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

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 21510067**

**Name: Vardhan V. Kulkarni**

**Title of practical:**

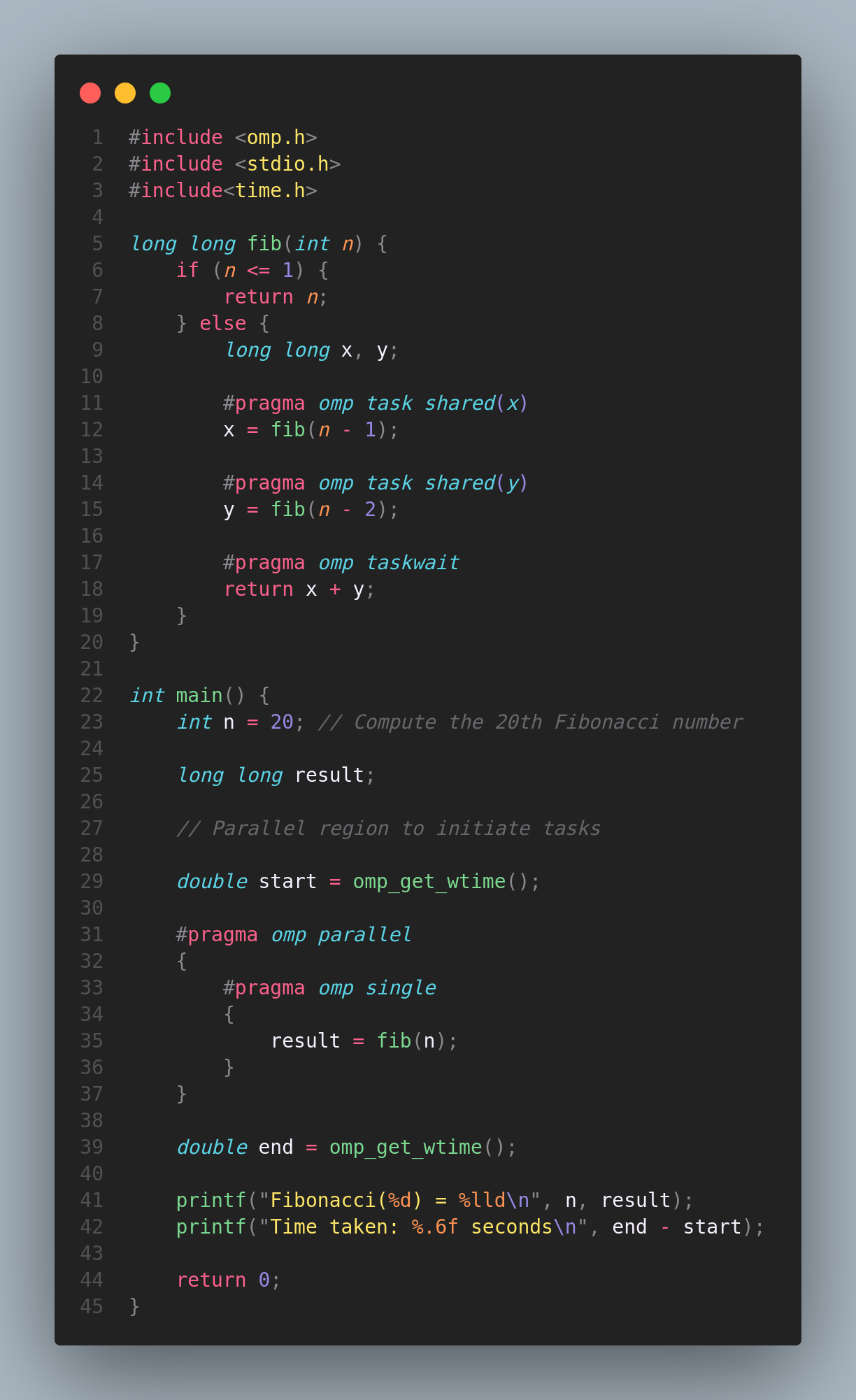
Study and Implementation of Synchronization

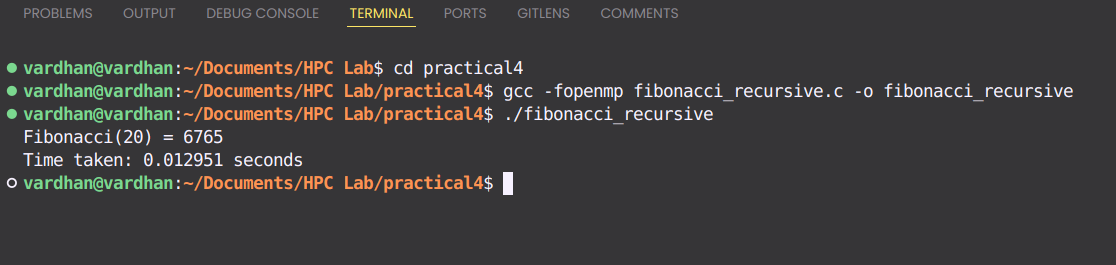
**Problem Statement 1:**

# Analyse 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:**

This program calculates the Fibonacci number using a parallel approach with OpenMP. The computation leverages the task parallelism model, which is particularly suited for the recursive nature of the Fibonacci sequence.

