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. 3

Exam Seat No: 2020BTECS00041

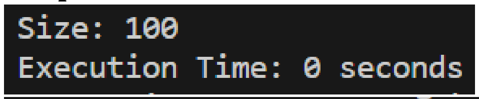
Title of practical:
Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

Problem Statement 1:
Analyse and implement a Parallel code for below program using OpenMP.
// C Program to find the minimum scalar product of two vectors (dot product)

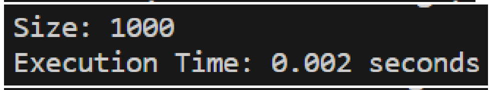
Sequential Program:

```
#include<bits/stdc++.h>
#include<iostream>
#include <omp.h>
#include <time.h>
using namespace std;
#define NUM_thread 100
#define N 100
int main()
{
    clock_t start_time = clock();
    vector<int>vector1(N),vector2(N);
    for(int i=0;i<N;i++)
    {
        vector1[i]=900;
        vector2[i]=900;
    }
    long long int product=0;
    int i;
    for (i = 0; i < N; i++)
    {
        product += vector1[i] * vector2[i];
    }
    cout << "\nScalar Product: " << product << endl;
    cout << "Num threads: " << NUM_thread;
    cout << "\nSize: " << N;
    clock_t end_time = clock();
    double elapsed_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;
    cout << "\nExecution Time: " << elapsed_time << " seconds\n";
    return 0;
}
```

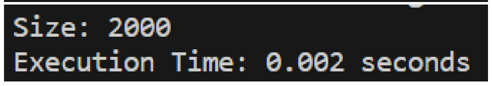
Output:



```
Size: 100
Execution Time: 0 seconds
```



```
Size: 1000
Execution Time: 0.002 seconds
```

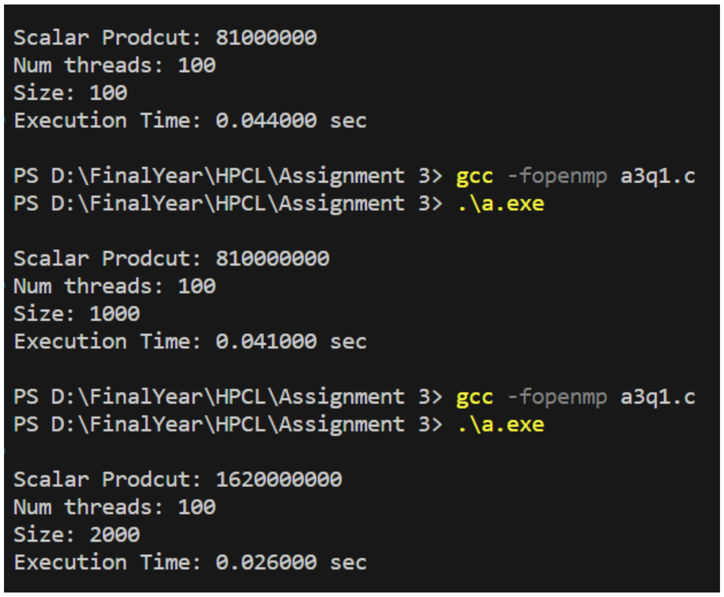


```
Size: 2000
Execution Time: 0.002 seconds
```

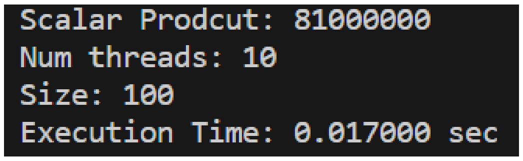
With Reduction, Nowait Clause:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#define NUM_thread 100
#define N 2000
// 10/100  100 1000 2000
int main()
{
    clock_t start_time = clock();
    int vector1[N], vector2[N];
    for(int i = 0; i < N; i++)
    {
        vector1[i] = 900;
        vector2[i] = 900;
    }
    int product=0;
    int i;
    // Parallel section
    // #pragma omp parallel for num_threads(NUM_thread) reduction(+ :
product)
    #pragma omp for nowait
    for ( i = 0; i < N; i++)
    {
        product += vector1[i] * vector2[i];
    }
    printf("\nScalar Prodcut: %d", product);
    printf("\nNum threads: %d", NUM_thread);
    printf("\nSize: %d", N);
    clock_t end_time = clock();
    double elapsed_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;
    printf("\nExecution Time: %f sec\n\n", elapsed_time);
    return 0;
}
```

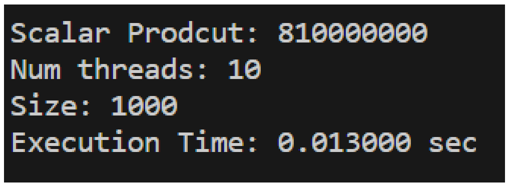
Screenshots:
Using reduction clause:



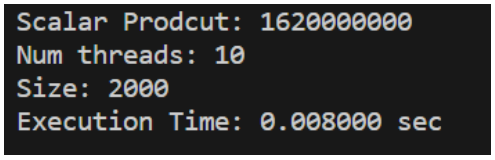
```
Scalar Prodcut: 81000000
Num threads: 100
Size: 100
Execution Time: 0.044000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q1.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Scalar Prodcut: 810000000
Num threads: 100
Size: 1000
Execution Time: 0.041000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q1.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Scalar Prodcut: 1620000000
Num threads: 100
Size: 2000
Execution Time: 0.026000 sec
```



