PDC - Lab 2

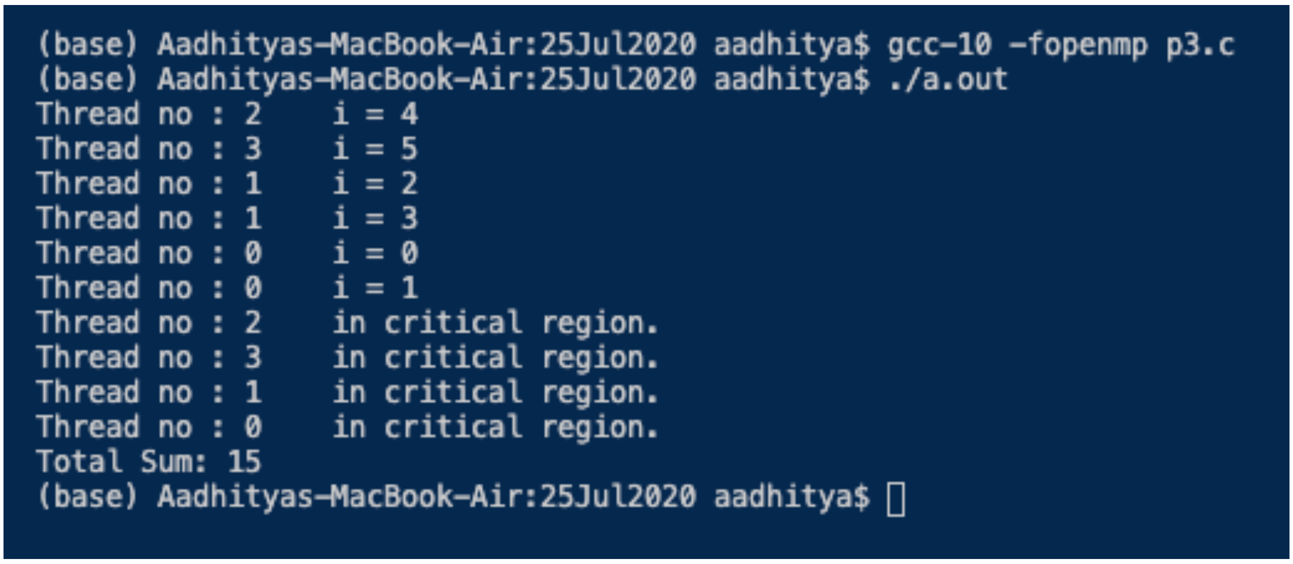
Aadhitya Swarnesh I
[redacted]

25 - July - 2020

Question 1 :

To find the sum of n numbers of an array.

```
#include <stdio.h>
#include <omp.h>
int main()
{
    int cursum, total,n;
    int a[5]={1,2,3,4,5};
    n = sizeof(a)/sizeof(int);

    #pragma omp parallel private(cursum) shared(total)
    {
        cursum = 0;
        total = 0;
        #pragma omp for
            for(int i = 0; i <= n; i++)
            {
                cursum += a[i];
                printf("Thread no : %d\t i = %d\n", 
omp_get_thread_num(), i);
            }
        #pragma omp critical
        {
            total += cursum;
            printf("Thread no : %d\t in critical region.
\n", omp_get_thread_num());
        }
    }
    printf("Total Sum: %d\n", total);
    return 0;
}
```



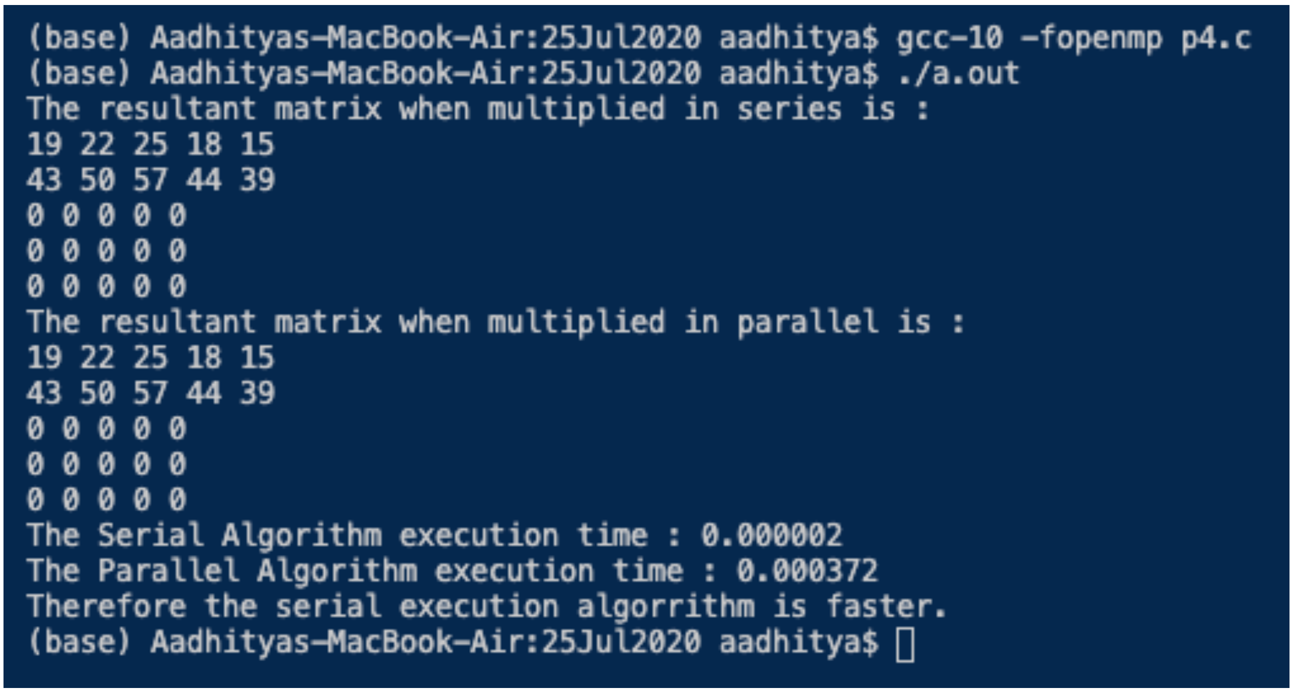
```
(base) Aadhityas-MacBook-Air:25Jul2020 aadhitya$ gcc-10 -fopenmp p3.c
(base) Aadhityas-MacBook-Air:25Jul2020 aadhitya$ ./a.out
Thread no : 2    i = 4
Thread no : 3    i = 5
Thread no : 1    i = 2
Thread no : 1    i = 3
Thread no : 0    i = 0
Thread no : 0    i = 1
Thread no : 2    in critical region.
Thread no : 3    in critical region.
Thread no : 1    in critical region.
Thread no : 0    in critical region.
Total Sum: 15
(base) Aadhityas-MacBook-Air:25Jul2020 aadhitya$
```

