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

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

**Course:** High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 22510023**

**Name : Ganesh Chavhan**

**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:

### Design and Approach

Used **OpenMP task parallelism** for recursive decomposition.

**Synchronization:**

#pragma omp taskwait ensures child tasks complete.

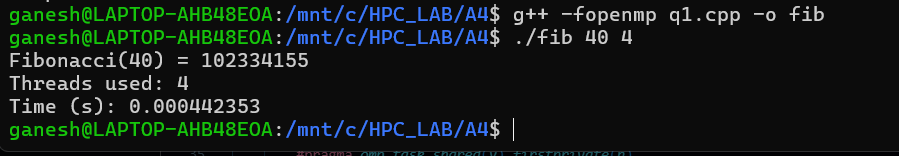
#pragma omp taskgroup ensures all tasks finish before leaving region.

**Clauses Used:** parallel, single, task, taskwait, taskgroup, firstprivate, shared.

Cutoff technique used: small subproblems solved sequentially to reduce task overhead.

**Screenshots:**





**Information:**

1. **Time complexity:** O(n) with iterative cutoff, task overhead reduced.
2. **Synchronization points:** taskwait at each recursive split, taskgroup at root.
3. **Observation:** Increasing threads gives speedup but overhead dominates for small n.

**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

### Design and Approach

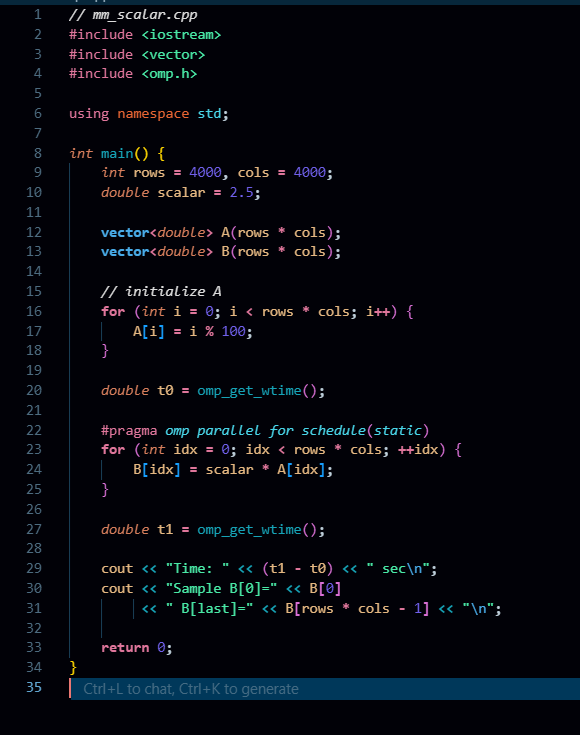
Used **circular buffer** for shared data.

**Synchronization:**

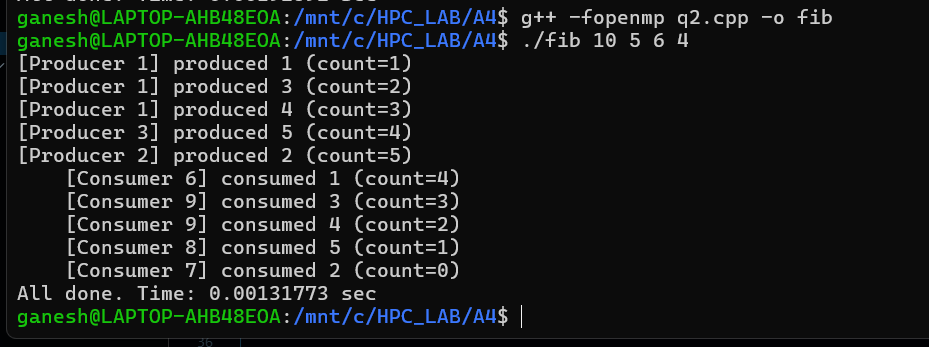
#omp\_lock\_t for mutual exclusion in buffer push/pop.

#pragma omp atomic for counters (produced, consumed, count).

Producers and consumers created as **OpenMP tasks** inside a single region.

Spin-wait used for full/empty buffer states.

**Screenshots:**



**Information:**

**Invariant:** 0 <= count <= BUF\_SIZE always holds.

**Synchronization points:** Locks on buffer, atomics for counters.

**Observation:** Increasing buffer size reduces contention; too many threads cause more lock contention.

**Github Link: <https://github.com/Ganesh-Chavhan/HPC_LAB/tree/main/A4>**