```
Scalar Prodcut: 81000000
Num threads: 10
Size: 100
Execution Time: 0.017000 sec
```

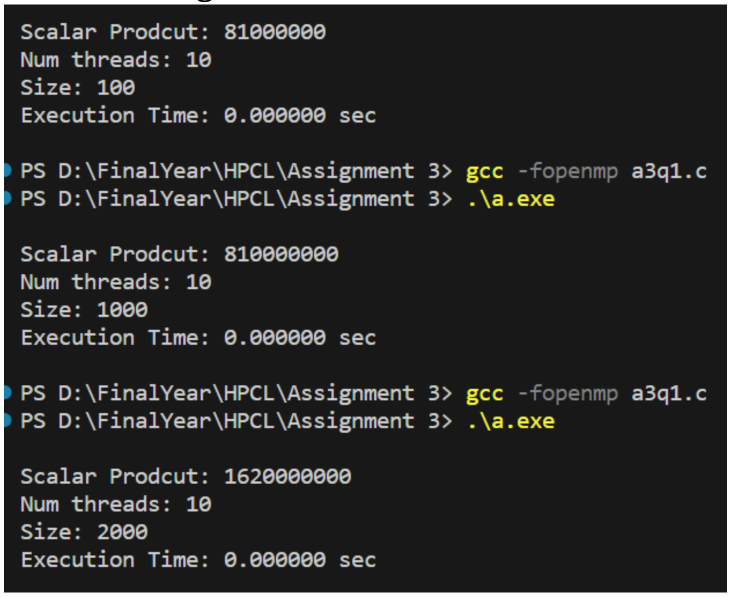


```
Scalar Prodcut: 810000000
Num threads: 10
Size: 1000
Execution Time: 0.013000 sec
```

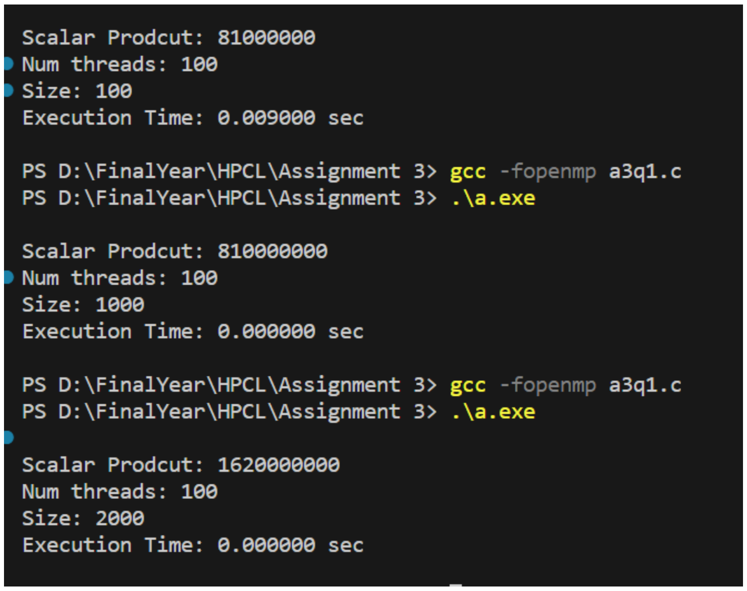


```
Scalar Prodcut: 1620000000
Num threads: 10
Size: 2000
Execution Time: 0.008000 sec
```

Using nowait clause:



```
Scalar Prodcut: 81000000
Num threads: 10
Size: 100
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q1.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Scalar Prodcut: 810000000
Num threads: 10
Size: 1000
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q1.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Scalar Prodcut: 1620000000
Num threads: 10
Size: 2000
Execution Time: 0.000000 sec
```



