Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

Final Year: High Performance Computing Lab 2022-23 Sem I

Class: Final Year (Computer Science and Engineering)

Year: 2022-23 Semester: 1

Course: High Performance Computing Lab

Assignment No : 4

Q1: Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

Sequential Code :

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


int fib(int n){
    int i;
    int f[n+2];
    f[0] = 1;
    f[1] = 1;

    for (i = 2; i <= n; i++)
    {
        f[i] = f[i-1] + f[i-2];
    }

    return f[n];

}



int main ()
{
    int n = 100000;

    double startTime = omp_get_wtime();
    printf("%d\n", fib(n));
    double endTime = omp_get_wtime();

    printf("%f\n" , endTime - startTime);


    return 0;

}
```

```
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_04$ g++ -fopenmp
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_04$ ./a.out
48392605
Execution Time : 0.008897
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_04$
```

Parallel Code :

04 > C Q1.c > main()

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


int fib(int n){
    int i;
    int f[n+2];

    #pragma omp parallel for schedule(static, 8)
    for(int i=0;i<n+2;i++){
        f[i] = -1;
    }

    f[0] = 1;
    f[1] = 1;

    #pragma omp parallel for private(i) shared(f) schedule(static, 8)
    for(i=2;i<=n;i++)
    {
        while(f[i-1] == -1 || f[i-2] == -1);

        #pragma omp critical
        {
            f[i] = f[i-1] + f[i-2];
        }

    }

    return f[n];

}



int main ()
{
    int n = 1000000;

    double startTime = omp_get_wtime();

    #pragma omp parallel shared(n)
    {
        #pragma omp single
        printf("%d\n", fib(n));
    }

    double endTime = omp_get_wtime();

    printf("%f\n" , endTime - startTime);


    return 0;

}
```

```
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_04$ ./a.out
48392605
0.037241
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_04$
```

Information 1:

The shared clause declares the variables in the list to be shared among all the threads in a team. All threads within a team access the same storage area for shared variables.The firstprivate clause provides a superset of the functionality provided by the private clause. The private variable is initialized by the original value of the variable when the parallel construct is encountered.

Q2: Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

Serial Code :

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

int mutex = 1;
int full = 0;

int empty = 10, x = 0;
void producer()
{
    --mutex;
    ++full;
    --empty;
    printf("\nProducer produces" "item %d",x);
    x++;
    ++mutex;
}
void consumer()
{
    --mutex;
    --full;
    ++empty;
    printf("\nConsumer consumes " "item %d",x);
    x--;
    ++mutex;
}
int main()
{

    int n, i;
    printf("\n1. Press 1 for Producer" "\n2. Press 2 for Consumer" "\n3. Press 3 for Exit");
    for (i = 1; i > 0; i++) {
        printf("\nEnter your choice:");
        scanf("%d", &n);
        switch (n) {
            case 1:
                if ((mutex == 1) && (empty != 0)) {
                    producer();

                }
                else {
                    printf("Buffer is full!");
                }

            break;
            case 2:
                if ((mutex == 1) && (full != 0)) {
                    consumer();
                }
                else {
                    printf("Buffer is empty!");
                }

                break;
            case 3:
                exit(0); break;
        }

    }
```

Parallel Code :

```c
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0 , y = 0;
void producer(){
    #pragma omp critical
    {
        --mutex;
        ++full;
        --empty;
        printf("\nProducer produces" "item %d",x);
        x++;
        ++mutex;
    }
}
void consumer()
{
    #pragma omp critical
    {
        --mutex;
        --full;
        ++empty;
        printf("\nConsumer consumes " "item %d",y);
        y++;
        ++mutex;
    }

}
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer" "\n2. Press 2 for Consumer" "\n3. Press 3 for Exit");
    #pragma omp parallel for private(i) schedule(static , 8)
    for (i = 1; i > 0; i++) {
        printf("\nEnter your choice:");
        scanf("%d", &n);
        switch (n) {
            case 1:
                if ((mutex == 1) && (empty != 0)) {
                    producer();

                }
                else {
                    printf("Buffer is full!");
                }
            break;
            case 2:
                if ((mutex == 1) && (full != 0)) {
                    consumer();
                }
                else {
                    printf("Buffer is empty!");
                }
                break;
            case 3:
                exit(0);
                break;
        }
```

Information 2:

A thread waits at the start of a critical region identified by a given name until no other thread in the program is executing a critical region with that same name. Critical sections not specifically named by omp critical directive invocation are mapped to the same unspecified name.