Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

Final Year: High Performance Computing Lab 2022-23 Sem I

Class: Final Year (Computer Science and Engineering)

Year: 2022-23 Semester: 1

Course: High Performance Computing Lab

Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

Assignment No : 3

Title of practical:

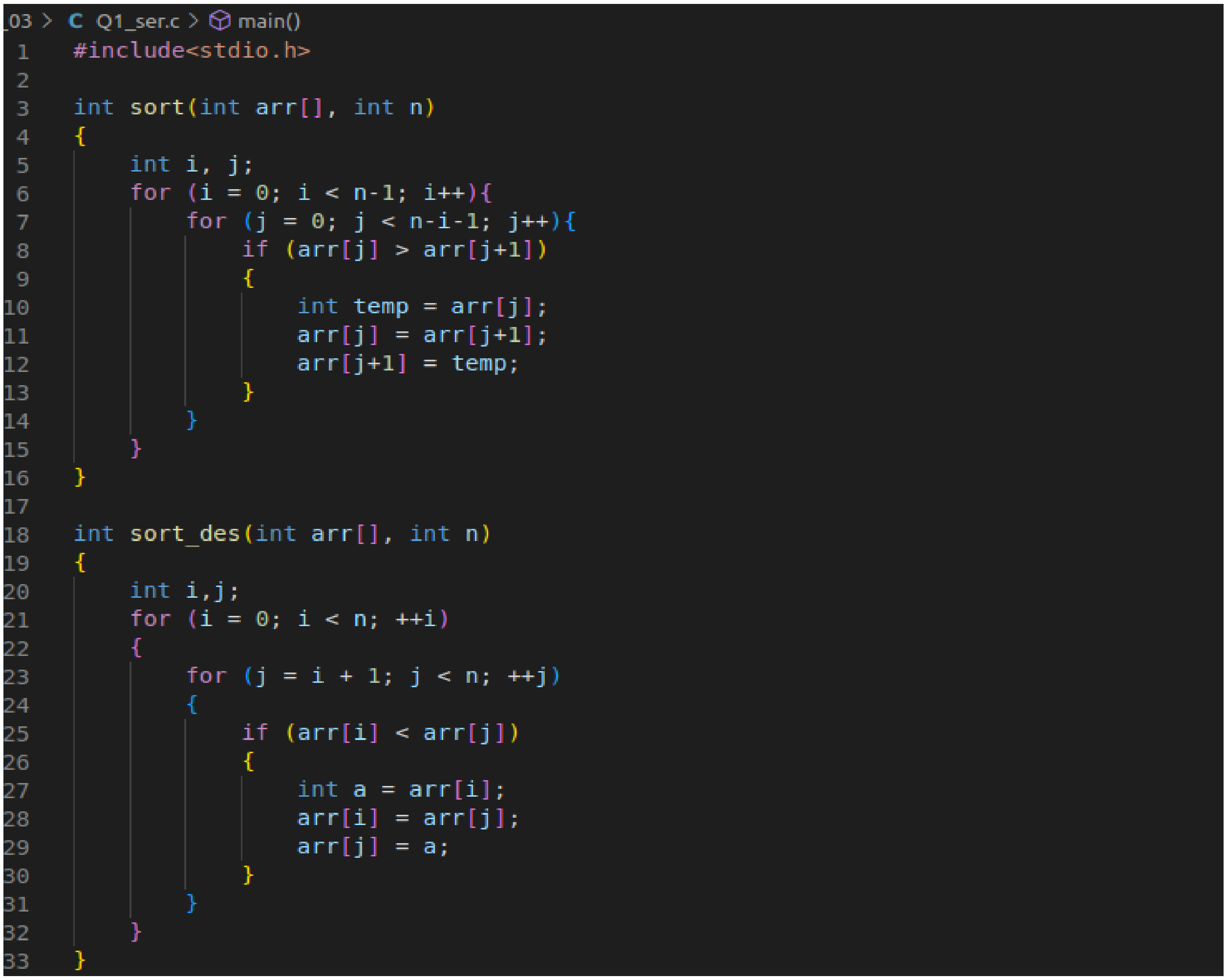
To Implement a parallel code for vector scalar addition

Problem Statement 1:

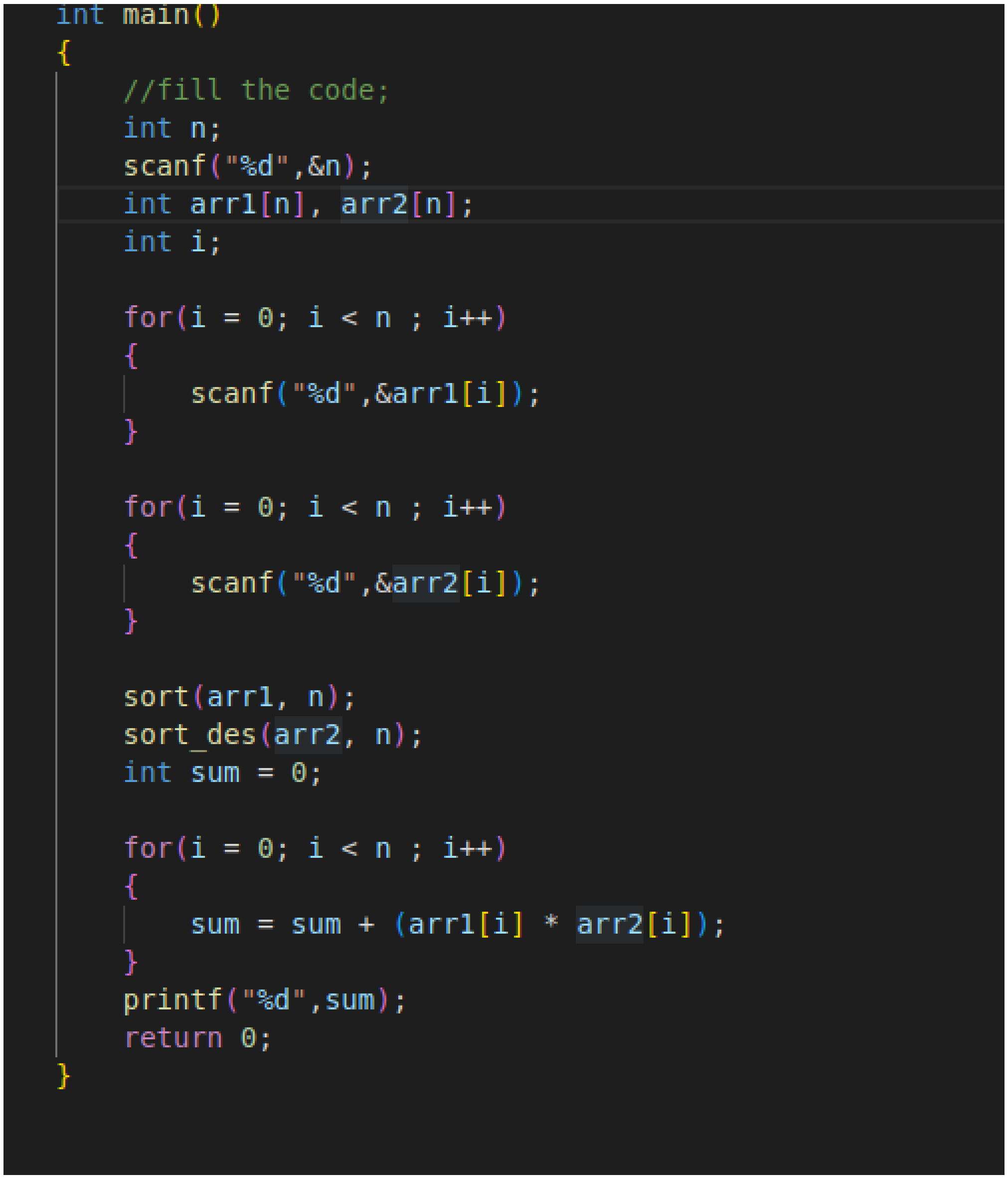
To Study and implementation of first private and shared variables

Q1: Analyse and implement a Parallel code for below program using OpenMP.

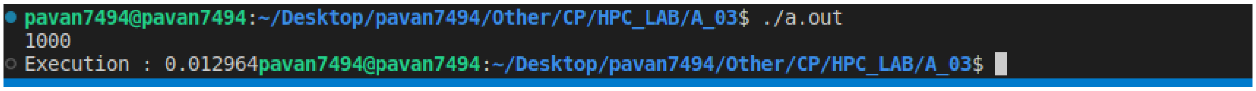
Sequential Code :



```c
#include<stdio.h>

int sort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++){
        for (j = 0; j < n-i-1; j++){
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

int sort_des(int arr[], int n)
{
    int i,j;
    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            if (arr[i] < arr[j])
            {
                int a = arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
    }
}
```

