**Session 2025-2026**

| **Vision:** Dream of where you want. | **Mission:** Means to achieve Vision |
| --- | --- |

**Program Educational Objectives of the program (PEO):** (broad statements that describe the professional and career accomplishments)

| PEO1 | **Preparation** | **P: Preparation** | **Pep-CL abbreviation**  **pronounce as Pep-si-lL easy to recall** |
| --- | --- | --- | --- |
| PEO2 | **Core Competence** | **E: Environment (Learning Environment)** |
| PEO3 | **Breadth** | **P: Professionalism** |
| PEO4 | **Professionalism** | **C: Core Competence** |
| PEO5 | **Learning Environment** | **L: Breadth (Learning in diverse areas)** |

**Program Outcomes (PO):** (statements that describe what a student should be able to do and know by the end of a program)

**Keywords of POs:**

Engineering knowledge, Problem analysis, Design/development of solutions, Conduct Investigations of Complex Problems, Engineering Tool Usage, The Engineer and The World, Ethics, Individual and Collaborative Team work, Communication, Project Management and Finance, Life-Long Learning

**PSO Keywords:** Cutting edge technologies, Research

“I am an engineer, and I know how to apply engineering knowledge to investigate, analyse and design solutions to complex problems using tools for entire world following all ethics in a collaborative way with proper management skills throughout my life.” *to contribute to the development of cutting-edge technologies and Research*.

**Integrity:** I will adhere to the Laboratory Code of Conduct and ethics in its entirety.

**Name and Signature of Student and Date**

(Signature and Date in Handwritten)

| **Session** | **2025-26 (ODD)** | | **Course Name** | **HPC Lab** | |
| --- | --- | --- | --- | --- | --- |
| **Semester** | **7** | | **Course Code** | **22ADS706** | |
| **Roll No** | **35** | | **Name of Student** | **Aman Makode** | |
|  |  | |  |  |  |
| Practical Number | | **Practical 3** | | | |
| Course Outcome | | Upon successful completion of this practical, you will be able to:   1. Understand and Apply Parallel Programming Concepts 2. Demonstrate Practical Skills in HPC Tools and Environments | | | |
| Aim | | **Introduction to OpenMP** | | | |
| Problem Definition | | To understand and practice the fundamentals of the Linux operating system and the High-Performance Computing environment by learning basic Linux commands, remote access techniques (SSH), module handling, and job submission using a scheduler in an HPC cluster. | | | |
| Theory | | **OpenMP** stands for **Open Multi-Processing**. It's an API (Application Programming Interface) designed for writing parallel programs on **shared-memory** systems.  **Key Features:**   * **Shared Memory Multiprocessing:** All threads created by OpenMP share access to the same main memory, which makes sharing data between them easy. * **Fork-Join Model:** A program starts as a single master thread. When a parallel region is encountered, the master thread **forks** a team of slave threads. At the end of the region, the threads synchronize and **join**, leaving only the master thread to continue. * **Simple Syntax:** Parallelism is often added with simple #pragma directives, making the code clean and readable. * **Environment Control:** You can control settings like the number of threads at runtime using environment variables (e.g., OMP\_NUM\_THREADS). * **Scalability:** It scales well on modern multi-core CPUs, allowing programs to take full advantage of available hardware. | | | |
| Procedure and Execution : | | This practical will guide you through writing, compiling, and running several OpenMP programs on a CentOS/Linux system.  **Part I: Basic Loop Parallelization**  This example demonstrates how to easily parallelize a standard for loop.  **1. Sample Program: Parallel For-Loop**  Create a file named openmp\_example.c with the following code. The #pragma omp parallel for directive tells OpenMP to split the loop iterations among the available threads.  2**. Execution Steps**  Follow these steps in your Linux terminal.   * **Compile the Program:** Use the -fopenmp flag to enable OpenMP. * **Set Number of Threads:** Set the OMP\_NUM\_THREADS environment variable. * **Run the Program**   **Part II: Advanced Concepts**  These examples explore more specific OpenMP directives for controlling thread behaviour.  **Example 1: The schedule Clause**  The schedule clause controls how loop iterations are distributed among threads. **Static scheduling** divides iterations into fixed-size chunks and assigns them to threads in a round-robin fashion.  **Example 2 : Barrier for Synchronization**  A **barrier** is a synchronization point. No thread can proceed past a barrier until all other threads in the team have reached it.  reached the barrier. Thread %d continuing.\n", id);  **Example 3 : Nested Parallelism**  OpenMP allows you to create parallel regions inside other parallel regions. This must be explicitly enabled. | | | |
| Code: | | | |
| Output: | | | |
| Output Analysis : | | Part1:  The output will show different threads handling different loop iterations. The order of the printed lines will be unpredictable because the threads run concurrently.  **Part 2:**   1. The output clearly shows that iterations 0-2 go to one thread, 3-5 to the next, and so on. This predictable distribution can be useful for balancing loads when loop iterations have similar computational costs 2. All four "before barrier" messages will appear first (in any order). Only after all threads have printed this message will the final message from Thread 0 be printed. This guarantees that certain tasks are complete across all threads before the program moves on 3. The output shows two outer threads (0 and 1). Each of these outer threads then becomes a master for its own inner parallel region, spawning two new inner threads (which will also be numbered 0 and 1 within their own team). | | | |
| Link of student Github profile where lab assignment has been uploaded | | <https://github.com/aman-makode-11/HPCPractical/blob/main/35_P-3_HPC.docx> | | | |
| Conclusion | | This practical provided a comprehensive, hands-on introduction to OpenMP programming. We successfully progressed from the fundamental fork-join model and basic loop parallelization using #pragma omp parallel for to more advanced concepts. By implementing scheduling clauses, barrier synchronization, and nested parallelism, we learned how to exert finer control over thread behavior. The successful compilation and execution of these examples in a Linux environment solidify the core skills required to develop efficient, scalable parallel applications for multi-core systems. | | | |
| Plag Report (Similarity index < 12%) | |  | | | |
| Date | | **2/09/2025** | | | |