**Key Constructs and Clauses:**

1. Parallel Region (#pragma omp parallel): A parallel region is created to initiate multiple threads that can execute the subsequent tasks concurrently.
2. Single (#pragma omp single): Ensures that only one thread enters the Fibonacci function initially, creating tasks for the recursive calls.
3. Task (#pragma omp task): The recursive calls to compute fib(n-1) and fib(n-2) are parallelized using tasks. This allows different threads to work on these tasks simultaneously, exploiting the available parallelism.
4. Taskwait (#pragma omp taskwait): The taskwait directive ensures that the thread waits for the completion of the tasks created for fib(n-1) and fib(n-2) before combining their results. This synchronization is crucial to ensure that the correct values are returned.

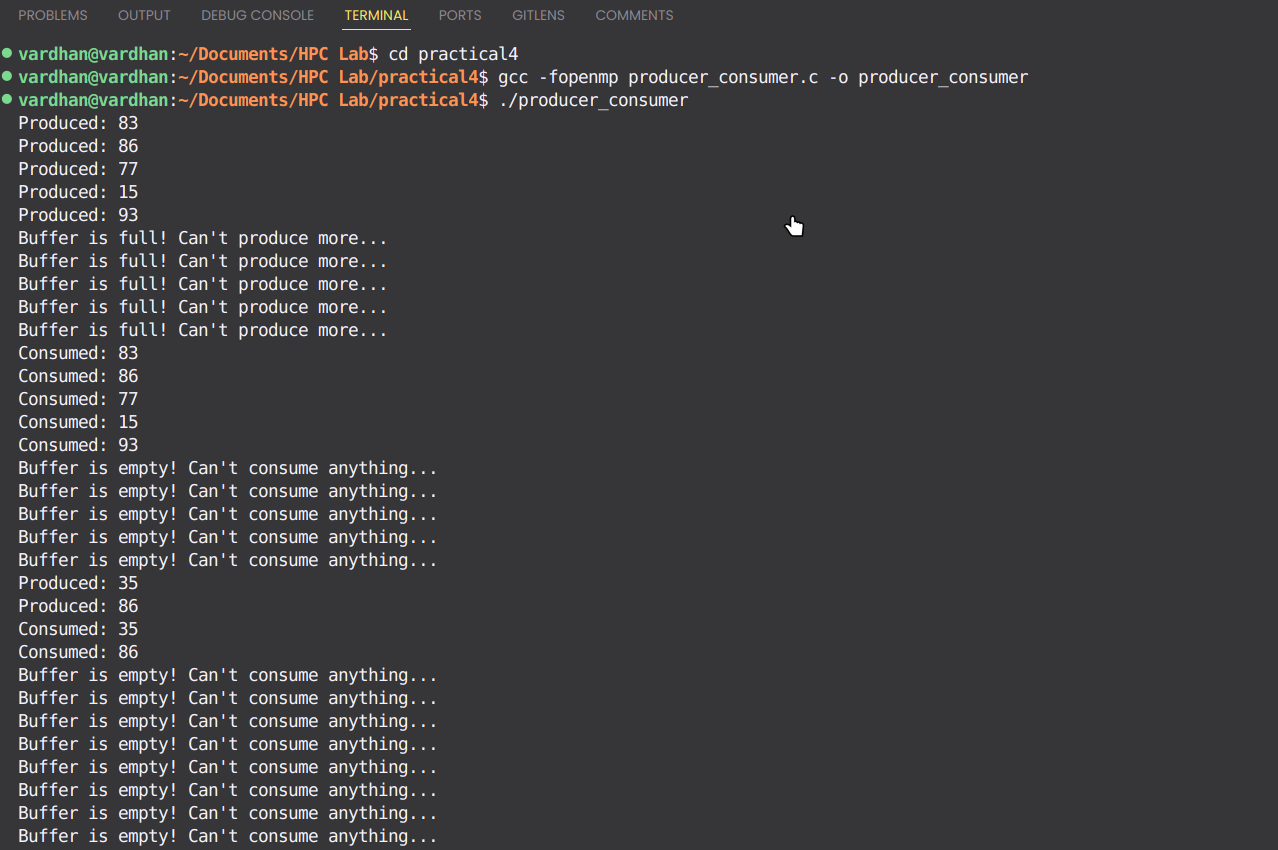
**Problem Statement 2:**

# Analyse 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 program implements the Producer-Consumer problem using OpenMP to manage the synchronization of shared resources between multiple producer and consumer threads. The key challenge is ensuring that producers can safely add items to the buffer and consumers can remove items without conflicts, even when operating in parallel.

Key Constructs and Clauses:

1. Critical Section (#pragma omp critical): The critical directive is used to protect the sections of code where producers add items to the buffer and consumers remove items. This ensures that only one thread at a time can modify the shared buffer, preventing race conditions and ensuring data consistency.
2. Parallel Region (#pragma omp parallel): A parallel region is created with a specified number of threads, combining both producers and consumers. Each thread either performs the role of a producer or a consumer based on its thread ID.
3. Thread Identification (omp\_get\_thread\_num()): The program distinguishes between producer and consumer threads using the thread ID, allowing different threads to perform specific tasks based on their ID.
4. Buffer Management: The circular buffer is managed using in and out indices, which are updated in a thread-safe manner within the critical sections. This ensures that the buffer operates correctly as a circular queue, with producers writing to the in index and consumers reading from the out index.

**Github Link:** [**Practical 4 Repository**](https://github.com/git8vardhan/HPCL/tree/master/practical4)