

<b>Devi Ahilya University, Indore, India Institute of Engineering &amp; Technology</b>				<b>IV Year B.E. (Information Technology)</b>			
<b>Subject Code &amp; Name 6ITRE5</b>	<b>Instructions Hours per Week</b>			<b>Credits</b>			
<b>High Performance Computing</b>	<b>L</b>	<b>T</b>	<b>P</b>	<b>L</b>	<b>T</b>	<b>P</b>	<b>Total</b>
<b>Duration of Theory Paper: 3 Hours</b>	<b>3</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>1</b>	<b>1</b>	<b>5</b>

**Learning Objective :**

- Understand the basics of parallel computing architectures, including Flynn's Taxonomy and shared and distributed memory models.
- Describe and apply performance metrics, including speed up and scalability, to measure the effectiveness of parallel algorithms.
- Analyze communication overheads and latency and their impact on parallel computing performance.
- Demonstrate the use of OpenMP for parallelization of matrix algebra algorithms and understand its syntax.
- Use MPI for parallelization and communication, including domain decomposition and load balancing.

**UNIT-I**

Introduction to high performance computing , Architectures for parallel computing- Flynn;s Taxonomy, Shared and distributed memory , Examples of parallel algorithms.

**UNIT-II**

Performance metrics- speed up, scalability Communication overheads and latency , Introduction to OpenMP.

**UNIT-III**

OpenMP parallelization of Matrix algebra algorithms ,Demonstration of OpenMP codes, Introduction to MPI , Communications using MPI Domain decomposition and Schwarz algorithm, Load balancing

**UNIT-IV**

Jacobi solver – serial and parallel implementation , Code demonstration and performance evaluation , Architectures of GPU, GPU Memories , Introduction to CUDA.

**UNIT-V**

Thread algebra for matrix calculations , Examples of CUDA kernels , Matrix algebra using CUDA , Performance optimization , Code demonstration

**Learning outcome:**

- 1) Students with a strong foundation in parallel computing and high-performance computing. They will be able to apply these concepts and techniques to real-world problems and develop optimized and scalable solutions.

**List of Practical Assignments:**

- 1) Set up a high-performance computing environment: It is essential to have a proper high-performance computing environment to practice and learn parallel computing. Students can install and configure HPC environments like OpenMPI, OpenMP, and CUDA.

- 2) Implement parallel algorithms: Students can start with simple parallel algorithms like matrix multiplication, sorting, and searching.
- 3) Measure performance metrics: Students should learn how to measure the performance of parallel algorithms using metrics like speedup and scalability. They should also learn how to measure communication overheads and latency.
- 4) Implement OpenMP and MPI codes: OpenMP and MPI are two popular parallel programming models. Students can practice implementing codes using these models and observe the performance differences between them.
- 5) Implement Jacobi solver: The Jacobi solver is a popular iterative method used to solve linear equations. Students can implement serial and parallel versions of the Jacobi solver and compare their performance.
- 6) Implement matrix algebra using CUDA: Students should learn how to implement matrix algebra using CUDA. They should also learn how to optimize performance by using shared memory and other techniques.
- 7) Parallelize existing code: Students can practice parallelizing existing codes using OpenMP, MPI, or CUDA. They can analyze the performance of the original code and compare it with the parallelized version.
- 8) Experiment with load balancing: Load balancing is an important technique used to distribute the computational workload across multiple processors. Students can experiment with load balancing algorithms and observe their impact on performance.
- 9) Evaluate and optimize performance: After implementing and parallelizing codes, students should evaluate the performance of their codes and optimize them for better performance. They should also learn how to analyze performance bottlenecks and optimize code accordingly.

### **Reference Books**

- [1]. "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein.
- [2]. "High-Performance Computing: Programming and Applications" by John Levesque, Robert W. Numrich, and Anne M. Trefethen.
- [3]. "Parallel Computing for Science and Engineering" by George Em Karniadakis and Robert M. Kirby II.
- [4]. "Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers" by Barry Wilkinson and Michael Allen.
- [5]. "Programming Massively Parallel Processors: A Hands-on Approach" by David B. Kirk and Wen-mei W. Hwu.
- [6]. "CUDA by Example: An Introduction to General-Purpose GPU Programming" by Jason Sanders and Edward Kandrot.