Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

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

Course: High Performance Computing Lab

Practical No. 5

Exam Seat No: 2020BTECS00038

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.
Screenshots:

```
#include <iostream>
#include <omp.h>
#include <time.h>
#include <vector>
using namespace std;
const int N = 1000;

int main()
{
    omp_set_num_threads(8);

    clock_t start, end;
    start = clock();

    int matrix1[N][N] = {
        {1, 0, 0, 0, 0},
        {6, 7, 0, 0, 0},
        {11, 12, 13, 0, 0},
```

```
        {16, 17, 18, 19, 0},
        {21, 22, 23, 24, 25}};

    int matrix2[N][N] = {
        {5, 0, 0, 0, 0},
        {9, 8, 0, 0, 0},
        {15, 14, 10, 0, 0},
        {20, 19, 18, 11, 0},
        {25, 24, 23, 22, 21}};

    vector<vector<int>> result(N, vector<int>(N));
    // vector<vector<int>> matrix1(N, vector<int>(N));
    // vector<vector<int>> matrix2(N, vector<int>(N));

#pragma omp parallel for schedule(dynamic, 20)
    for (int i = 0; i < N; i++)
    {
        if(i>=0 && i<40)
         cout << i << " " <<omp_get_thread_num()<<" "<< endl;
        for (int j = 0; j <= i; j++)
        {
            result[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }

    end = clock();
    double duration = ((double)end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken to execute in seconds : %f", duration);

    return 0;
}
```

Output:

```
PS D:\Sem7\HPC_Lab\Assignment5> .\a.exe

Time taken to execute in seconds : 1.482000
PS D:\Sem7\HPC_Lab\Assignment5> cd "d:\Sem7\HPC_Lab\Assignment5\" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }

Time taken to execute in seconds : 2.087000
```

Information & Analysis:

Here in the problem of addition of two lower triangular matrices, the upper triangular part of the matrices is with zero entries so, there is no use to add them. If we directly parallelize the code with static scheduling and using parallel for construct only, the threads having the addition of zero entries will remain idle for some time. For avoiding idling of the processors, we use dynamic scheduling here.

We can clearly see that execution time in case of parallel program is much less than serial code due to dynamic scheduling and also as our input size is increasing the execution time increases gradually.

Number of Threads	Data Size	Sequential Time	Parallel Time
4	120	0.00100	0.00100
8	120	0.00100	0.00100
4	1200	0.05000	0.01700
8	1200	0.05000	0.02500
4	12000	2.34600	1.56200
8	12000	2.34600	1.45200

Problem Statement 2: Implementation of Matrix-Matrix multiplication

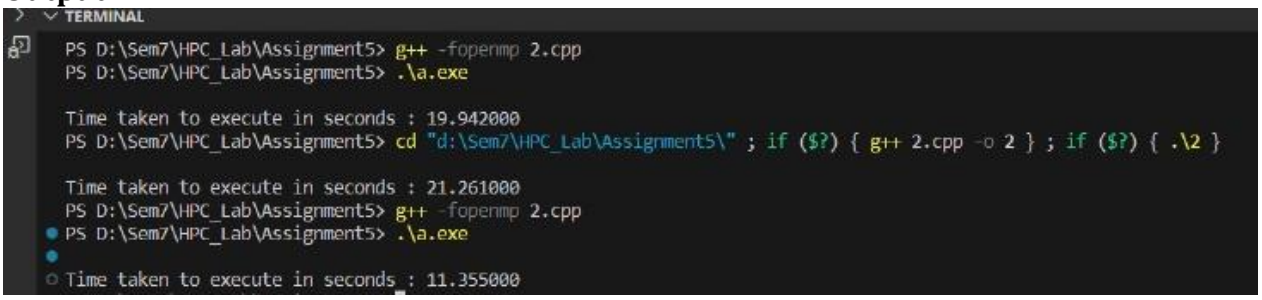
Screenshots:

```cpp
#include <iostream>
#include <vector>
#include <omp.h>
#include <time.h>
using namespace std;
const int N = 1000;

int main()
{
    clock_t start, end;
    start = clock();
    vector<vector<int>> matrix1(N, vector<int>(N, 1));
    vector<vector<int>> matrix2(N, vector<int>(N, 2));
    vector<vector<int>> result(N, vector<int>(N, 0));
```

```
#pragma omp parallel for num_threads(8) schedule(static, 250)
    for (int i = 0; i < N; i++)
    {
#pragma omp parallel for schedule(static, 250)
        for (int j = 0; j < N; j++)
        {
            for (int k = 0; k < N; k++)
            {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
    end = clock();
    double duration = ((double)end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken to execute in seconds : %f", duration);
    return 0;
}
```

Output:



Information & Analysis:

In matrix-matrix multiplication problem, if we execute the serial program it takes the O(n^3) time as it has three nested loops to run. If the outermost loop is executed parallelly, the time taken is little less and again we want to reduce the time the inner loop is also executed parallelly. Now we are giving one matrix row and one matrix column to the processor, and not the whole matrix.

Number of Threads	Data Size	Sequential Time	Parallel Time
8	10	0.00000	0.00200
4	100	0.01700	0.02800
8	100	0.01700	0.02100

4	1000	21.6350	11.0130
8	1000	21.6350	10.9110

Github Link: https://github.com/divyakekade/HPC-Assignments