**Assignment 4**

**PRN**: 22510012

**Name**: Jai Chowgule

**Batch**: B6

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:

**Source Code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <omp.h>**

**#define N 20**

**int main() {**

**int fib[N];**

**fib[0] = 0;**

**fib[1] = 1;**

**double start, end;**

**start = omp\_get\_wtime();**

**#pragma omp parallel**

**{**

**// Step 1: Only one thread initializes first two terms**

**#pragma omp single**

**{**

**printf("Thread %d initialized first two terms\n", omp\_get\_thread\_num());**

**}**

**// Step 2: Parallel computation of Fibonacci sequence**

**#pragma omp for schedule(static) nowait**

**for (int i = 2; i < N; i++) {**

**// Protect shared updates**

**#pragma omp critical**

**{**

**fib[i] = fib[i - 1] + fib[i - 2];**

**}**

**printf("Thread %d computed fib[%d] = %d\n", omp\_get\_thread\_num(), i, fib[i]);**

**}**

**// Step 3: Barrier before printing**

**#pragma omp barrier**

**// Step 4: Safe printing**

**#pragma omp for schedule(static)**

**for (int i = 0; i < N; i++) {**

**#pragma omp critical**

**{**

**printf("fib[%d] = %d\n", i, fib[i]);**

**}**

**}**

**}**

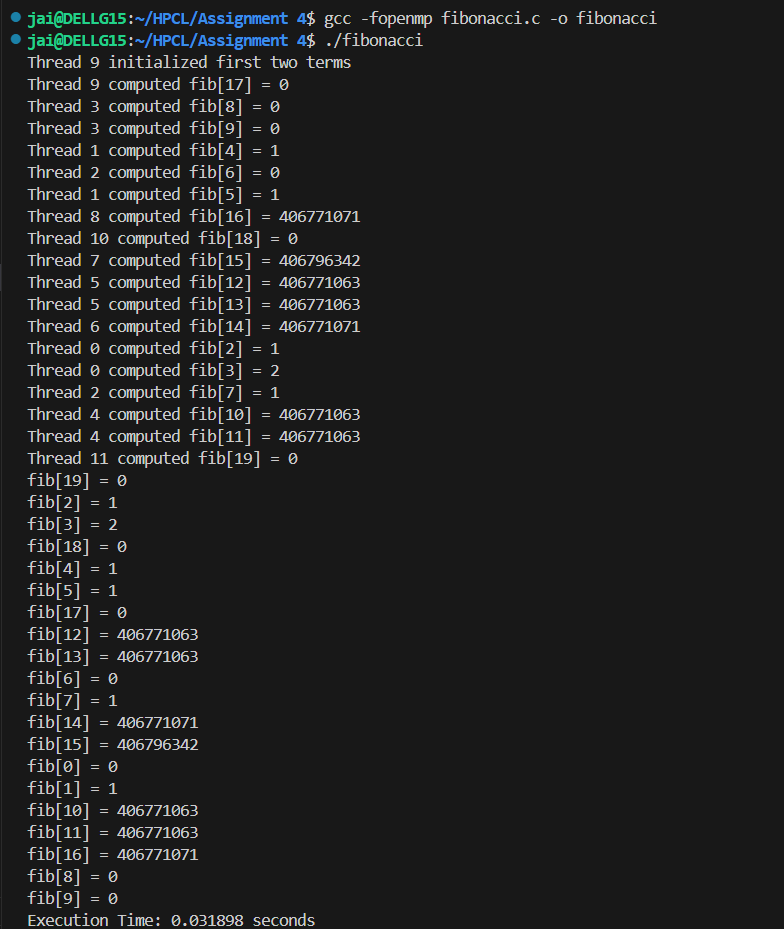
**end = omp\_get\_wtime();**

**printf("Execution Time: %f seconds\n", end - start);**

**return 0;**

**}**

**Output:**

****

**Analysis:**

Implemented parallel Fibonacci computation in C using OpenMP.

Used single for initialization, for with schedule(static) nowait for parallel work, critical for safe updates/printing, and barrier for synchronization.

