**Class:** Final Year (Computer Science and Engineering)

**Year:** 2023-24 **Semester:** 1

**Course:** High Performance Computing Lab

# Practical No. 5

**Exam Seat No: 2020BTECS00022**

# Title of practical: Implementation of OpenMP programs.

Implement following Programs using OpenMP with C:

1. Implementation of sum of two lower triangular matrices.
2. Implementation of Matrix-Matrix Multiplication.

# Problem Statement 1: Implementation of sum of two lower triangular matrices. Code:

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

int main(){

int dimention;

printf("Enter dimention for matrix = "); scanf("%d",&dimention);

//Serial Code

int m1[dimention][dimention]; int count=0; printf("\nSerial\n");

double start\_time\_serial = omp\_get\_wtime(); for(int i=0;i<dimention;i++){

for(int j=0;j<dimention;j++){ m1[i][j]=++count; printf("%d\t",m1[i][j]);

}

printf("\n");

}

int sum=0;

for(int i=0;i<dimention;i++){ for(int j=0;j<dimention;j++){

if(i>j) sum+=m1[i][j];

}

}

int sum2=0;

for(int i=0;i<dimention;i++){ for(int j=0;j<dimention;j++){

if(j<dimention-i) continue; else sum2+=m1[i][j];

}

}

double end\_time\_serial = omp\_get\_wtime(); printf("\nLeft Lower Triangle Sum = %d",sum); printf("\nRight Lower Triangle Sum = %d",sum2); printf("\nTwo Lower Triangles Sum = %d",(sum+sum2));

//Parallel code count=0; printf("\nParallel\n");

double start\_time\_parallel = omp\_get\_wtime(); #pragma omp parallel for ordered num\_threads(8) for(int i=0;i<dimention;i++){

for(int j=0;j<dimention;j++){ m1[i][j]=++count; printf("%d\t",m1[i][j]);

}

printf("\n");

}

sum=0;

#pragma omp parallel for num\_threads(8) for(int i=0;i<dimention;i++){

for(int j=0;j<dimention;j++){ if(i>j) sum+=m1[i][j];

}

}

sum2=0;

#pragma omp parallel for num\_threads(8) for(int i=0;i<dimention;i++){

for(int j=0;j<dimention;j++){ if(j<dimention-i) continue;

else

sum2+=m1[i][j];

}

}

double end\_time\_parallel = omp\_get\_wtime();

printf("\nLeft Lower Triangle Sum = %d",sum); printf("\nRight Lower Triangle Sum = %d",sum2); printf("\nTwo Lower Triangles Sum = %d",(sum+sum2));

printf("\n\nSerial Method Time: %f seconds\n", (end\_time\_serial - start\_time\_serial));

printf("\nParallel Method Time: %f seconds\n\n", (end\_time\_parallel - start\_time\_parallel));

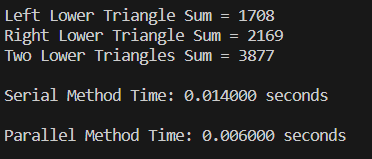
return 0;

}

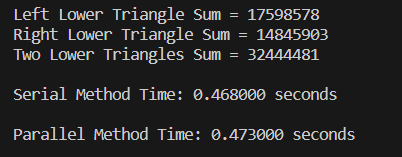
# Screenshots:

Keeping number of threads constant and varying size of Data.

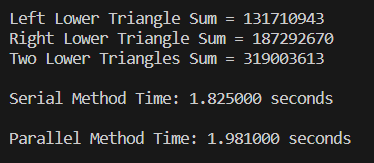
Threads = 8, Matrix size = 10



Threads = 8, Matrix size = 100

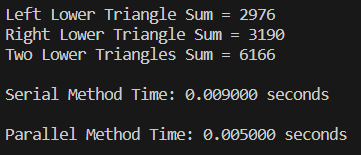


Threads = 8, Matrix Size = 200

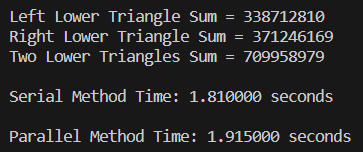


Keeping data constant and increasing number of threads.

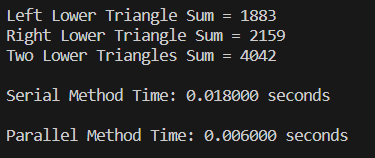
Threads = 2, Matrix size = 10



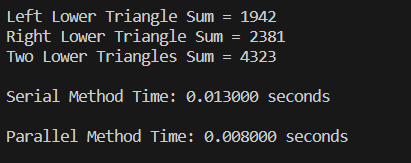
Threads = 2, Matrix size = 200



Threads = 4, Matrix size = 10



Threads = 10, Matrix size = 10



# Information:

It is observed that keeping threads constant i.e., 8 we will get same difference in serial and parallel time, there is no affect of change in data size. On other hand by varying the number threads to appropriate amount will give significantly reduce in time of parallel execution.

