**Walchand College of Engineering, Sangli**

**Department of Computer Science and Engineering**

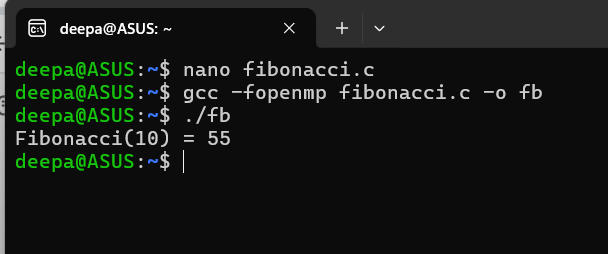
**Class: FY M.Tech (Computer Science and Engineering)**

**Year: 2024-25 Semester: 2**

**Course: High Performance Computing Lab**

**Screenshots**:

**Output**:



**Information**:

**Recursive Parallelization:**

* The Fibonacci function is recursively computed with #pragma omp parallel sections.
* Two sections compute fibonacci(n-1) and fibonacci(n-2) in parallel.

**Synchronization:**

* #pragma omp single ensures that only one thread computes the Fibonacci function in main.

**Parallelism Management:**

* Sections are used to divide computations between threads.
* OpenMP efficiently manages thread execution.

**OpenMP Constructs**

**#pragma omp parallel**

This creates a team of threads to execute the enclosed block of code.

In fibonacci(int n), it ensures parallel execution of Fibonacci computation.

**#pragma omp single nowait**

Ensures that only one thread executes the enclosed block.

The nowait clause prevents implicit synchronization, allowing other threads to continue execution without waiting.

**#pragma omp task shared(x) & #pragma omp task shared(y)**

These create independent tasks for computing fibonacci(n-1) and fibonacci(n-2).

The shared(x) and shared(y) clauses ensure that the computed values are accessible by all threads.

**#pragma omp taskwait**

This directive forces the thread to wait until all previously created tasks (x and y computations) are completed before proceeding to the next step.

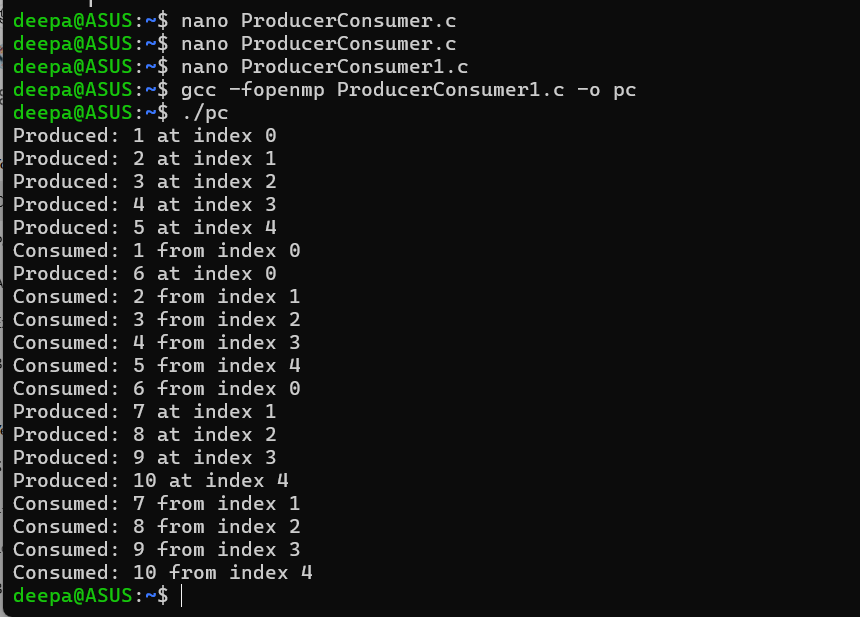
It ensures proper synchronization so that x + y returns the correct result.

**#pragma omp single in main()**

Ensures only one thread executes fibonacci(n) while other threads remain idle, preventing redundant computations.

**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 of output:**

****

**Information:**

1. #pragma omp parallel sections

* Used to run producer() and consumer() in parallel.

2. #pragma omp critical

* Ensures only one thread modifies the buffer, count, in, or out at a time.
* Prevents race conditions.

3. Buffer Synchronization (Busy Waiting)

* while (count == BUFFER\_SIZE); ensures producer waits if the buffer is full.
* while (count == 0); ensures consumer waits if the buffer is empty.

**Github Link:https://github.com/DeepaH29/HPCPracticals**