Observed that synchronization ensures correctness but adds overhead, reducing speedup for small tasks.

Concluded that minimal synchronization gives better performance while maintaining correctness.

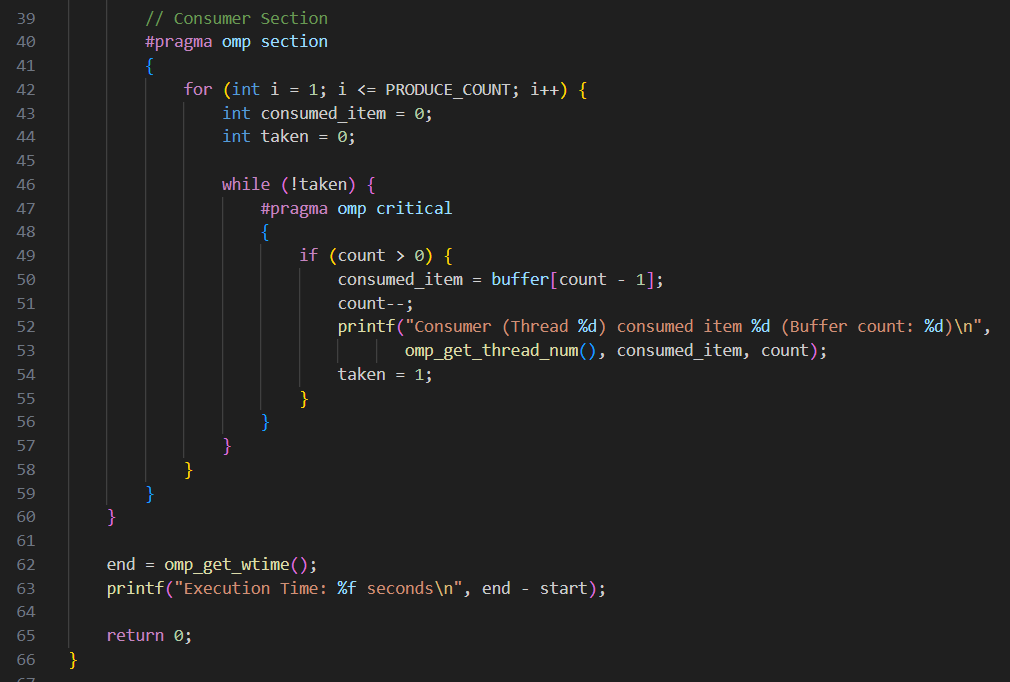
**Problem Statement-2**:

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

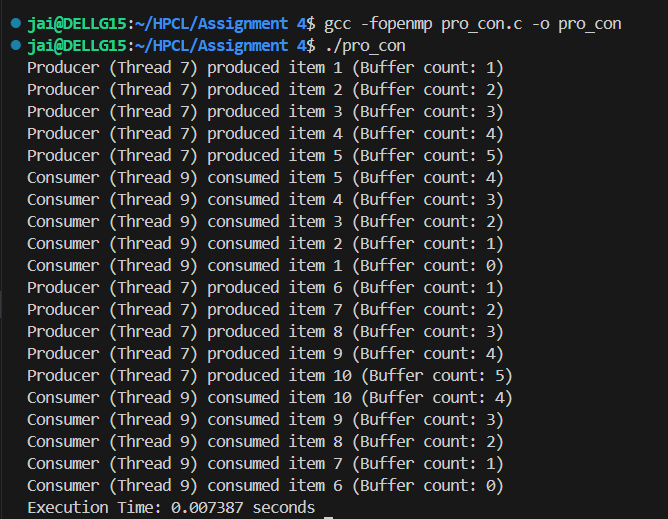
Producer Consumer Problem

**Source Code:**

****

****

**Output:**

**  
Analysis:**

Without critical, multiple threads could modify count or buffer at the same time, causing data corruption.

For a small buffer, synchronization overhead dominates.

The while loops simulate blocking behavior until a slot/item is available.

OpenMP is not ideal for producer–consumer compared to message queues, but works for demonstration.

Github Link: [**https://github.com/Jai-173/HPCL**](https://github.com/Jai-173/HPCL)