

BCS702 – Parallel Computing

Important Question Bank

Module 1: Parallel Hardware and Shared Memory

1. Explain the architectural differences between **SIMD** and **MIMD** systems with relevant examples.
2. Discuss the internal components of a **SIMD vector processor** and how it handles data.
3. Describe the different types of **interconnects** used in shared-memory systems (like Bus, Crossbar, and Multistage).
4. What is **Cache Coherence**? Explain the problem and provide a common solution used in multiprocessor systems.
5. Explain the concept of **False Sharing** and how it negatively impacts the performance of a parallel program.
6. Discuss the advantages and drawbacks of using **one-sided communication** in distributed systems.
7. What is **non-determinism** in shared memory systems, and why is it a challenge for programmers?
8. Compare **Shared Memory** vs. **Distributed Memory** architectures in terms of scalability and ease of programming.

Module 2: Parallel Program Design and Performance

9. Explain **Amdahl's Law** and how it is used to calculate the maximum potential speedup of a parallel system.
10. Define the following performance metrics: **Total Execution Time (T_{serial}/T_{parallel}), Speedup, and Efficiency**.
11. Discuss the concept of **Scalability** and how it changes when you increase the number of processors versus the problem size.
12. Describe the standard methods used to accurately measure the **execution time** of a parallel program.
13. How is **Input and Output (I/O)** typically handled in an MIMD system to avoid bottlenecks?
14. Explain the difference between **Strong Scaling** and **Weak Scaling** in parallel computing.

Module 3: Programming with MPI

15. What is the **Message Passing Interface (MPI)**? Explain its basic execution model and environment.
16. Describe the functionality of essential MPI functions: `MPI_Init`, `MPI_Finalize`, `MPI_Comm_size`, and `MPI_Comm_rank`.
17. Compare **Point-to-Point communication** (Send/Receive) with **Collective communication** (Broadcast/Gather).
18. Explain the syntax and working of `MPI_Reduce` and `MPI_Allreduce` with a simple diagram.
19. Describe how the **Trapezoidal Rule** can be parallelized using MPI; provide the general logic or pseudo-code.
20. Explain the difference between `MPI_Scatter` and `MPI_Gather` and how they manage data distribution.

Module 4: Shared Memory Programming with OpenMP

21. Explain the **Fork-Join model** used in OpenMP for creating and managing threads.
22. Describe the use of the `#pragma omp parallel` directive and explain the `private`, `shared`, and `default` clauses.
23. What is a **Loop-Carried Dependency**? Explain why it prevents parallelization and how the `reduction` clause solves it.
24. Compare the different **Scheduling types** in OpenMP: Static, Dynamic, and Guided.
25. Explain the difference between `#pragma omp critical` and `#pragma omp atomic` for protecting shared data.
26. Describe the `#pragma omp task` directive and explain how it is used to parallelize recursive algorithms.

Module 5: GPU Programming with CUDA

27. Describe the **CUDA Hardware Architecture**, specifically focusing on the hierarchy of Grids, Blocks, and Threads.
28. Explain the roles of the **Host (CPU)** and the **Device (GPU)** in a typical CUDA application workflow.
29. Describe the different levels of **Memory in CUDA**: Global, Shared, Constant, and Local memory.
30. What is a **Warp** in CUDA? Explain the concept of **Warp Divergence** and its effect on GPU performance.
31. Explain the syntax used to launch a CUDA kernel: `KernelName<<<blocks, threads>>>(args)`.
32. Define **GPGPU** (General-Purpose computing on Graphics Processing Units) and discuss why GPUs are faster than CPUs for certain tasks.