

High Performance Computing Lab : 04

Study and Implementation of Synchronization constructs

PRN : 2019BTECS00055

Name : Aditya Rajendra Manapure

Problem Statement 1:

To analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements.

Screenshot 1:

```
Assignment-4 > C++ Ques1_Fibonacci.cpp > fib(int)
1 //Fibonacci Computation
2 //Author : 2019BTECS00055 Aditya Manapure
3 #include<stdio.h>
4 #include<omp.h>
5
6 #define THREAD_COUNT 8
7
8 int fib(int n)
9 {
10     int i, j;
11     if (n < 2)
12         return n;
13     else
14     {
15         #pragma omp task shared(i) firstprivate(n)
16         i = fib(n - 1);
17
18         #pragma omp task shared(j) firstprivate(n)
19         j = fib(n - 2);
20
21         #pragma omp taskwait
22         return i + j;
23     }
24 }
25
26
27 int main ()
28 {
29     int n = 10;
30     double start = omp_get_wtime();
31
32     //omp_set_dynamic(0);
33     omp_set_num_threads(THREAD_COUNT);
34     #pragma omp parallel shared(n)
35     {
36         #pragma omp single
37         printf("fib(%d) = %d\n", n, fib(n));
38     }
39     double end = omp_get_wtime();
40     double total_time = end - start;
41     printf("\nExecution Time : %lf\n", total_time);
42     return 0;
43 }
44
```

Screenshot 2:

```
C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-4>g++ -fopenmp Que1_Fibonacci.cpp
C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-4>a.exe
fib(10) = 55
Execution Time : 0.009000
```

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.

Problem Statement 2:

Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements.

Screenshot 1:

```
//Producer Consumer Problem
//Author : Aditya Manapure
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()
{
    --mutex;

    ++full;

    --empty;

    x++;
    printf("\nProducer produces"
           "item %d",
           x);
    ++mutex;
}
```

Screenshot 2:

```
28 void consumer()
29 {
30     --mutex;
31
32     --full;
33
34     ++empty;
35     printf("\nConsumer consumes "
36           "item %d",
37           x);
38     x--;
39
40     ++mutex;
41 }
```

Screenshot 3:

```
42
43 ∨ int main()
44 {
45     int n, i;
46     printf("\n1. Press 1 for Producer"
47           "\n2. Press 2 for Consumer"
48           "\n3. Press 3 for Exit");
49
50     #pragma omp critical
51
52     for (i = 1; i > 0; i++) {
53
54         printf("\nEnter your choice:");
55         scanf("%d", &n);
56
57         switch (n) {
58             case 1:
59                 if ((mutex == 1)
60                     && (empty != 0)) {
61                     producer();
62                 }
63
64                 else {
65                     printf("Buffer is full!");
66                 }
67                 break;
68
69             case 2:
70                 if ((mutex == 1)
71                     && (full != 0)) {
72                     consumer();
73                 }
74                 else {
75                     printf("Buffer is empty!");
76                 }
77                 break;
78
79             case 3:
80                 exit(0);
81                 break;
82         }
83     }
```

Output :

```
C:\Users\aditya\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-4>g++ -fopenmp Que2_ProducerConsumer.cpp
C:\Users\aditya\OneDrive\Documents\College\sem 7\HPC\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:2

Consumer consumes item 1
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:2

Consumer consumes item 3
Enter your choice:2
```

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.

Github link :

https://github.com/Aditya-Manapure/2019BTECS00055_HPC_LAB/tree/main/Assignment-4