Question 2 :

To perform matrix multiplication using serial and parallel programming and to compare the performance by measuring their time of execution.

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


int multiply_parallel(int m, int n, int p, int a[][5],
int b[][5], int c[][5])
{
    int i,j,k;
    #pragma omp parallel shared(a,b,c) private(i,j,k)
    {
        #pragma omp for  schedule(static)
        for (i=0; i<m; i=i+1)
        {
            for (j=0; j<n; j=j+1)
            {
                a[i][j]=0.;
                for (k=0; k<p; k=k+1)
                {
                    a[i][j]=(a[i][j])+((b[i][k])*(c[k]
[j]));
                }
            }
        }
   }
   return 0;
}
```

```
int multiply_serial(int m, int n, int p, int a[5][5], int
b[][5], int c[][5])
{
    int i, j, k ;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            a[i][j]=0;
            for (k=0; k<p; k++)
            {
                a[i][j] += b[i][k]*c[k][j] ;
            }
        }
    }
    return 0;
}


int main()
{
    clock_t start, end;
    double serial_time, parallel_time;

    int m, n, p;
    m = 5;
    n = 5;
    p = 5;
    int a[5][5] = {{1, 2, 3, 4, 5}, {3, 4, 5, 6, 7}};
// Dimensions m x n
    int b[5][5] = {{5, 6, 7, 8, 9}, {7, 8, 9, 5, 3}};
// Dimensions n x p
    int serial_mul[5][5], parallel_mul[5][5];
    // Series Multiplication
    start = clock();
    multiply_serial(m, n, p, serial_mul, a, b);
    end = clock();
    serial_time = ((double) (end - start)) /
CLOCKS_PER_SEC;

    // Parallel Multiplication
    start = clock();
    multiply_parallel(m, n, p, parallel_mul, a, b);
    end = clock();
    parallel_time = ((double) (end - start)) /
CLOCKS_PER_SEC;

    // Printing the matrices of dimensions m x p :
    int i, j;

    // Series Multiplication
    printf("The resultant matrix when multiplied in
series is :\n");
    for (i=0;i<m;i++)
    {
        for (j=0;j<p;j++)
        {
            printf("%d ", serial_mul[i][j]);
        }
        printf("\n");
    }

    // Parallel Multiplication
    printf("The resultant matrix when multiplied in
parallel is :\n");
    for (i=0;i<m;i++)
    {
        for (j=0;j<p;j++)
        {
            printf("%d ", parallel_mul[i][j]);
        }
        printf("\n");
    }

    // Print the performance measures.

    printf("The Serial Algorithm execution time : %f\n", 
serial_time);
    printf("The Parallel Algorithm execution time : 
%f\n", parallel_time);

    if (serial_time > parallel_time)
    {
        printf("Therefore the parallel execution 
algorrithm is faster.\n");
    }
    else if (serial_time < parallel_time)
    {
        printf("Therefore the serial execution algorrithm 
is faster.\n");
    }
    else
    {
        printf("Both the algorithms perform equally.\n");
    }
}
```



```
(base) Aadhityas-MacBook-Air:25Jul2020 aadhitya$ gcc-10 -fopenmp p4.c
(base) Aadhityas-MacBook-Air:25Jul2020 aadhitya$ ./a.out
The resultant matrix when multiplied in series is :
19 22 25 18 15
43 50 57 44 39
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
The resultant matrix when multiplied in parallel is :
19 22 25 18 15
43 50 57 44 39
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
The Serial Algorithm execution time : 0.000002
The Parallel Algorithm execution time : 0.000372
Therefore the serial execution algorrithm is faster.
(base) Aadhityas-MacBook-Air:25Jul2020 aadhitya$ 
```

Note : As we have used arrays of very small dimensions, the series programming is faster, if we move to a higher dimensional matrix say of dimensions about [1000 X 1000], then parallel programming paradigm would serve as a better performance booster to solve the problem.

Note : We can also increase the number of threads by changing the environment variable by using the command :

```
export OMP_NUM_THREADS=<number of threads to use>
```

Here we can specify any number of threads needed to better handle the problem in hand.