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: 2020BTECS00041

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

Sequential Program:

```c
#include <stdio.h>
#include <omp.h>
#include <time.h>
#define N 500
void sumLowerTriangularMatrices(int A[][N], int B[][N], int C[][N])
{
    int j, i;
    for (i = 0; i < N; i++) {
        for (j = 0; j <= i; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}

int main()
{
    clock_t start=clock();
    int A[N][N];
    int B[N][N];
    int C[N][N]={0};
```

```
    int value = 40000;
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j <= i; j++)
            A[i][j] = value;
    }
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j <= i; j++)
            B[i][j] = value;
    }
    sumLowerTriangularMatrices(A, B, C);
    clock_t end=clock();
    printf("\nSize of Matrix: %d x %d", N, N);
    printf("    Time taken: %f\n\n",(double)(end-start)/CLOCKS_PER_SEC);
    return 0;
}
```

Output of sequential:

```
PS D:\FinalYear\HPCL\Assignment 5> gcc q1sequential.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Size of Matrix: 300 x 300    Time taken: 0.000000
```

Parallel Program:

```
#include <stdio.h>
#include <omp.h>
#include <time.h>
#define N 300
#define chunk_size 10
#define threads 1
void sumLowerTriangularMatrices(int A[][N], int B[][N], int C[][N])
{
    int j,i;
    #pragma omp parallel for private(i, j) schedule(static, chunk_size)
num_threads(threads)
    // #pragma omp parallel for private(i, j) schedule(dynamic, chunk
size) num_threads(threads)
    for (i = 0; i < N; i++)
    {
        for (j = 0; j <= i; j++)
        {
```

```
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}

int main()
{
    clock_t start=clock();
    int A[N][N];
    int B[N][N];
    int C[N][N]={0};
    int value = 40000;
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j <= i; j++)
            A[i][j] = value;
    }
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j <= i; j++)
            B[i][j] = value;
    }
    sumLowerTriangularMatrices(A, B, C);
    clock_t end=clock();
    printf("Time taken: %f\n",(double)(end-start)/CLOCKS_PER_SEC);
    return 0;
}
```

Output for Parallel Program:

Static

```
Chunk Size: 10    Number of Threads: 1    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 1    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 1    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 1    Time taken: 0.000000
```

```
Chunk Size: 10    Number of Threads: 2    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 2    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 2    Time taken: 0.002000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 2    Time taken: 0.000000
```

```
Chunk Size: 10    Number of Threads: 4    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 4    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 4    Time taken: 0.004000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 4    Time taken: 0.000000
```

```
Chunk Size: 10    Number of Threads: 8    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 8    Time taken: 0.015000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 8    Time taken: 0.008000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 8    Time taken: 0.000000
```

Dynamic Schedule:

```
Chunk Size: 10    Number of Threads: 1    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 1    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 1    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 1    Time taken: 0.000000
```

```
Chunk Size: 10    Number of Threads: 2    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 2    Time taken: 0.008000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 2    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 2    Time taken: 0.000000
```

```
Chunk Size: 10    Number of Threads: 4    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 4    Time taken: 0.002000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 4    Time taken: 0.010000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 4    Time taken: 0.000000
```

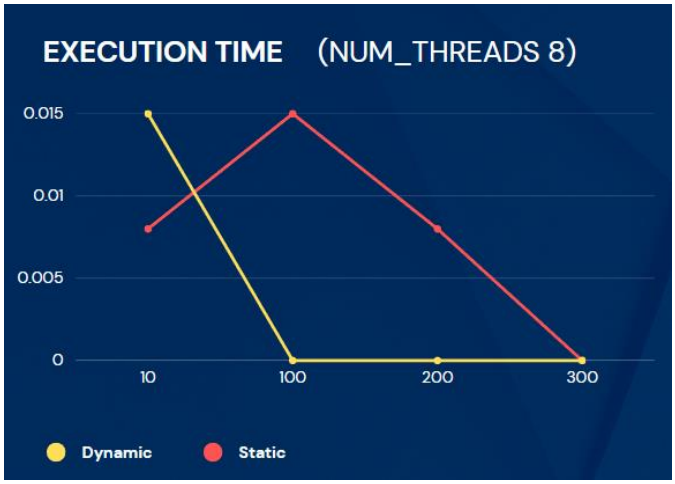
```
Chunk Size: 10    Number of Threads: 8    Time taken: 0.015000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 100    Number of Threads: 8    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 200    Number of Threads: 8    Time taken: 0.000000

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Chunk Size: 300    Number of Threads: 8    Time taken: 0.000000
```

Execution Time for Static schedule:

Chunk size\threads	1	2	4	8
10	0.000	0.000	0.000	0.000
100	0.000	0.000	0.000	0.015
200	0.000	0.002	0.004	0.008
300	0.000	0.000	0.000	0.000

Execution Time for Dynamic Schedule:

Chunk size\threads	1	2	4	8
10	0.000	0.000	0.000	0.015
100	0.000	0.008	0.002	0.000
200	0.000	0.000	0.010	0.000
300	0.000	0.000	0.000	0.000



Sequential program of sum of two lower triangular matrices for size 300 x 300 has the execution time 0.000
Hence speed up will be 0.