```
Scalar Prodcut: 81000000
Num threads: 100
Size: 100
Execution Time: 0.009000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q1.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Scalar Prodcut: 810000000
Num threads: 100
Size: 1000
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q1.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Scalar Prodcut: 1620000000
Num threads: 100
Size: 2000
Execution Time: 0.000000 sec
```

Analysis:

Sequential Execution Time:

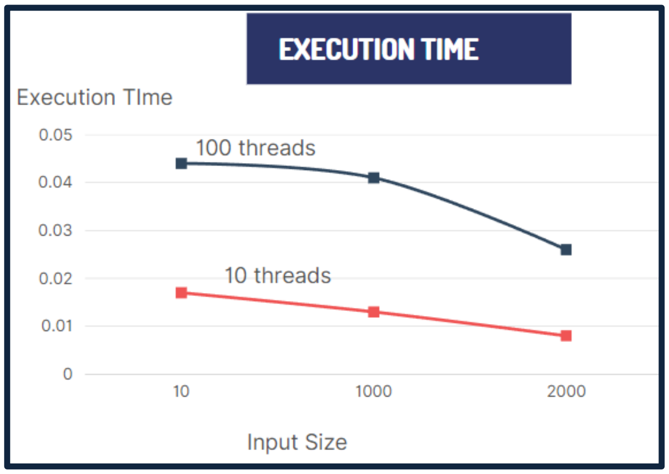
Input Size	Time
100	0
1000	0.002
2000	0.002

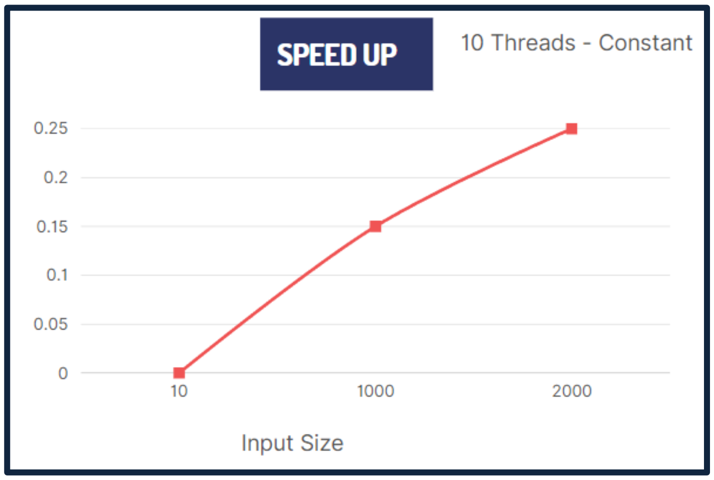
Parallel Execution Time:

Num of threads	Input Size	Reduction	Nowait
10	100	0.017	0.000
	1000	0.013	0.000
	2000	0.008	0.000
100	100	0.044	0.009
	1000	0.041	0.000
	2000	0.026	0.000

Speed up For reduction , threads 10:

Input Size	Speed Up
100	0
1000	0.15
2000	0.25





Information:

The reason for the code using the `reduction` clause taking more time for execution compared to the one with only the `nowait` clause is likely due to the nature of the work being performed and how it interacts with parallelism.

1. **Reduction Clause Overhead**: When you use the `reduction` clause in OpenMP, the compiler and runtime system need to perform additional operations to combine the partial results from each thread. This can introduce some overhead, especially for complex reduction operations. The `reduction` clause is intended for situations where you need to compute a reduction operation accurately, such as summing values, and it ensures the correct result at the end of the parallel region. This overhead can make the code slightly slower than a simple `nowait` clause.

2. **Parallelization Granularity**: The effectiveness of parallelization depends on the granularity of the parallel tasks. If the work done within the loop is relatively small compared to the overhead of thread creation and synchronization, the parallel version of the code might not show a significant speedup compared to the serial version. In such cases, a simple `nowait` clause may make the code faster because it avoids some of the synchronization overhead.
3. **Additional Work**: In the code with the `nowait` clause, there is "additional work" done after the loop. If this additional work is substantial and can be efficiently parallelized, it may offset the difference in execution time. The `nowait` clause allows the program to overlap this additional work with the loop execution, potentially making the overall execution time shorter.

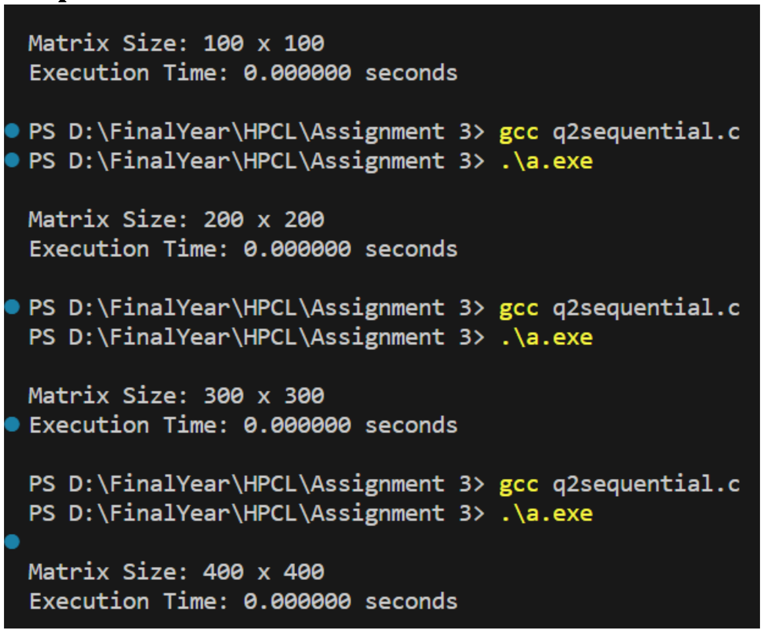
Problem Statement 2:

Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread
(Use functions in C in calculate the execution time or use GPROF)
i. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.
ii. Explain whether or not the scaling behaviour is as expected.

Sequential:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define size 400
int main()
{
    clock_t start_time = clock();
    int A[size][size];
    int B[size][size];
    // size of matrix {100, 200, 300, 400};
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
           A[i][j] = 100;
    }
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
           B[i][j] = 100;
    }
    int C[size][size];
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
            C[i][j] = A[i][j] + B[i][j];
    }
    printf("\nMatrix Size: %d x %d", size, size);
    clock_t end_time = clock();
    double execution_time =(double)( end_time -
start_time)/CLOCKS_PER_SEC;
    printf("\nExecution Time: %f seconds\n\n", execution_time);
    return 0;
}
```

