[i] + B[i]
	end for
	
	(Optional: Output or use the result array C)
		10	CO2	L1

	End Program Explanation: • The #pragma omp parallel for directive tells the compiler to split the following for loop iterations among multiple threads automatically. • Each thread gets a different chunk of the indices i and performs the addition C[i] = A[i] + B[i] on its chunk. • OpenMP handles the creation, management, and synchronization of threads, making the code simple and concise.			

5	a)
	Discuss point-to-point communication in MPI with suitable examples of send/receive operations.
	
	Answer:
	
	Point-to-point communication involves a direct, one-to-one data transfer between two
	specific MPI processes: a sender and a receiver. It is the fundamental building block for
	message passing.
	• Key Functions:
	o MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag,
	MPI_Comm comm): Sends count elements of datatype from buffer buf to
	process dest.
	o MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
	MPI_Comm comm, MPI_Status *status): Receives a message into buffer buf from process source.
	
	• Example:
	Process 0 sends an integer to Process 1.
	c
	// In Process 0
	int data_to_send = 123;
	MPI_Send(&data_to_send, 1, MPI_INT, 1, 0,
	MPI_COMM_WORLD);
	
	// In Process 1
	int received_data;
	
	MPI_Recv(&received_data, 1, MPI_INT, 0, 0,
	MPI_COMM_WORLD,
	MPI_STATUS_IGNORE);
	// Now received_data = 123
	
	b)
	Describe in detail collective communication operations such as Scatter, Gather, and All_reduce.
	
	Answer:
		5+5	CO3	L2

	Collective communication involves a group of processes (all processes in a communicator) participating in a coordinated communication operation. 1. MPI_Scatter: o Description: One process (the root) divides its send buffer into equal chunks and sends a distinct chunk to every process in the communicator, including itself. o Use Case: Distributing parts of an input array from a root process to all other processes. 2. MPI_Gather: o Description: The inverse of Scatter. The root process collects distinct chunks of data from every process in the communicator and assembles them into a single receive buffer in rank order. o Use Case: Collecting results from all worker processes onto a single master process. 3. MPI_Allreduce: o Description: Combines data from all processes using a reduction operation (like sum, max, min) and distributes the result back to all processes. o Use Case: Calculating a global sum, a global maximum, or any other associative/commutative operation where every process needs the final result. For example, calculating the dot product in a parallel linear algebra routine.			
6	a) Discuss GPU architecture as an extension of SIMD Parallelism. **<u>Answer:</u>** GPU architecture is a highly parallel architecture that can be viewed as a powerful extension of the traditional SIMD (Single Instruction, Multiple Data) model. • SIMD: A single control unit (CU) fetches an instruction, and multiple processing elements (PEs) execute that same instruction simultaneously on different data elements. • GPU Extension: A GPU consists of many Streaming Multiprocessors (SMs). Each SM contains many cores (e.g., CUDA Cores in NVIDIA GPUs). These cores are organized in groups (warps in NVIDIA, wavefronts in AMD). o SIMT (Single Instruction, Multiple Threads): This is the GPU's execution	5+5	CO1	L2

model. A single instruction is executed concurrently by a large number of
threads (often thousands). These threads are grouped into warps. All threads in
a warp execute the same instruction in lockstep on different data, which is the
essence of SIMD.
o Massive Parallelism: While SIMD typically handles a few to dozens of data
elements, a GPU can launch thousands of threads, making it massively
parallel. This makes GPUs exceptionally well-suited for data-parallel tasks
like graphics rendering, matrix operations, and machine learning.

b) Compare data-parallel programming and task-parallel programming with suitable use cases

Answer:

Feature	Data-Parallel Programming	Task-Parallel Programming
Core Idea	Same operation applied to different data items.	Different operations (tasks) executed concurrently.
Decomposition	Problem is decomposed by data.	Problem is decomposed by functions/tasks.
Synchronization	Often requires synchronization (e.g., barriers).	Requires coordination and communication between tasks.
Suitability	Regular problems with large, homogeneous datasets.	Irregular problems, pipelines, or heterogeneous tasks.
Use Cases	Vector addition, image filtering, matrix multiplication.	Web server (handling different requests), compiling multiple files in a large project, a complex simulation with different physics models.