**Class:** Final Year (Computer Science and Engineering)

**Year:** 2025-26 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 4**

**PRN: 22510039**

**Title of practical:**

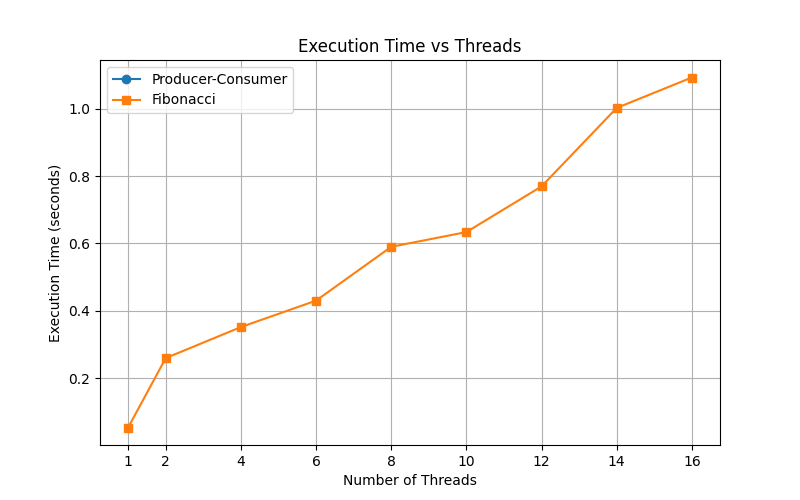
Study and Implementation of Synchronization

**Problem Statement 1:**

# Analyze and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

# Fibonacci Computation:

**Screenshots:**



**Information:**

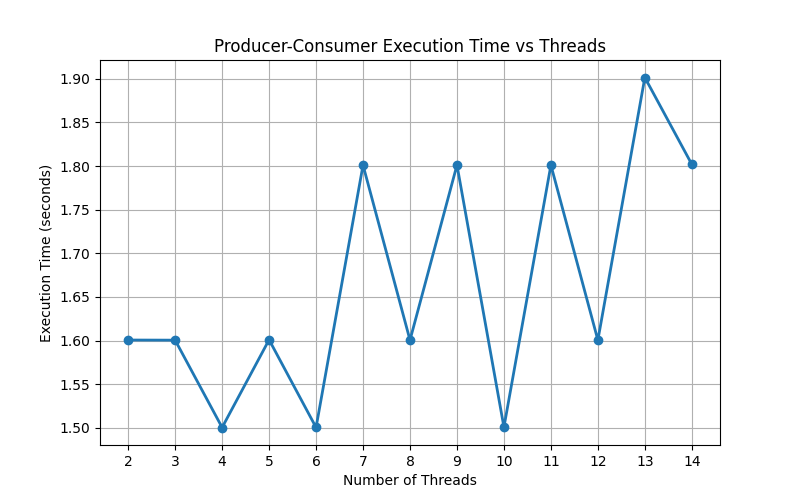
* Ideal Scaling: In a perfectly parallelizable problem, doubling the number of threads would halve the execution time. This is known as linear speedup.
* Fibonacci's Behavior: For Fibonacci, you will likely find that the execution time initially decreases but then starts to increase after a certain number of threads. This is because the overhead of creating and managing tasks eventually outweighs the benefits of parallel execution. The exponential number of tasks generated by the recursive calls puts a heavy burden on the OpenMP runtime. For smaller values of n, the overhead is especially noticeable, as the sub-problems are too small to warrant parallelization.

**Problem Statement 2:**

# Analyze and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

## Producer Consumer Problem

**Screenshots:**



**Information:**

This C program demonstrates the Producer-Consumer problem using OpenMP, a classic concurrency pattern. It simulates a scenario where producer threads add items to a shared circular buffer, while consumer threads remove them. The code uses the #pragma omp critical directive to ensure safe, synchronized access to the buffer and its state variables (in, out, and count), preventing race conditions. The main function systematically benchmarks the program's performance by varying the number of threads and measuring the execution time, highlighting the trade-off between parallel work and synchronization overhead.

**Github Link:** [**https://github.com/thundersp/hpcl**](https://github.com/thundersp/hpcl)