Practical 4

PRN: 23520005
Name: Manas Indrapal Gedam
Batch: B3

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:

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

#define N 10

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

    // Parallel computation of Fibonacci using synchronization
    #pragma omp parallel for shared(fib)
    for (int i = 2; i < N; i++) {
        // Each fib[i] depends on previous values
        // Synchronize using critical to avoid race conditions
        #pragma omp critical
        {
            fib[i] = fib[i-1] + fib[i-2];
        }
    }

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

    return 0;
}
```

```
manas@Manass-MacBook-Air HPCL 4 % gcc-15 -fopenmp Q.c -o Q
manas@Manass-MacBook-Air HPCL 4 % ./Q
Fibonacci Series up to 10 terms:
0 1 1 644894 1289787 1934681 1289788 1289789 2579577 3869366
```

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

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

#define SIZE 5
#define NUM_ITEMS 10

int buffer[SIZE];
int count = 0; // items in buffer

int main() {
    omp_set_num_threads(2); // one producer, one consumer

    #pragma omp parallel sections
    {
        // Producer Thread
        #pragma omp section
        {
            int items_produced = 0;
            while (items_produced < NUM_ITEMS) {
                #pragma omp critical
                {
                    // Wait until there is space in the buffer
                    if (count < SIZE) {
                        int item_to_produce = items_produced + 1;
                        buffer[count] = item_to_produce;
                        count++;
                        printf("Producer produced: %d (buffer count = %d)\n", item_to_produce, count);
                        items_produced++;
                    }
                }
                // No barrier here! Allows producer to work independently.
            }
        }

        // Consumer Thread
        #pragma omp section
        {
            int items_consumed = 0;
            while (items_consumed < NUM_ITEMS) {
                #pragma omp critical
                {
                    // Wait until there is an item in the buffer
                    if (count > 0) {
                        count--;
                        int item = buffer[count];
                        printf("Consumer consumed: %d (buffer count = %d)\n", item, count);
                        items_consumed++;
                    }
                }
                // No barrier here! Allows consumer to work independently.
            }
        }
    }
    return 0;
}
```

```
manas@Manass-MacBook-Air HPCL 4 % gcc-15 -fopenmp Q2.c -o Q2
manas@Manass-MacBook-Air HPCL 4 % ./Q2
Producer produced: 1 (buffer count = 1)
Producer produced: 2 (buffer count = 2)
Producer produced: 3 (buffer count = 3)
Producer produced: 4 (buffer count = 4)
Producer produced: 5 (buffer count = 5)
Consumer consumed: 5 (buffer count = 4)
Consumer consumed: 4 (buffer count = 3)
Consumer consumed: 3 (buffer count = 2)
Consumer consumed: 2 (buffer count = 1)
Consumer consumed: 1 (buffer count = 0)
Producer produced: 6 (buffer count = 1)
Producer produced: 7 (buffer count = 2)
Producer produced: 8 (buffer count = 3)
Producer produced: 9 (buffer count = 4)
Producer produced: 10 (buffer count = 5)
Consumer consumed: 10 (buffer count = 4)
Consumer consumed: 9 (buffer count = 3)
Consumer consumed: 8 (buffer count = 2)
Consumer consumed: 7 (buffer count = 1)
Consumer consumed: 6 (buffer count = 0)
manas@Manass-MacBook-Air HPCL 4 %
```