

**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

**Screenshots:**

```
void sumLowerTriangularMatrices(int A[][N], int B[][N], int C[][N]) {  
    clock_t start_time = clock();  
    #pragma omp parallel for  
    for (int i = 0; i < N; i++) {  
        for (int j = 0; j <= i; j++) {  
            C[i][j] = A[i][j] + B[i][j];  
        }  
    }  
    clock_t end_time = clock();  
    printf("Total time: %d in seconds\n", end_time - start_time);  
}
```

```
PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c  
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe  
Time taken: 0.015000  
Matrix C:  
11 0 0 0  
11 11 0 0  
11 11 11 0  
11 11 11 11
```

Using schedule(static)

```
void sumLowerTriangularMatrices(int A[][N], int B[][N], int C[][N]) {  
    int j,i;  
    #pragma omp parallel for private(i,j) schedule(static)  
    for (i = 0; i < N; i++) {  
        for (j = 0; j <= i; j++) {  
            C[i][j] = A[i][j] + B[i][j];  
        }  
    }  
}
```

```
PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c  
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe  
Time taken: 0.000000  
Matrix C:  
11 0 0 0  
11 11 0 0  
11 11 11 0  
11 11 11 11
```

Using Dynamic schedule with size of chunk 4 and number of threads 4.

```
void sumLowerTriangularMatrices(int A[][N], int B[][N], int C[][N]) {  
    int j,i;  
    #pragma omp parallel for private(i,j) schedule(dynamic,4) num_threads(4)  
    for (i = 0; i < N; i++) {  
        for (j = 0; j <= i; j++) {  
            C[i][j] = A[i][j] + B[i][j];  
        }  
    }  
}
```

```
PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c  
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe  
Time taken: 0.000000  
Matrix C:  
11 0 0 0  
11 11 0 0  
11 11 11 0  
11 11 11 11
```

Using Static schedule with size of chunk 4 and number of threads 4

```
void sumLowerTriangularMatrices(int A[][N], int B[][N], int C[][N]) {  
    int j,i;  
    #pragma omp parallel for private(i,j) schedule(static,4) num_threads(4)  
    for (i = 0; i < N; i++) {  
        for (j = 0; j <= i; j++) {  
            C[i][j] = A[i][j] + B[i][j];  
        }  
    }  
}
```

```
PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q1.c  
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe  
Time taken: 0.000000  
Matrix C:  
11 0 0 0  
11 11 0 0  
11 11 11 0  
11 11 11 11
```

### 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.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#define N 5
int A[N][N];
int B[N][N];
int C[N][N];
int main()
{
    int i,j,k;
    for (i= 0; i< N; i++)
        for (j= 0; j< N; j++)
        {
            A[i][j] = 2;
            B[i][j] = 2;
        }
    clock_t start=clock();
    #pragma omp parallel for private(i,j,k) shared(A,B,C)
    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];
            }
        }
    }
    clock_t end=clock();
    printf("elapsed time = %f seconds.\n", (double)(end-
start)/CLOCKS_PER_SEC);
    for (i= 0; i< N; i++)
    {
        for (j= 0; j< N; j++)
        {
            printf("%d\t",C[i][j]);
        }
        printf("\n");
    }
}
```

### Screenshots:

For n=5 with private and shared clause

```
PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q2.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe
elapsed time = 0.001000 seconds.
20      20      20      20      20
20      20      20      20      20
20      20      20      20      20
20      20      20      20      20
20      20      20      20      20
```

For n=8 using dynamic schedule and specifying number of threads

```
#pragma omp parallel for private(i,j,k) shared(A,B,C) schedule(dynamic, 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];
        }
    }
}
```

```
PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q2.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe
elapsed time = 0.000000 seconds.
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
```

For n=8 without dynamic schedule

```
#pragma omp parallel for private(i,j,k) shared(A,B,C) num_threads(8)
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];
        }
    }
}
```

```
PS D:\FinalYear\HPCL\Assignment 5> gcc -fopenmp a5q2.c
PS D:\FinalYear\HPCL\Assignment 5> .\a.exe
elapsed time = 0.000000 seconds.
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
32      32      32      32      32      32      32      32
```



Using ordered reduction clause when n=5

```
#pragma omp parallel for private(i,j,k) shared(A,B,C) ordered reduction(+:sum)
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;
    }
}
```

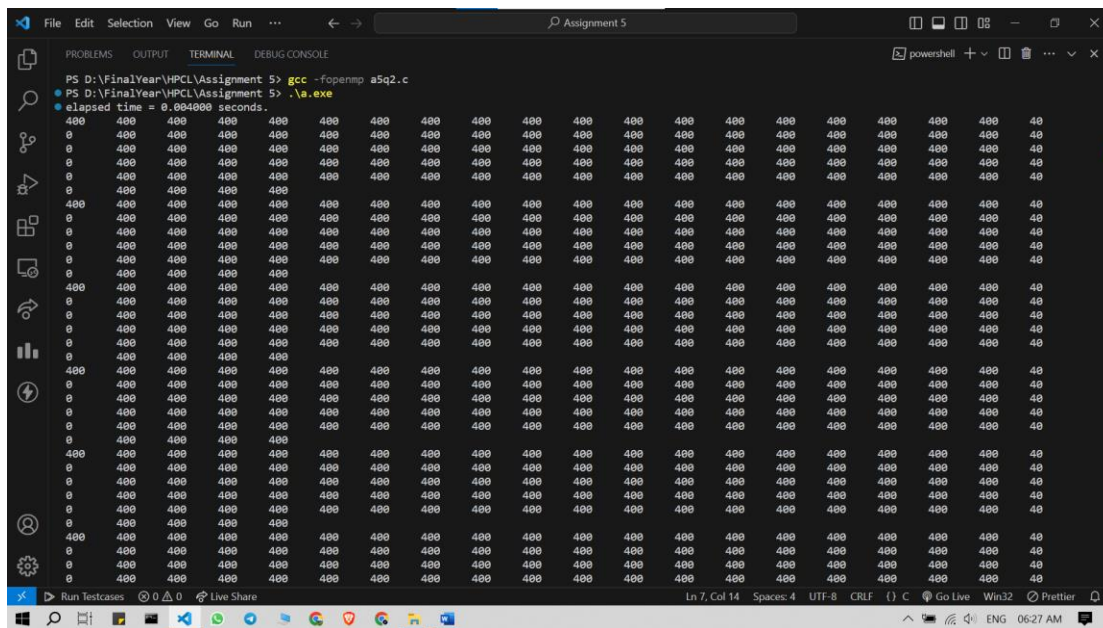
elapsed time = 0.002000 seconds.

20	20	20	20	20
20	20	20	20	20
20	20	20	20	20
20	20	20	20	20
20	20	20	20	20

Using reduction clause

```
clock_t start=clock();
int mul=0,sum=0;
#pragma omp parallel for private(i,j,k) shared(A,B,C) reduction (+:sum)
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] += A[i][k] * B[k][j];
        }
        C[i][j]=sum;
        sum=0;
    }
}
```





### Information:

Different openmp clauses are used to parallelize the program. 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. 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. The sequential execution of the iteration in all associated loops determines the order of the iterations in the collapsed iteration space.

### 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. The reduction clause implies data privatization of the reduction variable that removes race conditions by replacing accesses to the original variable with accesses to per-thread private copies

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