Output:

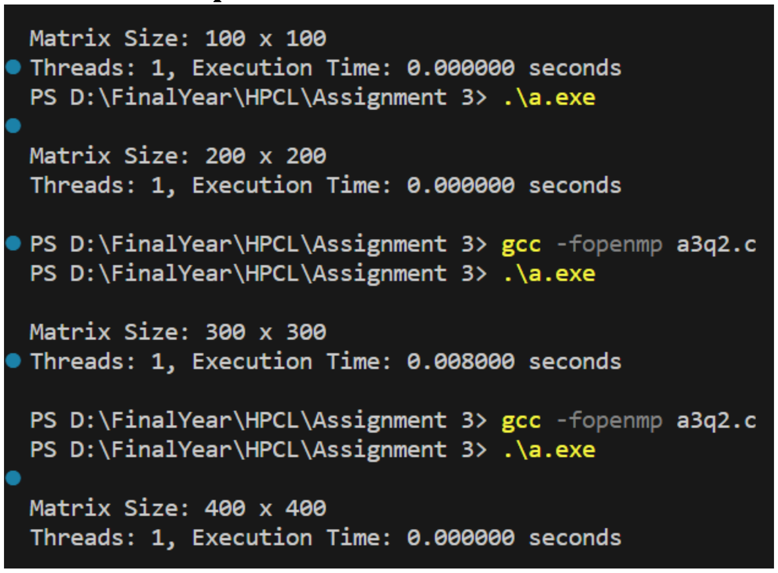


```
Matrix Size: 100 x 100
Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc q2sequential.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 200 x 200
Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc q2sequential.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 300 x 300
Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc q2sequential.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 400 x 400
Execution Time: 0.000000 seconds
```