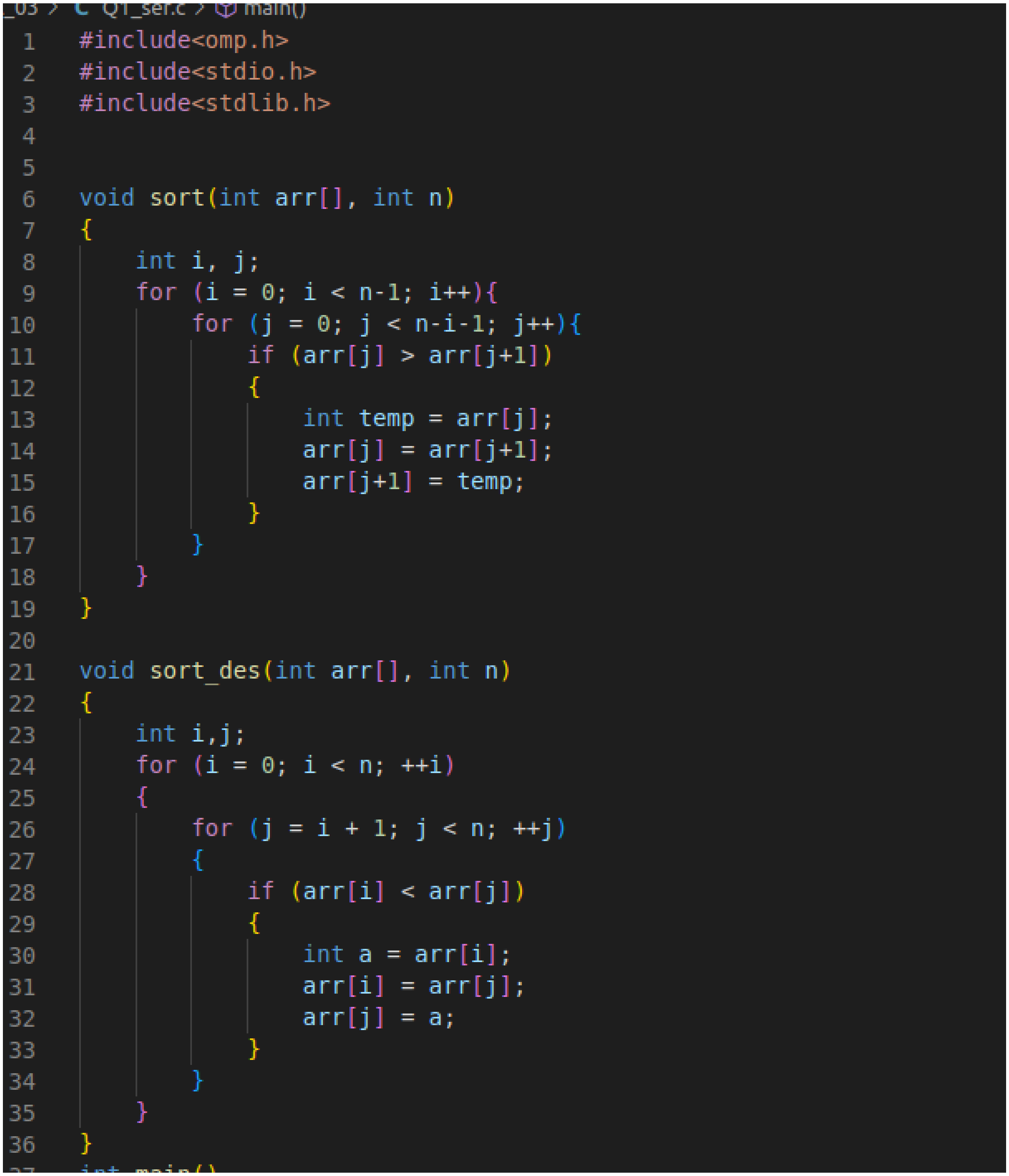


```
int main()
{
    //fill the code;
    int n;
    scanf("%d",&n);
    int arr1[n], arr2[n];
    int i;

    for(i = 0; i < n ; i++)
    {
        scanf("%d",&arr1[i]);
    }

    for(i = 0; i < n ; i++)
    {
        scanf("%d",&arr2[i]);
    }

    sort(arr1, n);
    sort_des(arr2, n);
    int sum = 0;

    for(i = 0; i < n ; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("%d",sum);
    return 0;
}
```

