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

Class: Final Year (Computer Science and Engineering)

Year: 2025-26 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 4

Exam Seat No: 22510064

Github Link: [Sem-7-Assign/HPC lab at main · parshwa913/Sem-7-Assign · GitHub]

Title of practical:

Study and Implementation of Synchronization

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:

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

int main() {
    int n, i;
    printf("Enter number of Fibonacci terms: ");
    scanf("%d", &n);

    if (n < 1) {
        printf("Number of terms must be positive.\n");
        return 0;
    }

    long long fib[n];
    fib[0] = 0;
    if (n > 1) fib[1] = 1;
```

```
    #pragma omp parallel
    {
        #pragma omp single
        {
            for (i = 2; i < n; i++) {
                #pragma omp critical
                {
                    fib[i] = fib[i - 1] + fib[i - 2];
                }
            }
        }
    }

    printf("Fibonacci Series: ");
    for (i = 0; i < n; i++)
        printf("%lld ", fib[i]);
    printf("\n");

    return 0;
}
```

Screenshots:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4> gcc -fopenmp fibonacci.c -o fibonacci
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4> ./fibonacci
Enter number of Fibonacci terms: 8
Fibonacci Series: 0 1 1 2 3 5 8 13
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4> gcc -fopenmp fibonacci.c -o fibonacci
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4> ./fibonacci
Enter number of Fibonacci terms: 12
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34 55 89
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4>
```

Information:

In parallel programming, synchronization is needed to prevent race conditions when multiple threads update shared data.
OpenMP provides synchronization constructs like:

#pragma omp critical — ensures only one thread executes the section at a time.

#pragma omp barrier — all threads wait until all have reached the barrier.

The Fibonacci sequence is computed as:
F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)

Since multiple threads may update the shared fib[] array, we use critical sections.

Algorithm:

1. **Accept number of terms n.**
2. **Initialize fib[0] = 0, fib[1] = 1.**
3. **Parallelize loop from i = 2 to n-1.**
4. **Use #pragma omp critical to ensure safe updates.**
5. **Display the Fibonacci sequence.**

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

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

#define SIZE 5

int main() {
    int buffer[SIZE];
    int count = 0; // items in buffer
    int i;

    #pragma omp parallel num_threads(2) shared(buffer, count)
private(i)
    {
        int tid = omp_get_thread_num();
```

```
        if (tid == 0) {
            // Producer
            for (i = 0; i < SIZE; i++) {
                #pragma omp critical
                {
                    buffer[count] = i * 10;
                    printf("Producer produced: %d\n", buffer[count]);
                    count++;
                }
                #pragma omp barrier
            }
        }
        else {
            // Consumer
            for (i = 0; i < SIZE; i++) {
                #pragma omp barrier
                #pragma omp critical
                {
                    printf("Consumer consumed: %d\n", buffer[count -
1]);

                    count--;
                }
            }
        }
    }
    return 0;
}
```

Screenshots:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4> gcc -fopenmp producer_consumer.c -o producer_consumer
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4> ./producer_consumer
Producer produced: 0
Consumer consumed: 0
Producer produced: 10
Consumer consumed: 10
Producer produced: 20
Consumer consumed: 20
Producer produced: 30
Consumer consumed: 30
Producer produced: 40
Consumer consumed: 40
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A4>
```

Information:

Theory:
The Producer–Consumer problem is a classic synchronization example.
Producer: Generates data and stores it in a buffer.
Consumer: Removes data from the buffer.
Shared variables (buffer, count) must be accessed safely to avoid race conditions.

We use:

#pragma omp critical to ensure exclusive buffer access.
#pragma omp barrier to synchronize production and consumption steps.

Algorithm:

Create a buffer of fixed size SIZE.
Create two threads: Producer and Consumer.
Producer: Adds items to buffer inside a critical section, then hits a barrier.
Consumer: Waits at barrier, then consumes items inside a critical section.
Repeat until all items are produced and consumed.