

**Name:** Meet Vipul Gandhi

**PRN:** 2020BTECS00112

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

**Year:** 2023-24

**Semester:** 1

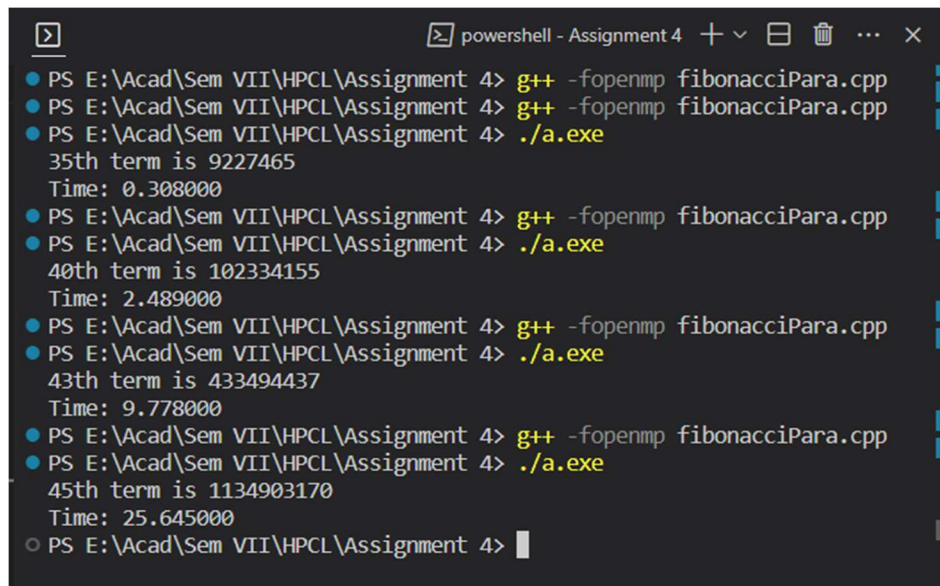
**Course:** High Performance Computing Lab

### Practical No. 4

**Title of practical:** Study and Implementation of Synchronization. Analyze and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

**Problem Statement 1:** Fibonacci Computation

**Screenshots:**



```
PS E:\Acad\Sem VII\HPCL\Assignment 4> g++ -fopenmp fibonacciPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 4> g++ -fopenmp fibonacciPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 4> ./a.exe
35th term is 9227465
Time: 0.308000
PS E:\Acad\Sem VII\HPCL\Assignment 4> g++ -fopenmp fibonacciPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 4> ./a.exe
40th term is 102334155
Time: 2.489000
PS E:\Acad\Sem VII\HPCL\Assignment 4> g++ -fopenmp fibonacciPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 4> ./a.exe
43th term is 433494437
Time: 9.778000
PS E:\Acad\Sem VII\HPCL\Assignment 4> g++ -fopenmp fibonacciPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 4> ./a.exe
45th term is 1134903170
Time: 25.645000
PS E:\Acad\Sem VII\HPCL\Assignment 4>
```