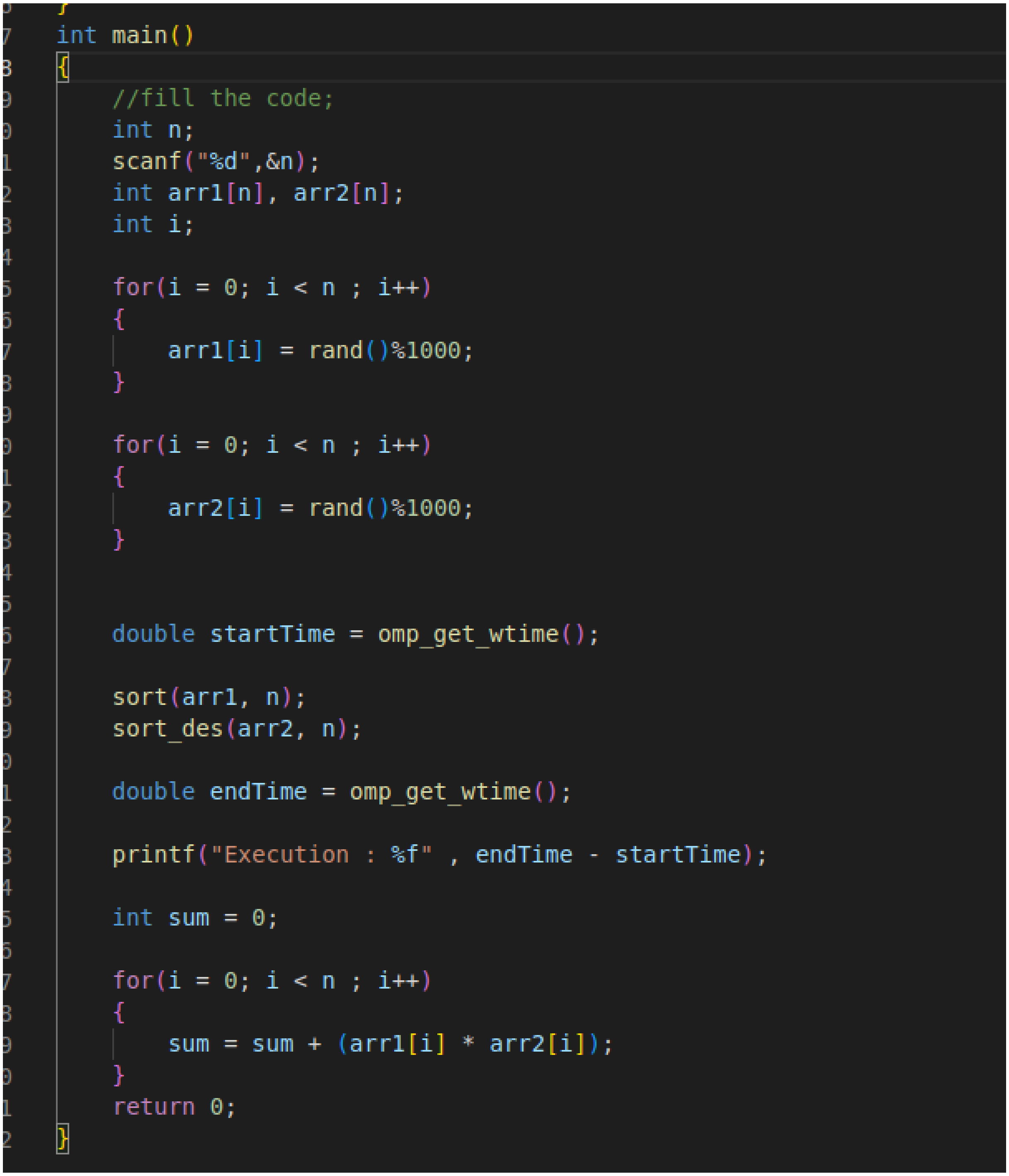


```
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_03$ ./a.out
1000
Execution : 0.012964pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_03$ 
```

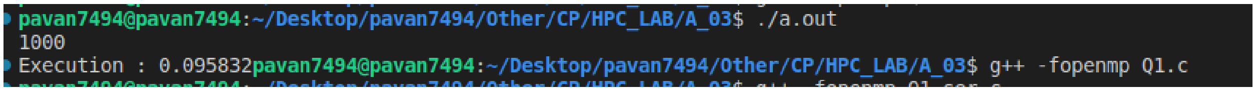
Parallel Code :



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


void sort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++){
        for (j = 0; j < n-i-1; j++){
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

void sort_des(int arr[], int n)
{
    int i,j;
    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            if (arr[i] < arr[j])
            {
                int a = arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
    }
}
int main()
```



```c
}
int main()
{
    //fill the code;
    int n;
    scanf("%d",&n);
    int arr1[n], arr2[n];
    int i;

    for(i = 0; i < n ; i++)
    {
        arr1[i] = rand()%1000;
    }

    for(i = 0; i < n ; i++)
    {
        arr2[i] = rand()%1000;
    }


    double startTime = omp_get_wtime();

    sort(arr1, n);
    sort_des(arr2, n);

    double endTime = omp_get_wtime();

    printf("Execution : %f" , endTime - startTime);

    int sum = 0;

    for(i = 0; i < n ; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    return 0;
}
```



