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

Course: High Performance Computing Lab

Practical No. 4

PRN: 22510057

Name: Ashutosh Gundu Birje

Batch: B8

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:

Screenshots:

Assignment_No_4 > 01_.c > main()

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

#define N 20

int main() {
    int fib[N];
    fib[0] = 0;
    fib[1] = 1;

    #pragma omp parallel
    {
        #pragma omp single
        {
            for (int i = 2; i < N; i++) {
                #pragma omp task firstprivate(i)
                {
                    #pragma omp taskwait
                    fib[i] = fib[i-1] + fib[i-2];
                }
            }
        }
    }

    printf("Fibonacci Sequence:\n");
    for (int i = 0; i < N; i++) {
        printf("%d ", fib[i]);
    }
    printf("\n");

    return 0;
}
```

```
E:\ashutosh\LY\SEM 1\LAB\HPC>cd "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_4\" && gcc 01_.c -
o 01_ && "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_4\"01_
Fibonacci Sequence:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181

e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_4>
```

Information:

Synchronization:
In Fibonacci computation, each element fib[i] depends on the previous two (fib[i-1] and fib[i-2]). Without proper synchronization, threads might read uninitialized values.

OpenMP Constructs Used:

#pragma omp parallel	**Creates multiple threads for parallel execution.**
#pragma omp single	**Ensures only one thread initiates the tasks.**
#pragma omp task	**Defines independent units of work to be executed in parallel.**
#pragma omp taskwait	**Waits for all created tasks to complete before proceeding.**

Clauses Demonstrated:
firstprivate(i): Ensures each task gets its own copy of i.
taskwait: Synchronizes dependent calculations.

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

Screenshots:

Assignment_No_4 > C 02_.c > ...

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

#define BUFFER_SIZE 5
#define PRODUCE_COUNT 10

int buffer[BUFFER_SIZE];
int count = 0;

void producer() {
    for (int i = 1; i <= PRODUCE_COUNT; i++) {
        #pragma omp critical
        {
            if (count < BUFFER_SIZE) {
                buffer[count] = i;
                count++;
                printf("Produced: %d (Buffer count: %d)\n", i, count);
            }
        }
    }
}
```

Assignment_No_4 > C 02_.c > ...

```c
void consumer() {
    for (int i = 1; i <= PRODUCE_COUNT; i++) {
        #pragma omp critical
        {
            if (count > 0) {
                int item = buffer[count - 1];
                count--;
                printf("Consumed: %d (Buffer count: %d)\n", item, count);
            }
        }
    }
}

int main() {
    #pragma omp parallel sections
    {
        #pragma omp section
        producer();

        #pragma omp section
        consumer();
    }

    return 0;
}
```

```
e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_4>cd "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_4\
" && gcc 02_.c -o 02_ && "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_4\"02_
Produced: 1 (Buffer count: 1)
Produced: 2 (Buffer count: 2)
Produced: 3 (Buffer count: 3)
Produced: 4 (Buffer count: 4)
Produced: 5 (Buffer count: 5)
Consumed: 5 (Buffer count: 4)
Consumed: 4 (Buffer count: 3)
Consumed: 3 (Buffer count: 2)
Consumed: 2 (Buffer count: 1)
Consumed: 1 (Buffer count: 0)
```

Information:

Synchronization:

Multiple threads try to modify shared variables (buffer[] and count). Without synchronization, race conditions occur.

The critical section ensures only one thread modifies buffer and count at a time.

OpenMP Constructs Used:

#pragma omp parallel sections	**Divides work between producer and consumer.**
#pragma omp section	**Assigns specific blocks to different threads.**
#pragma omp critical	**Ensures mutual exclusion when accessing shared data.**

Clauses Demonstrated:
critical: Prevents simultaneous access to the shared buffer.
sections/section: Used to split producer and consumer logic between threads.

Github Link:
https://github.com/Ashutoshbirje/HPC-LAB/tree/master/Assignment_No_4