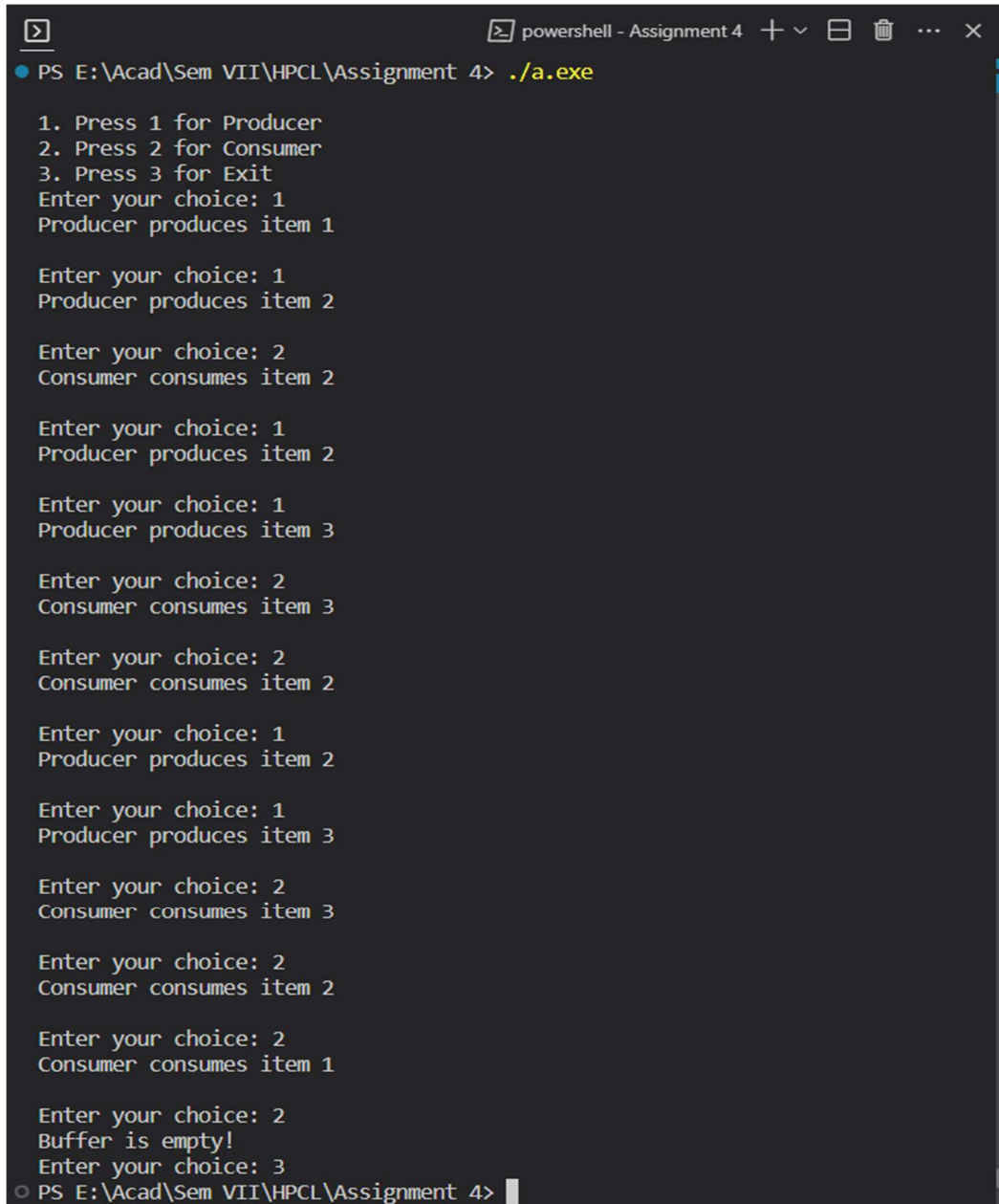
### Information:

In this example, we're using tasks to parallelize the Fibonacci calculation. Here's how it works:

- fibonacci(n) is a recursive function that calculates the nth Fibonacci term. If n is less than or equal to 1, it returns n.
- In the recursive case, two tasks are created ( $x = \text{fibonacci}(n - 1)$  and  $y = \text{fibonacci}(n - 2)$ ).
- The taskwait directive ensures that the program waits until both tasks are completed before summing their results.
- In the main function, a single parallel region is created to start the Fibonacci calculation.

Keep in mind that using tasks for Fibonacci might not always result in a performance improvement, as the overhead of task creation and management can sometimes outweigh the benefits of parallelization. For more complex problems, different algorithms (like memoization or iterative approaches) may be more suitable. Additionally, the cutoff condition for task creation (in this case if  $(n > 20)$ ) might need to be adjusted based on your system's performance characteristics.

**Screenshots:**



```
PS E:\Acad\Sem VII\HPCL\Assignment 4> ./a.exe

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice: 1
Producer produces item 1

Enter your choice: 1
Producer produces item 2

Enter your choice: 2
Consumer consumes item 2

Enter your choice: 1
Producer produces item 2

Enter your choice: 1
Producer produces item 3

Enter your choice: 2
Consumer consumes item 3

Enter your choice: 2
Consumer consumes item 2

Enter your choice: 1
Producer produces item 2

Enter your choice: 1
Producer produces item 3

Enter your choice: 2
Consumer consumes item 3

Enter your choice: 2
Consumer consumes item 2

Enter your choice: 2
Consumer consumes item 1

Enter your choice: 2
Buffer is empty!
Enter your choice: 3

PS E:\Acad\Sem VII\HPCL\Assignment 4>
```

**Information:**

- a. We use **#pragma omp critical** sections to ensure that only one thread at a time can access the shared data structures (buffer in this case).
- b. The producer produces 10 items, and the consumer consumes 10 items.
- c. This is to properly synchronize the problem.

**GitHub Link:** <https://github.com/meetgandhi692/HPC-Lab/tree/79b911b51b14aed89f30d1a35fe33b2761b4459f/Assignment%204>