```
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_03$ ./a.out
1000
Execution : 0.095832pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_03$ g++ -fopenmp Q1.c
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_03$ g++ -fopenmp Q1_ser.c
```

Information 1:

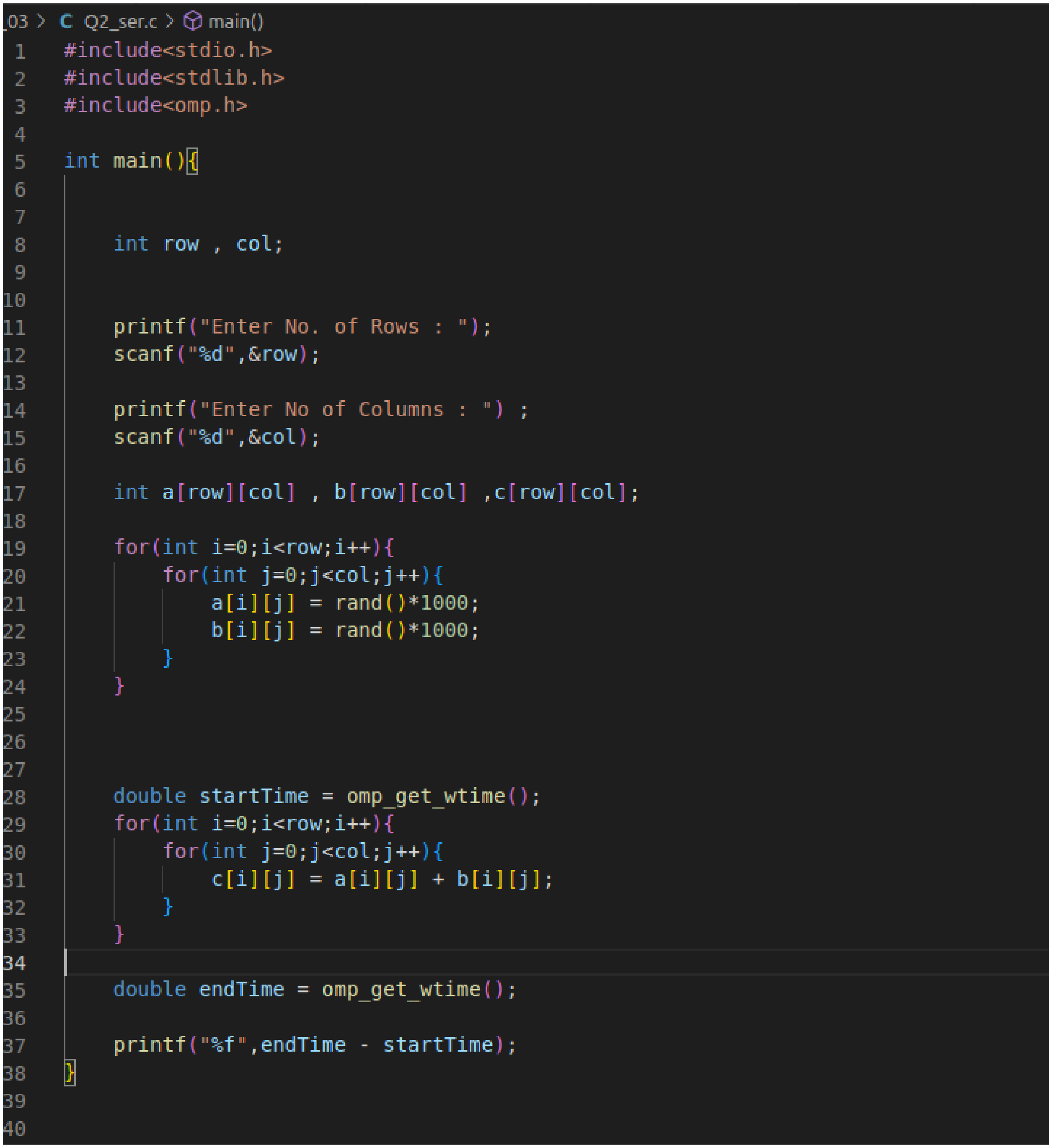
The default clause explicitly determines the data-sharing attributes of variables that are referenced in a parallel, teams, or task generating construct and would otherwise be implicitly determined. The default (shared) clause causes all variables referenced in the construct that have implicitly determined data-sharing attributes to be shared. The default (none) clause requires that each variable that is referenced in the construct, and that does not have a predetermined data-sharing attribute, must have its data-sharing attribute explicitly determined by being listed in a data-sharing attribute clause.

