Date: 05/09/2022

High Performance Computing Lab

Assignment - 4

PRN: 2019BTECS00034
Name: Rutikesh Sawant

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).

Code:

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

int fibonacci(int n) {
    int i, j;
    if (n < 2)
        return n;
    else {
#pragma omp task shared(i)
        i = fibonacci(n - 1);
#pragma omp task shared(j)
        j = fibonacci(n - 2);
#pragma omp taskwait
        return i + j;
    }
}
```

```
int main(int argc, char **argv) {
    char *a = argv[1];
    int n = atoi(a), result;
#pragma omp parallel
    {
#pragma omp single
        result = fibonacci(n);
    }
    printf("Result is %d\n", result);
}
```

```
Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ g++ -fopenmp q1.cpp

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ ./a.exe 1
Result is 1

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ ./a.exe 2
Result is 1

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ ./a.exe 3
Result is 2

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ ./a.exe 5
Result is 5

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ ./a.exe 10
Result is 55

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ ./a.exe 15
Result is 610

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$
```

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).

Producer Consumer Problem:
Using critical clause which allows only one thread to execute part of a program

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

```c
#include <stdlib.h>

// Initialize a mutex to 1
int mutex = 1;
// Number of full slots as 0a
int full = 0;
// Number of empty slots as size
// of buffer

int empty = 10, x = 0;
// Function to produce an item and
// add it to the buffer
void producer() {
    // Decrease mutex value by 1
    --mutex;
    // Increase the number of full
    // slots by 1
    ++full;
    // Decrease the number of empty
    // slots by 1
    --empty;
    // Item produced
    x++;
    printf("\nProducer produces "
           "item %d",
           x);
    // Increase mutex value by 1
    ++mutex;
}
// Function to consume an item and
// remove it from buffer
void consumer() {
```

```
    // Decrease mutex value by 1
    --mutex;
    // Decrease the number of full
    // slots by 1
    --full;
    // Increase the number of empty
    // slots by 1
    ++empty;
    printf("\nConsumer consumes "
           "item %d",
           x);
    x--;
    // Increase mutex value by 1
    ++mutex;
}
// Driver Code
int main() {
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");
// Using '#pragma omp parallel for'
// can give wrong value due to
// synchronization issues.
// 'critical' specifies that code is
// executed by only one thread at a
// time i.e., only one thread enters
// the critical section at a given time
#pragma omp critical
    for (i = 1; i > 0; i++) {
        printf("\nEnter your choice:");
        scanf("%d", &n);
```

```
        // Switch Cases
        switch (n) {
        case 1:
            // If mutex is 1 and empty
            // is non-zero, then it is
            // possible to produce
            if ((mutex == 1) && (empty != 0)) {
                producer();
            }
            // Otherwise, print buffer
            // is full
            else {
                printf("Buffer is full!");
            }
            break;
        case 2:
            // If mutex is 1 and full
            // is non-zero, then it is
            // possible to consume
            if ((mutex == 1) && (full != 0)) {
                consumer();
            }
            // Otherwise, print Buffer
            // is empty
            else {
                printf("Buffer is empty!");
            }
            break;
        // Exit Condition
        case 3:
            exit(0);
            break;
```

```
            }
        }
}
```

```
Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ g++ -fopenmp q2.cpp

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/HPC Lab/Assignment 4
$ ./a.exe

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:1

Producer produces item 4
Enter your choice:2

Consumer consumes item 4
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:2
```

```
Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:1

Producer produces item 4
Enter your choice:1

Producer produces item 5
Enter your choice:1

Producer produces item 6
Enter your choice:1

Producer produces item 7
Enter your choice:1

Producer produces item 8
Enter your choice:1

Producer produces item 9
Enter your choice:1

Producer produces item 10
Enter your choice:1
Buffer is full!
```

```
Producer produces item 10
Enter your choice:1
Buffer is full!
Enter your choice:1
Buffer is full!
Enter your choice:1
```