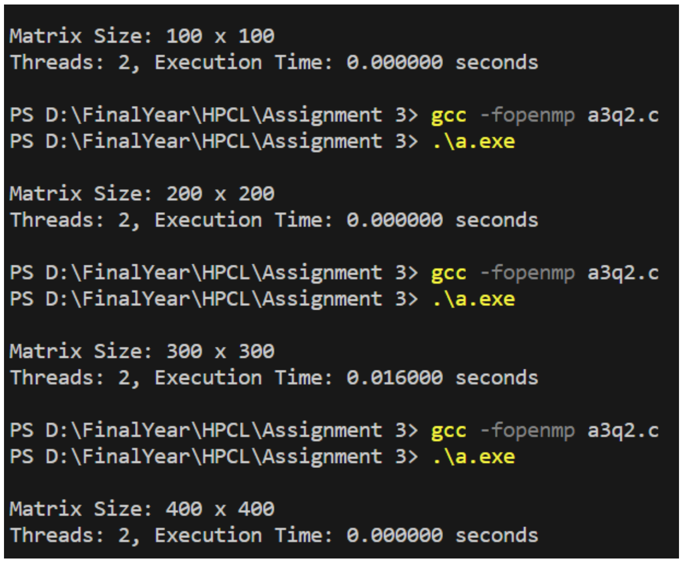
Parallel:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#define size 400
#define num_thread 1
int main()
{
    clock_t start_time = clock();
    int A[size][size];
    int B[size][size];
    // size of matrix {50, 100, 200, 300};
    // num threads {1, 2, 4, 8};
    for (int i = 0; i < size; i++)  {
        for (int j = 0; j < size; j++)
           A[i][j] = 100;
    }
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++)
           B[i][j] = 100;
    }
    int C[size][size];
    omp_set_num_threads(num_thread);
    #pragma omp parallel for
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++)
            C[i][j] = A[i][j] + B[i][j];
    }
    printf("\nMatrix Size: %d x %d\n", size, size);
    clock_t end_time = clock();
    double execution_time =(double)( end_time -
start_time)/CLOCKS_PER_SEC;
    printf("Threads: %d, Execution Time: %f seconds\n\n", num_thread,
execution_time);
    return 0;
}
```

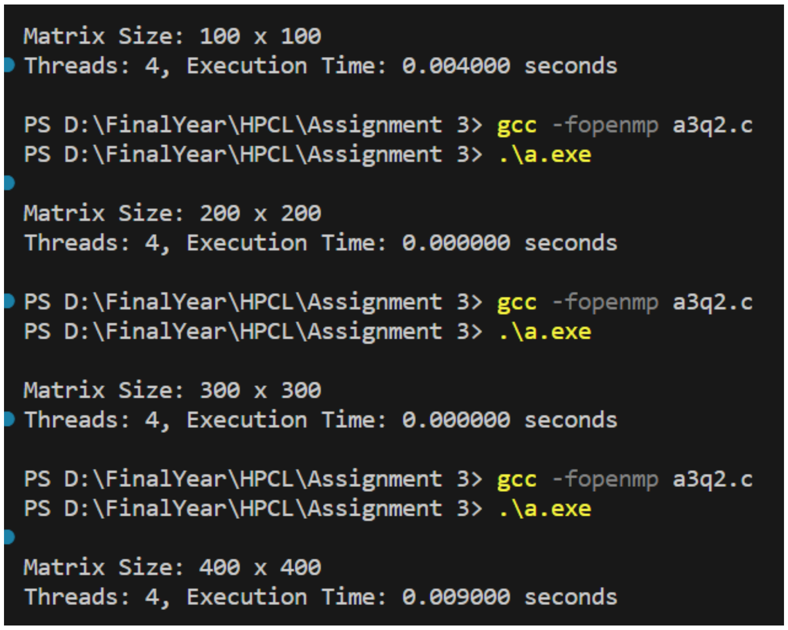
Output:



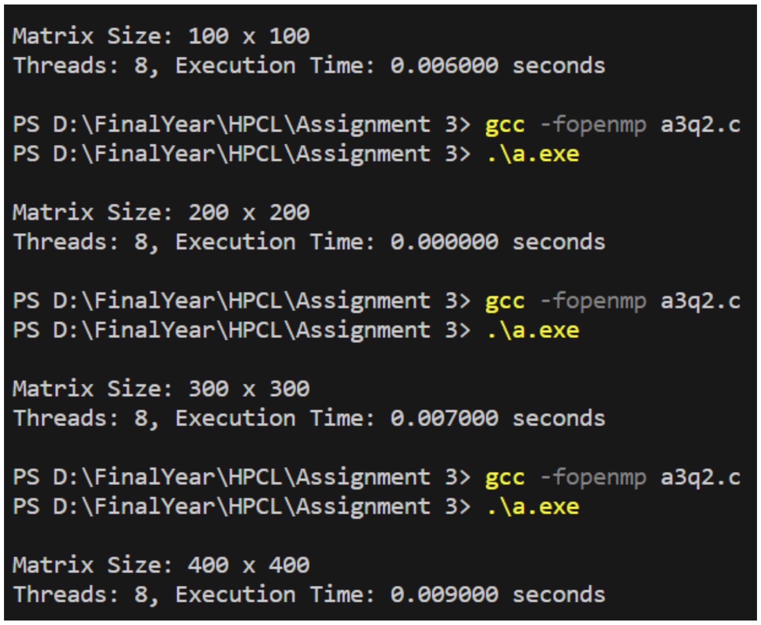
```
Matrix Size: 100 x 100
Threads: 1, Execution Time: 0.000000 seconds
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 200 x 200
Threads: 1, Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 300 x 300
Threads: 1, Execution Time: 0.008000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 400 x 400
Threads: 1, Execution Time: 0.000000 seconds
```



