PDC - Lab 3

Aadhitya Swarnesh I

[redacted]

1 - August - 2020

Question 1 :

To demonstrate reduction and barriers using the problem of multiplying two matrices

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

int multiply_parallel(int m, int n, int p, int a[][5],
int b[][5], int c[][5])
{
    int i,j,k,sum;
    #pragma omp parallel shared(a,b,c) private(i,j,k)
    {
        printf("Before Barrier construct - thread no =
%d\n", omp_get_thread_num());
        for (i=0; i<m; i=i+1)
        {
            #pragma omp reduction(+: sum)
            for (j=0; j<n; j=j+1)
            {
                sum=0.;
                for (k=0; k<p; k=k+1)
```

```
                {
                    sum=(a[i][j])+((b[i][k])*(c[k][j]));
                }
                a[i][j] = sum;
            }
        }
        // No barrier is placed here.
        printf("After Barrier construct - thread no =
%d\n", omp_get_thread_num());
        // For a task after the matrix multiplication.
   }
   return 0;
}


int multiply_parallel_barrier(int m, int n, int p, int
a[][5], int b[][5], int c[][5])
{
    int i,j,k,sum;
    #pragma omp parallel shared(a,b,c) private(i,j,k)
    {
        printf("Before Barrier construct - thread no =
%d\n", omp_get_thread_num());
        for (i=0; i<m; i=i+1)
        {
            #pragma omp reduction(+: sum)
            for (j=0; j<n; j=j+1)
            {
                sum=0.;
                for (k=0; k<p; k=k+1)
                {
                    sum=(a[i][j])+((b[i][k])*(c[k][j]));
                }
                a[i][j] = sum;
```

```
            }
        }
        #pragma omp barrier
        printf("After Barrier construct - thread no =
%d\n", omp_get_thread_num());
        // For a task after the matrix multiplication.
   }
   return 0;
}


int main()
{
    int m, n, p, i, j;
    m = 5;
    n = 5;
    p = 5;

    int a[5][5] = {{1, 2, 3, 4, 600}, {3, 4, 600, 6, 7},
{1, 2, 3, 4, 600}, {3, 4, 600, 6, 7}, {1, 2, 3, 4, 600}};
// Dimensions m x n
    int b[5][5] = {{600, 6, 7, 8, 9}, {7, 8, 9, 600, 3},
{1, 2, 3, 4, 600}, {3, 4, 600, 6, 7}, {1, 2, 3, 4, 600}};
// Dimensions n x p
    int  parallel_mul[5][5];

    // Parallel Multiplication without barrier
    printf("Implementation without barrier :\n");
    multiply_parallel(m, n, p, parallel_mul, a, b);
    printf("\n\n");

    // Parallel Multiplication with barrier for after
multiplication task.
    printf("Implementation with barrier :\n");
```

```
    multiply_parallel_barrier(m, n, p, parallel_mul, a,
b);
}
```

```
(base) Aadhityas-MacBook-Air:1Aug2020 aadhitya$ gcc-10 -fopenmp p4.c
(base) Aadhityas-MacBook-Air:1Aug2020 aadhitya$ ./a.out
Implementation without barrier :
Before Barrier construct - thread no = 0
After Barrier construct - thread no = 0
Before Barrier construct - thread no = 1
Before Barrier construct - thread no = 3
After Barrier construct - thread no = 3
After Barrier construct - thread no = 1
Before Barrier construct - thread no = 2
After Barrier construct - thread no = 2


Implementation with barrier :
Before Barrier construct - thread no = 1
Before Barrier construct - thread no = 2
Before Barrier construct - thread no = 3
Before Barrier construct - thread no = 0
After Barrier construct - thread no = 2
After Barrier construct - thread no = 1
After Barrier construct - thread no = 3
After Barrier construct - thread no = 0
(base) Aadhityas-MacBook-Air:1Aug2020 aadhitya$
```

Here we have used both the concepts of reductions and barriers, we will now justify its needs.

Reduction is used so that to reduce the number of atomic operations that will cost a loss in time and thereby reduce the performance. So with this a[i][j] will be approached only num_thread times.

Barriers are used as a synchronisation tool. We can see the difference in the output above. If we use barriers all the threads wait at the barrier and proceed only when all the threads have reached the barrier. Here we can use the barrier so that if we use the `a` array, then it will be filled completely and ready for further access thereby preventing race conditions.