Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2021-22 **Semester:** 1

Course: High Performance Computing Lab

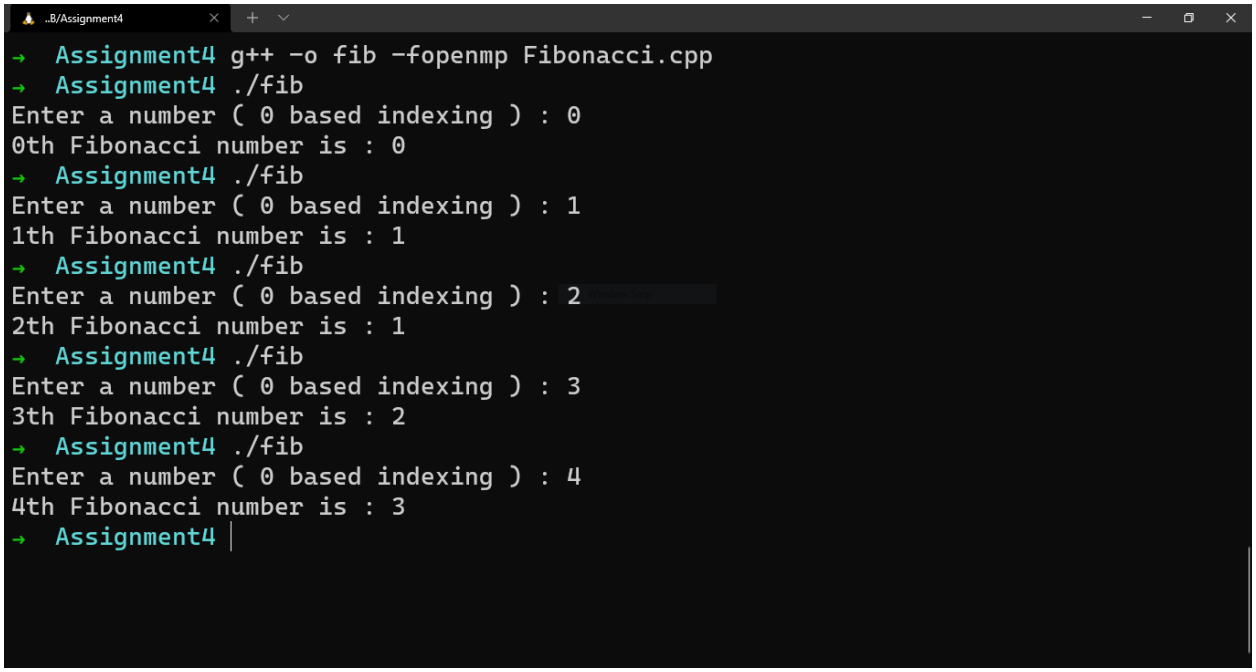
Practical No. 4

Exam Seat No:

1. 2018BTECS00033 - Mahendra Bhimrao Gharge

Problem Statement 1:
Q1: Analyse and implement a Parallel code for below program using OpenMPconsideringsynchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable) (Fibonacci)

Screenshot 1:



Information 1: As for the fibonacci series, for every number the previous two numbers should be calculated, we must use the ordered clause so that the whole operation goes sequentially.

```c
#include<stdio.h>
#include<omp.h>

int fib(int n)
{
    int f[n+2];
    int i;
    f[0] = 0;
    f[1] = 1;
    #pragma omp ordered
        for (i = 2; i <= n; i++)
        {
            f[i] = f[i-1] + f[i-2];
        }
     return f[n];
}

int main ()
{
    int n=0;
    printf("Enter a number ( 0 based indexing ) : ");
    scanf("%d",&n);
    printf("%dth Fibonacci number is : %d\n", n, fib(n));
    return 0;
}
```

Code for fibonacci with ordered clause.

Problem Statement 2:

Q2: Analyse and implement a Parallel code for below program using OpenMPconsideringsynchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable) (Producer consumer problem)

Screenshot 2:

```
../B/Assignment4
→  Assignment4 g++ -o pcp -fopenmp PCPparallel.cpp
→  Assignment4 ./pcp
Thread: 1 produces item 1
Thread: 5 produces item 2
Thread: 2 produces item 3
Thread: 0 produces item 4
Thread: 0 consumes item 4
Thread: 5 consumes item 3
Thread: 3 produces item 3
Thread: 3 consumes item 3
Thread: 7 produces item 3
Thread: 7 consumes item 3
Thread: 4 produces item 3
Thread: 4 consumes item 3
Thread: 2 consumes item 2
Thread: 6 produces item 2
Thread: 6 consumes item 2
Thread: 1 consumes item 1
→  Assignment4
```

Information 2: For producer consumer problem, we use lock variable. SO we create that lock with omp's builtin functions as shown in the code below.

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int full = 0;
int empty = 10, x = 0;

// create mutex lock variable
omp_lock_t mutex;

void producer()
{
    // set the mutex lock
```

```
    omp_set_lock(&mutex);
    if (empty == 0)
    {
        printf("Thread: %dBuffer is full!\n", omp_get_thread_num());
        omp_unset_lock(&mutex);
        return;
    }
    ++full;

    --empty;

    x++;
    printf("Thread: %d produces item %d\n", omp_get_thread_num(), x);

    // unset the mutex lock
    omp_unset_lock(&mutex);
}

void consumer()
{
    // set the mutex lock
    omp_set_lock(&mutex);
    if (full == 0)
    {
        printf("Thread: %d Buffer is empty!\n", omp_get_thread_num());
        x = 0;
        omp_unset_lock(&mutex);
        return;
    }

    --full;

    ++empty;
    printf("Thread: %d consumes item %d\n", omp_get_thread_num(), x);
    x--;

    // unset the mutex lock
```

```
    omp_unset_lock(&mutex);
}

int main()
{
    // initialize the mutex lock
    omp_init_lock(&mutex);
    omp_set_num_threads(8);
#pragma omp parallel
    {
        producer();
        consumer();
    }
    // remove the mutex lock
    omp_destroy_lock(&mutex);
}
```

Code for producer consumer problem using opm's building lock mechanism.

Github Link: https://github.com/g-mahendra/HPC_LAB_ASSIGNMENTS