```
Matrix Size: 100 x 100
Threads: 2, Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 200 x 200
Threads: 2, Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 300 x 300
Threads: 2, Execution Time: 0.016000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 400 x 400
Threads: 2, Execution Time: 0.000000 seconds
```



```
Matrix Size: 100 x 100
Threads: 4, Execution Time: 0.004000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 200 x 200
Threads: 4, Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 300 x 300
Threads: 4, Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 400 x 400
Threads: 4, Execution Time: 0.009000 seconds
```



```
Matrix Size: 100 x 100
Threads: 8, Execution Time: 0.006000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 200 x 200
Threads: 8, Execution Time: 0.000000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 300 x 300
Threads: 8, Execution Time: 0.007000 seconds

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q2.c
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Matrix Size: 400 x 400
Threads: 8, Execution Time: 0.009000 seconds
```

Information and analysis:

Size\threads	**1**	**2**	**4**	**8**
100	0.000	0.000	0.004	0.006
200	0.000	0.000	0.000	0.000
300	0.008	0.016	0.000	0.007
400	0.000	0.000	0.009	0.009

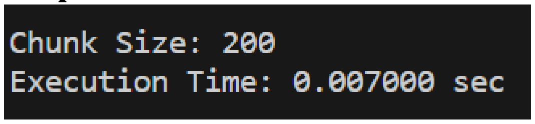
Sequential Execution Time for this is 0.000 seconds
Hence speed up in this case will also be 0

Problem Statement 3:
For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:
i. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyse the speedup.
ii. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyse the speedup.
iii. Demonstrate the use of nowait clause.

Sequential:

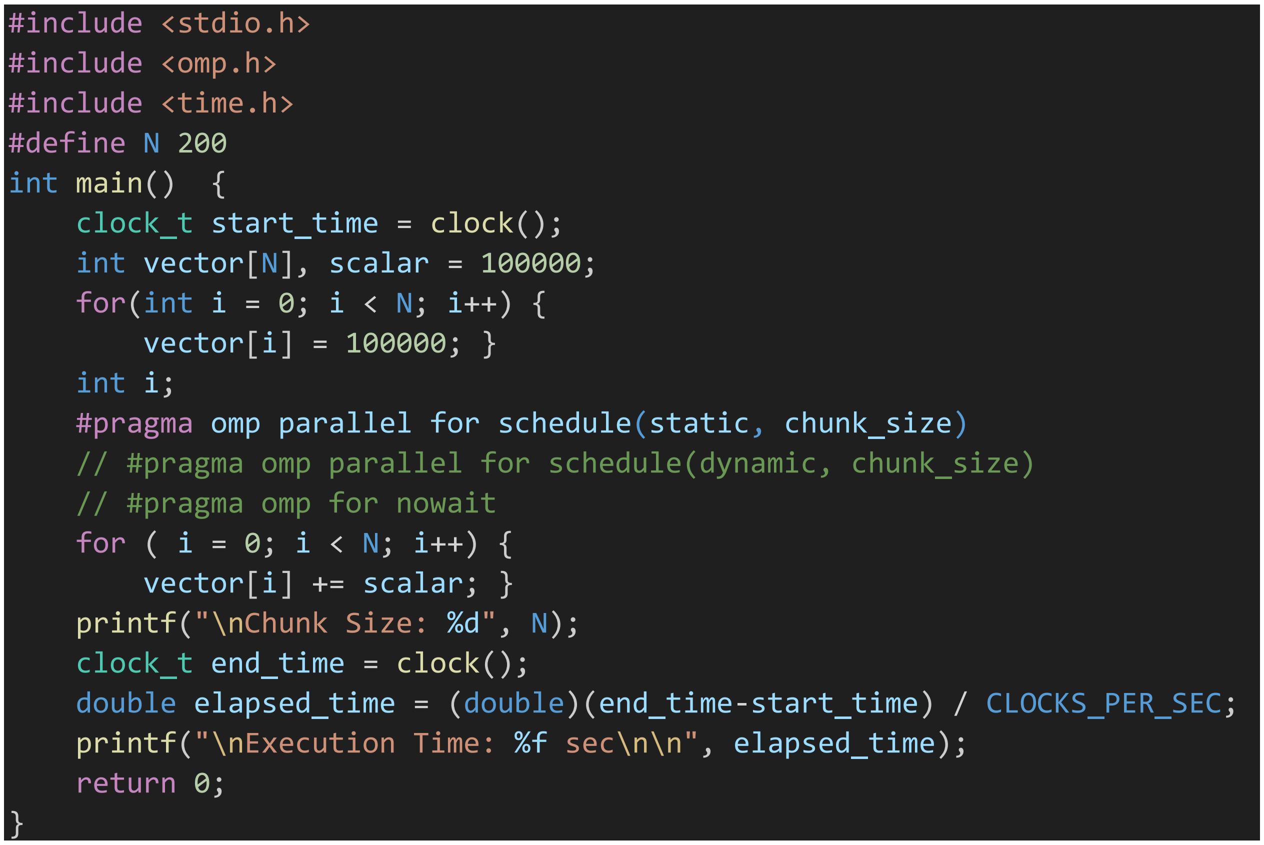
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#define N 200
int main()
{
    clock_t start_time = clock();
    int vector[N], scalar = 100000;
    for(int i = 0; i < N; i++)
    {
        vector[i] = 100000;
    }
    for (int i = 0; i < N; i++)
    {
        vector[i] += scalar;
    }
    printf("\nChunk Size: %d", N);
    clock_t end_time = clock();
    double elapsed_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;
    printf("\nExecution Time: %f sec\n\n", elapsed_time);
    return 0;
}
```