# Analysis:



**Analysis**

1.2

1

0.8

0.6

Matrix size - 10 Matrix size - 100

Matrix size - 200

0.4

0.2

0

Threads 2

Threads 4

Threads 8

Threads 10

**Speedup**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Threads | Data Size | Sequential Time(sec)(Ts) | Parallel Time(sec)(Tp) | Speedup(Ts/Tp) |
| 8 | 10 | 0.001400 | 0.006000 | 0.233333 |
| 8 | 100 | 0.468000 | 0.473000 | 0.989429 |
| 8 | 200 | 1.825000 | 1.981000 | 0.921252 |
| 2 | 10 | 0.001400 | 0.005000 | 0.280000 |
| 2 | 100 | 0.468000 | 0.450000 | 1.040000 |
| 2 | 200 | 1.825000 | 1.915000 | 0.953003 |
| 4 | 10 | 0.001400 | 0.006000 | 0.233333 |
| 4 | 100 | 0.468000 | 0.479000 | 0.977035 |
| 4 | 200 | 1.825000 | 1.894000 | 0.963569 |
| 10 | 10 | 0.001400 | 0.008000 | 0.175000 |
| 10 | 100 | 0.468000 | 0.485000 | 0.964948 |
| 10 | 200 | 1.825000 | 1.858000 | 0.982239 |

**Problem Statement 2: Implementation of Matrix-Matrix Multiplication. Code:**

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

int main()

{

int i,j,k,N; printf("Enter size = "); scanf("%d",&N);

int A[N][N];

int B[N][N];

int C[N][N];

clock\_t st = clock(); for (i= 0; i< N; i++)

{

for (j= 0; j< N; j++)

{

A[i][j] = i+j;

B[i][j] = i+j;

C[i][j] = 0;

}

}

printf("\nMatrix\n"); for (i= 0; i< N; i++)

{

for (j= 0; j< N; j++)

{

printf("%d\t",A[i][j]);

}

printf("\n");

}

#pragma omp parallel for private(i,j,k) shared(A,B,C) num\_threads(4) for (i = 0; i < N; i++)

{

for (j = 0; j < N; j++)

{

for (k = 0; k < N; k++)

{

C[i][j] += A[i][k] \* B[k][j];

}

}

}

printf("\nAnswer\n"); for (i = 0; i < N; i++)

{

for (j = 0; j < N; j++)

{

printf("%d\t",C[i][j]);

}

printf("\n");

}

clock\_t et = clock();

double elapsed\_time = (double)(et - st) / CLOCKS\_PER\_SEC; double elapsed\_miliseconds = elapsed\_time \* 1000; printf("\nTime taken: %f miliseconds", elapsed\_miliseconds); printf("\nTime taken: %f seconds\n", elapsed\_time);

}

# Screenshots:

Keeping number of threads constant and varying size of Data.

Threads = 4, Matrix size = 10



Threads = 4, Matrix size = 100



Threads = 4, Matrix size = 400



Keeping data constant and increasing number of threads.

Threads = 2, Matrix size = 100



Threads = 8, Matrix size = 100



**Analysis**

4.5

4

3.5

3

2.5

2

1.5

1

0.5

0

Matrix size - 10 Matrix size - 100

Matrix size - 400

Threads 2 Threads 4 Threads 8

**Speedup**



Threads = 20, Matrix size = 100



# Information:

Problem is calculating the matrix multiplication by varying the size of the matrix. It is noted that as we increase the size of the matrix the time required for sequential execution is more while parallel execution complete the same task in lesser time. Even when we increase the number of threads, more execution speed is achieved by parallelism.

# Analysis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Threads | Data Size | Sequential Time(sec)(Ts) | Parallel Time(sec)( Tp) | Speedup(Ts/Tp) |
| 4 | 10 | 0.057000 | 0.025000 | 2.280000 |
| 4 | 100 | 2.790000 | 2.374000 | 1.175232 |
| 4 | 400 | 61.349000 | 18.446000 | 3.325870 |
| 2 | 10 | 0.057000 | 0.016000 | 3.562500 |
| 2 | 100 | 2.790000 | 1.140000 | 2.447368 |
| 2 | 400 | 61.349000 | 14.753000 | 4.158408 |
| 8 | 10 | 0.057000 | 0.017000 | 3.352941 |
| 8 | 100 | 2.790000 | 1.175000 | 2.374468 |
| 8 | 400 | 61.349000 | 14.434000 | 4.250312 |
| 20 | 100 | 2.790000 | 1.177000 | 2.370433 |

GITHUB LINK :- [**https://github.com/GauravP07/HPC\_Assignments**](https://github.com/GauravP07/HPC_Assignments)