Q2. 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 calculating 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 behavior is as expected.

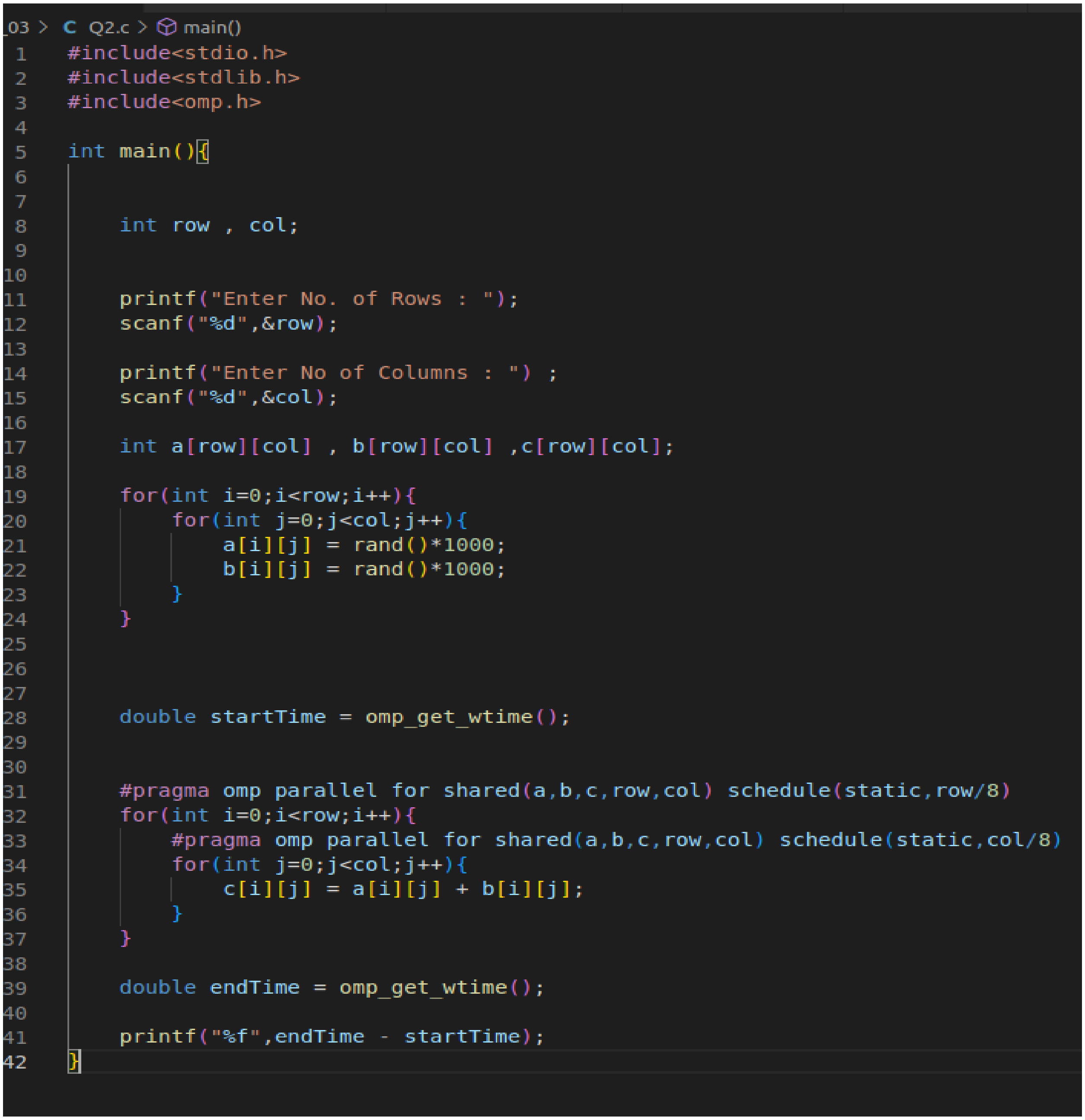
Sequential Program:



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

int main(){


    int row , col;


    printf("Enter No. of Rows : ");
    scanf("%d",&row);

    printf("Enter No of Columns : ") ;
    scanf("%d",&col);

    int a[row][col] , b[row][col] ,c[row][col];

    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            a[i][j] = rand()*1000;
            b[i][j] = rand()*1000;
        }
    }



    double startTime = omp_get_wtime();
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            c[i][j] = a[i][j] + b[i][j];
        }
    }

    double endTime = omp_get_wtime();

    printf("%f",endTime - startTime);
}