Output:



```
Chunk Size: 200
Execution Time: 0.007000 sec
```

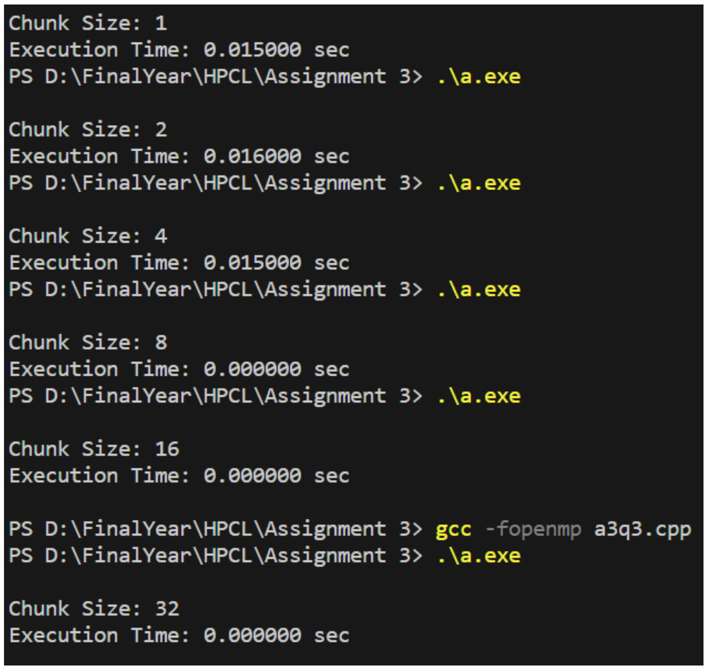
For Parallel:



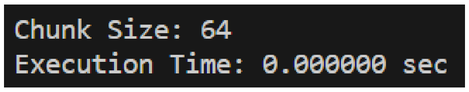
```
#include <stdio.h>
#include <omp.h>
#include <time.h>
#define N 200
int main()  {
    clock_t start_time = clock();
    int vector[N], scalar = 100000;
    for(int i = 0; i < N; i++) {
        vector[i] = 100000; }
    int i;
    #pragma omp parallel for schedule(static, chunk_size)
    // #pragma omp parallel for schedule(dynamic, chunk_size)
    // #pragma omp for nowait
    for ( i = 0; i < N; i++) {
        vector[i] += scalar; }
    printf("\nChunk Size: %d", N);
    clock_t end_time = clock();
    double elapsed_time = (double)(end_time-start_time) / CLOCKS_PER_SEC;
    printf("\nExecution Time: %f sec\n\n", elapsed_time);
    return 0;
}
```

Screenshots:

