HIGH PERFORMANCE COMPUTING:

COURSE OBJECTIVE:	• Provide a thorough grounding in the core principles and concepts of High-Performance Computing (HPC). • Develop proficiency in the programming languages and frameworks essential for implementing and managing parallel algorithms and systems. • Cultivate the ability to analyze and optimize the performance of HPC applications, enhancing efficiency and productivity in computational tasks. • Offer hands-on exercises and project-based learning to equip participants with practical experience in HPC environments. • Empower participants to use advanced computing technologies to solve complex computational problems efficiently, fostering innovative and effective solutions.
COURSE OUTCOME:	• Exhibit the foundation skills of the core principles and concepts of High-Performance Computing (HPC). • Develop proficiency in programming languages and frameworks essential for parallel computing. • Optimize the performance of HPC applications to improve efficiency and productivity. • Implement practical skills from hands-on exercises and project-based learning in HPC environments. • Utilize advanced computing technologies to solve complex computational problems efficiently, creating innovative and effective solutions.

Course Duration: 45 Hours

Course Content:

Unit 1: Introduction to HPC & Parallel Computing Overview

Introduction to HPC and supercomputing - Evolution of supercomputers - Hardware and software components of an HPC system - Concept of cluster - HPC applications - Need for parallel computing – Parallel architectures - Memory architecture Classification - Levels of parallelism -Parallel Programming Techniques

Unit 2: Parallel Programming with OpenMP (Basic & Advanced)

OpenMP theory- Shared programming memory model - OpenMP directives, runtime library, environment variables - Basic OpenMP Constructs: Parallel Regions, Work- sharing Constructs - Synchronization and Data Scope in OpenMP

- Setting up of OpenMP environment, compilation and execution of programs - Critical section, atomic and reduction, variable scoping - schedule clause (static, dynamic) OpenMP tasks, examples – Debugging

Unit 3: Parallel programming with MPI (Basic & Advanced)

Introduction to Message Passing Interface (MPI)- Comparison of MPI with - OpenMP-MPI Basics: Point-to- Point Communication-Send/receive, blocking/non-blocking-Setting up an MPI Environment-Writing and Running a Simple MPI Program-Collective communication in MPI-Overview of Collective Operations-Broadcast, Scatter, Gather, Reduce, and All-to-All - MPI Groups and Communicators-Derived data types-MPI with thread MPI application

Unit 4: GPGPU programming GPGPU architecture:

Introduction to GPGPU - Comparison with CPU-GPGPU Architecture-Memory structure - Different GPGPU architectures (NVIDIA, AMD etc.) Introduction to GPGPU Programming with OpenACC Topics: Different Programming models for GPGPU - Introduction to OpenACC - Execution model - Levels of parallelism - Setting up an OpenACC Environment - Writing and Running a Simple OpenACC Program

Unit 5 – GPGPU programming with CUDA and HPC Tools

Introduction to CUDA Programming Model-CUDA toolkit-CUDA Kernels and Threads-CUDA Memory Model: Global, Shared, Constant, and Local Memory-Thread Organization: Blocks and Grids-Memory Allocation and Deallocation in CUDA-Memory Copy Operations-Overview of CUDA libraries-Writing and executing sample CUDA Programs – Benchmarking - Profiling and debugging - Performance measurement and optimization - Job scheduling and resource management - Containers and virtualization - Hybrid programming (OpenMP + MPI) - Work on previous assignments

Test Projects:

Use Cases:

1 Benchmarking of molecular modelling application (NAMD) on HPC cluster
2 Solve the Jacobi iteration of linear systems of problem size of matrix 4096x

4096, in parallel by OpenMP

3 Implement parallel reduction to compute the sum, maximum, or minimum of an array using OpenMP. Use reduction clauses to synchronize and combine partial results from multiple threads efficiently.

4 Develop a parallel image filtering application using OpenMP. Implement filters such as Gaussian blur or edge detection with parallelization of pixel processing using OpenMP parallel loops. Utilize synchronization mechanisms to handle boundary conditions and avoid data races.

5 Implement a parallel Monte Carlo simulation using OpenMP that requires synchronization. For example, simulate concurrent access to a shared resource (e.g., bank account) by multiple threads and ensure correctness using OpenMP locks or atomic operations.

6 Develop a parallel tree search algorithm using tasking in OpenMP. For example, implement parallel depth-first or breadth-first search (DFS/BFS) algorithms for searching trees or graphs. Each node exploration can be represented as a task, allowing for efficient parallelization of tree search algorithms.

7 Implement multithreaded networking components for multiplayer games using OpenMP to handle concurrent network connections and message processing. Parallelize network communication tasks to improve the scalability and responsiveness of multiplayer game servers, supporting large numbers of concurrent players.

8 Implement mutual exclusion using OpenMP locks (omp_lock_t) to protect critical sections of code that should only be executed by one thread at a time. Use lock acquisition and release operations (omp_set_lock, omp_unset_lock) to enforce exclusive access to shared resources, preventing concurrent access and data corruption

9 Parallelize a sorting algorithm using MPI.

a. Choose a simple sorting algorithm (e.g., quicksort or mergesort) and parallelize it using MPI. Each process should sort its portion of the data, and then the sorted segments should be merged. Experiment with different data sizes and evaluate the parallel sorting algorithm's performance

10. Compare the performance of blocking and non-blocking collective communication calls for matrix multiplication of size 1024 X 1024

11. Calculate bandwidth using round-robin/circular shift method of point-to-point MPI communication calls

12. Implementation of dynamic process management by establishing communication between 2 MPI applications

13. Develop a parallel matrix multiplication algorithm using MPI and OpenMP optimized for large scale scientific computation

14. Conduct a performance analysis of the above application (10K x 10K) using gprof and identify potential optimization opportunities.

15. Write a program that performs the following vector operations:
    1. Vector addition
    2. Vector subtraction
    3. Element-wise multiplication
    4. Element-wise division
    5. Parallelize each operation using OpenACC directives. Experiment with different vector sizes and compare the parallelized version's performance against the sequential implementation.

16. Write a program that computes the element-wise maximum of two arrays. Parallelize the computation using OpenACC directives. Test the program with arrays of varying sizes and evaluate the performance improvement.

17. Implement a Monte Carlo simulation for estimating π by generating random points within a unit square and calculating the ratio of points inside a quarter circle to the total points. Parallelize the simulation using OpenACC directives. Experiment with different numbers of random points and observe the performance improvements

18. Implement a 2D heat diffusion simulation using a grid of temperature values. Parallelize the simulation using OpenACC directives to speed up the computation.

    Experiment with different grid sizes and observe the impact on parallel performance

19. Implement a CUDA program for matrix multiplication. Divide the matrix multiplication task among threads and blocks efficiently. Experiment with different matrix sizes and analyse the performance improvement achieved with CUDA parallelization
20. Write a program in MPI and CUDA to sort 1 million random integers. The task should be distributed across minimum two processes and GPUs.