Information:

A square matrix in which all the elements above the principal diagonal are zero is called a lower triangular matrix. The addition of two lower triangular matrix requires 2 for loops. The parallel programming of the problem is done using private clause, static or dynamic schedules and defining number of threads. The private clause creates different copies of variables listed. It helps to maintain consistency between different threads. The schedule clause describes how iterations of the loop are divided among the threads in group. The num_threads() is used to specify number of threads to be executed parallelly.

Analysis:

The performance of the program depends on the use of different clauses. The dynamic schedule clause is faster than the static schedule clause. The execution speed is directly proportional to the number of threads executed parallelly. A private variable has a different address in the execution context of every thread. Variables in private context are hidden from other threads. Each thread has its own private copy of the variable, and modifications made by a thread to its copy are not visible to other threads.

Problem Statement 2: Implementation of Matrix-Matrix Multiplication.

Sequential Program:

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define N 300

int A[N][N];
int B[N][N];
int C[N][N];

int main()
{
    clock_t start=clock();
    int i,j,k;
    for (i= 0; i< N; i++)
    for (j= 0; j< N; j++)
    {
        A[i][j] = 2;
        B[i][j] = 2;
    }
    int sum = 0, mul = 0;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            for (k = 0; k < N; k++)
            {
                mul = A[i][k] * B[k][j];
                sum += mul;
            }
            C[i][j] = sum;
            sum = 0;
        }
    }
    clock_t end=clock();
    printf("Time = %f seconds.\n", (double)(end-start)/CLOCKS_PER_SEC);
    return 0;
}
```

Output of Sequential:

```
PS D:\FinalYear\HPCL\Assignment 5> gcc q2sequential.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe
Time = 0.072000 seconds.
```

Parallel Program:

```
#include <pthread.h>
#include <stdio.h>
#include <omp.h>
#include <time.h>
#define N 300
int A[N][N], B[N][N], C[N][N];
int main() {
    clock_t start=clock();
    int i,j,k;
    for (i= 0; i< N; i++)
    for (j= 0; j< N; j++){
        A[i][j] = 2;
        B[i][j] = 2;
    }
    int sum = 0, mul = 0;
//#pragma omp parallel for private(i,j,k) shared(A,B,C) reduction(+:sum)
//#pragma omp parallel for private(i,j,k) shared(A,B,C) ordered
reduction(+:sum)
    #pragma omp for collapse(2)
    for (i = 0; i < N; ++i)
    {
        for (j = 0; j < N; ++j) {
            for (k = 0; k < N; ++k) {
                mul = A[i][k] * B[k][j];
                sum += mul;
            }
            C[i][j] = sum;
            sum = 0;
        }
    }
    clock_t end=clock();
    // printf("\nUsing reduction clause");
    // printf("\nUsing ordered reduction clause");
    printf("\nUsing collapse clause");
    printf("Time = %f sec\n\n", (double)(end-start)/CLOCKS_PER_SEC);
    return 0;
}
```

Screenshots:

```
Using reduction clauseTime = 0.120000 sec

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q2.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Using ordered reduction clauseTime = 0.122000 sec

PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q2.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe

Using collapse clauseTime = 0.078000 sec
```

Execution Time for:

1. Reduction clause = 0.120 sec
2. Ordered Reduction clause = 0.122 sec
3. Collapse clause = 0.078 sec

Speed Up for:

1. Reduction clause = 1.667
2. Ordered Reduction clause = 1.694
3. Collapse clause = 1.08

Information:

Different openmp clauses are used to parallelize the program. **Private** clause creates different copies of variables listed. It helps to maintain consistency between different threads. **Schedule** clause describes how iterations of loop are divided among the threads in group. **Reduction** clause specifies how to combine local copies of a variable in different threads into a single copy at the master when threads exit. **Ordered** specifies that the iteration of the loop must be executed as they would be in serial program. The matrix multiplication program contains 3 nested for loops, thus collapse clause is used to specifies how many loops in a nested loop should be collapsed into one large iteration space and divided according to the schedule clause. **Sequential execution** of the iteration in all associated loops determines the order of the iterations in the collapsed iteration space.

Analysis:

1. The performance of the program depends on the use of different clauses.
2. The dynamic schedule clause is faster than the static schedule clause.
3. The execution speed is directly proportional to number of threads executed parallelly.
4. A private variable has a different address in execution context of every thread. Variables in private context are hidden from other threads. Each thread has its own private copy of the variable, and modifications made by a thread to its copy are not visible to other threads.
5. Reduction clause implies data privatization of reduction variable that removes race conditions by replacing accesses to original variable with accesses to per-thread private-copies

Github Link: https://github.com/Nikita226/HPCL/tree/main/A5