Static

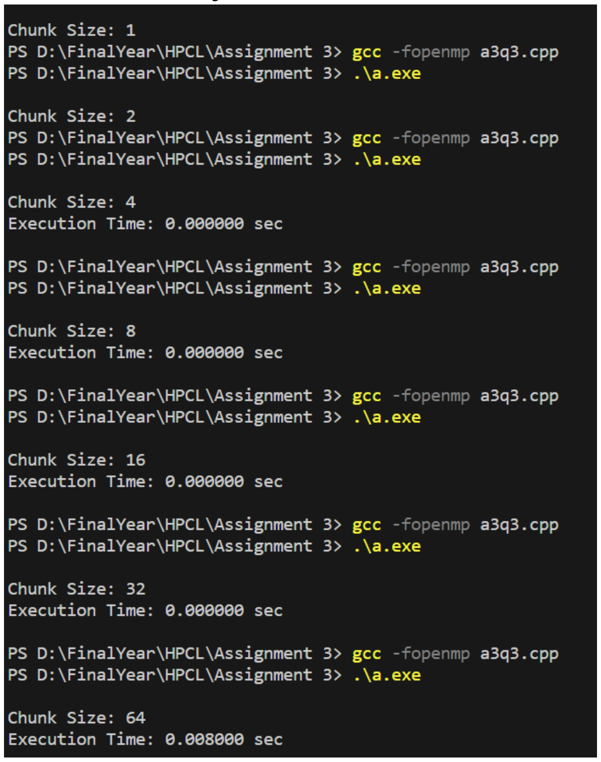


```
Chunk Size: 1
Execution Time: 0.015000 sec
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 2
Execution Time: 0.016000 sec
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 4
Execution Time: 0.015000 sec
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 8
Execution Time: 0.000000 sec
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 16
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 32
Execution Time: 0.000000 sec
```



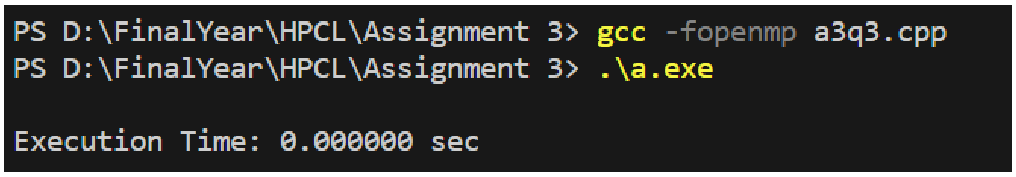
```
Chunk Size: 64
Execution Time: 0.000000 sec
```

Dynamic



```
Chunk Size: 1
PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 2
PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 4
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 8
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 16
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 32
Execution Time: 0.000000 sec

PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Chunk Size: 64
Execution Time: 0.008000 sec
```

Nowait:



```
PS D:\FinalYear\HPCL\Assignment 3> gcc -fopenmp a3q3.cpp
PS D:\FinalYear\HPCL\Assignment 3> .\a.exe

Execution Time: 0.000000 sec
```

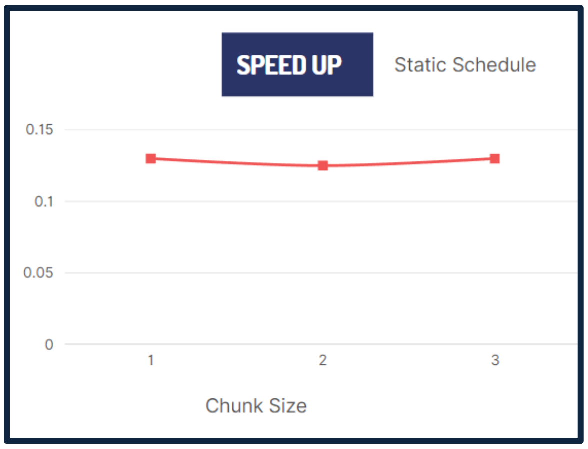
Information and analysis:
For static and dynamic Scheduling

Chunk Size	Execution Time for static	Execution Time for dynamic
1	0.015000 sec	0.000000 sec
2	0.016000 sec	0.000000 sec
4	0.015000 sec	0.000000 sec
8	0.000000 sec	0.000000 sec
16	0.000000 sec	0.000000 sec
32	0.000000 sec	0.000000 sec
64	0.000000 sec	0.000000 sec

Using nowait: 0.000000 seconds execution time

Sequential Execution Time: 0.002 seconds

Chunk Size	Speed Up for static schedule
1	0.13
2	0.125
4	0.13



Github Link: https://github.com/Nikita226/HPCL/tree/main/A3