```

Size of Matrix	250	500	750	1000	2000
Runtime	0.000452	0.000736	0.002858	Seg. Fault	Seg. Fault

Parallel Program :



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

int main(){


    int row , col;


    printf("Enter No. of Rows : ");
    scanf("%d",&row);

    printf("Enter No of Columns : ") ;
    scanf("%d",&col);

    int a[row][col] , b[row][col] ,c[row][col];

    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            a[i][j] = rand()*1000;
            b[i][j] = rand()*1000;
        }
    }



    double startTime = omp_get_wtime();


    #pragma omp parallel for shared(a,b,c,row,col) schedule(static,row/8)
    for(int i=0;i<row;i++){
        #pragma omp parallel for shared(a,b,c,row,col) schedule(static,col/8)
        for(int j=0;j<col;j++){
            c[i][j] = a[i][j] + b[i][j];
        }
    }

    double endTime = omp_get_wtime();

    printf("%f",endTime - startTime);
}
```

Size of Matrix / No.of threads	250	500	750	1000	2000
2	0.000656	0.000847	0.009830	Seg. Fault	Seg. Fault
4	0.000447	0.000589	0.007251	Seg. Fault	Seg. Fault
6	0.000444	0.000815	0.013948	Seg. Fault	Seg. Fault
8	0.029674	0.020178	0.026631	Seg. Fault	Seg. Fault

Conclusion :

As Task size increases, time required is also increased. As number of treads increases efficiency first increases and decreases so, maximum efficiency is achieved at 4 threads .

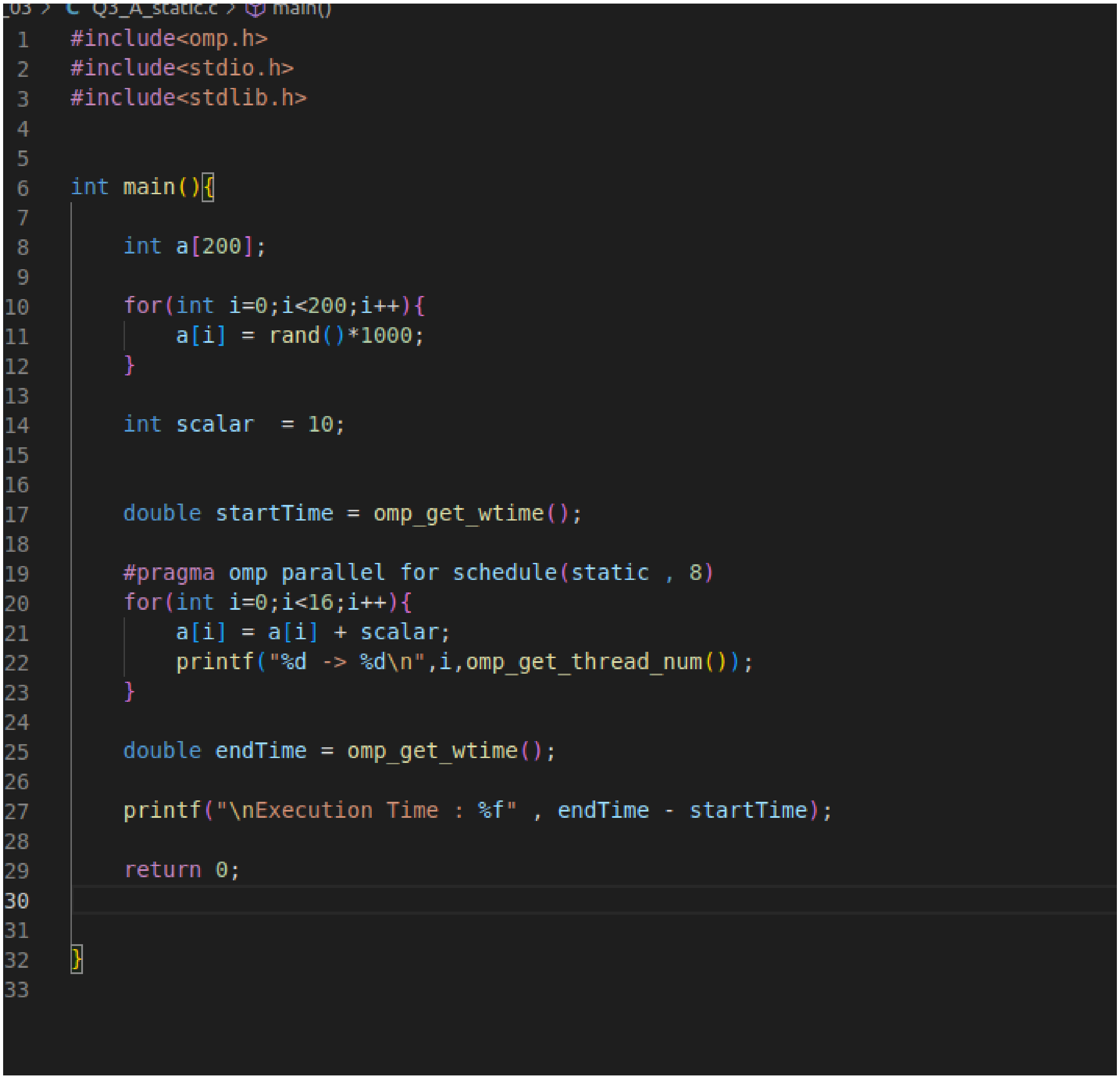
Information 2:

The reduction clauses are data-sharing attribute clauses that can be used to perform some forms of recurrence calculations in parallel. Reduction clauses include reduction scoping clauses and reduction participating clauses. Reduction scoping clauses define the region in which a reduction is computed. Reduction participating clauses define the participants in the reduction. Reduction clauses specify a reduction-identifier and one or more list items. A reduction-identifier is either an identifier or one of the following operators: +, -, *, &, |, ^, && and ||.

Q3. For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:

i. Use the STATIC schedule and set the loop iteration chunk size to various sizes
when changing the size of your matrix. Analyze the speedup.

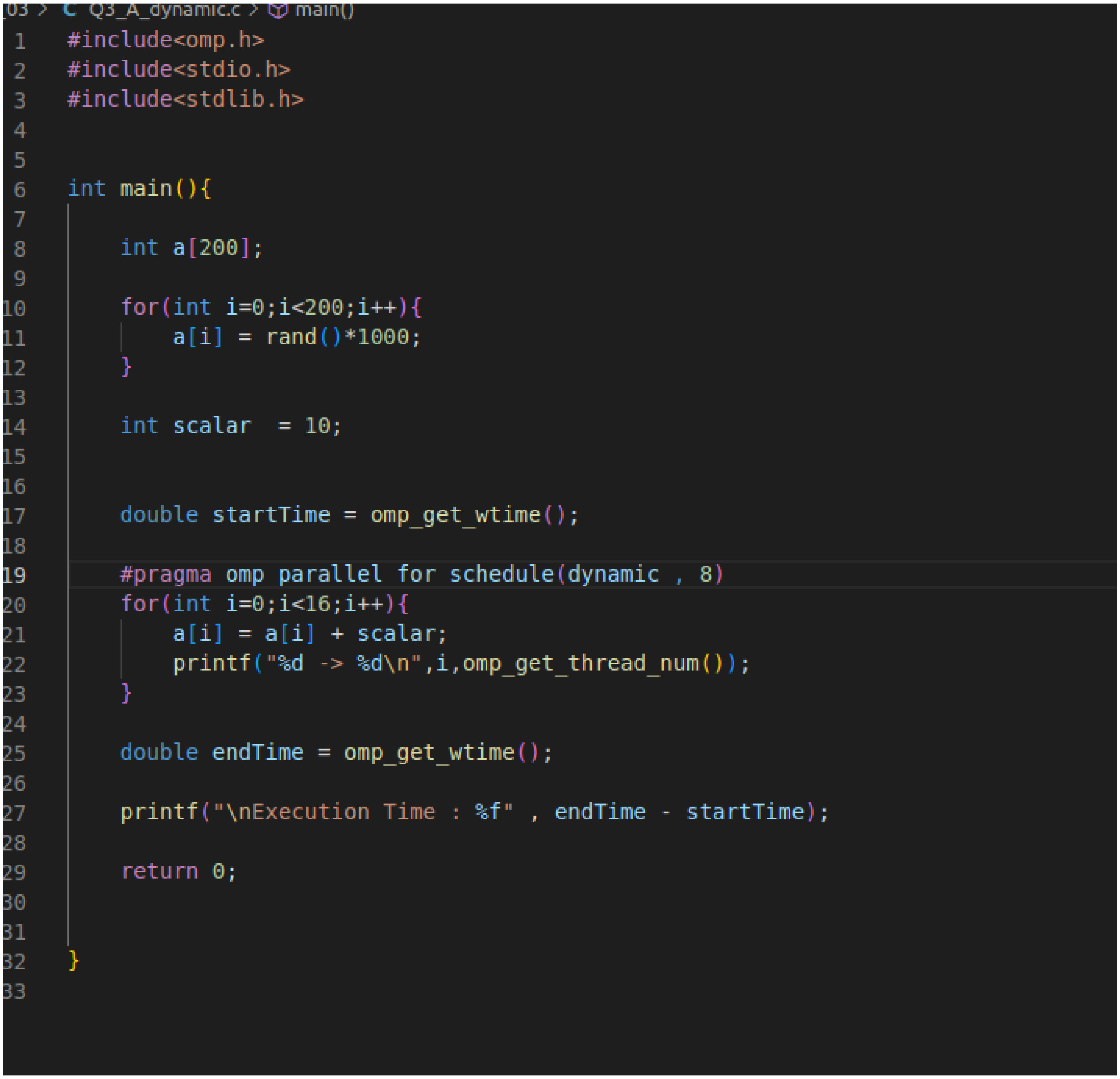
ii. Use the DYNAMIC schedule and set the loop iteration chunk size to various sizes
when changing the size of your matrix. Analyze the speedup.
iii. Demonstrate the use of nowait clause
i. Use the STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.



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


int main(){

    int a[200];

    for(int i=0;i<200;i++){
        a[i] = rand()*1000;
    }

    int scalar  = 10;


    double startTime = omp_get_wtime();

    #pragma omp parallel for schedule(static , 8)
    for(int i=0;i<16;i++){
        a[i] = a[i] + scalar;
        printf("%d -> %d\n",i,omp_get_thread_num());
    }

    double endTime = omp_get_wtime();

    printf("\nExecution Time : %f" , endTime - startTime);

    return 0;


}

```

Chunk Size	2	4	6	8
Runtime	0.023053	0.026576	0.028262	0.025873

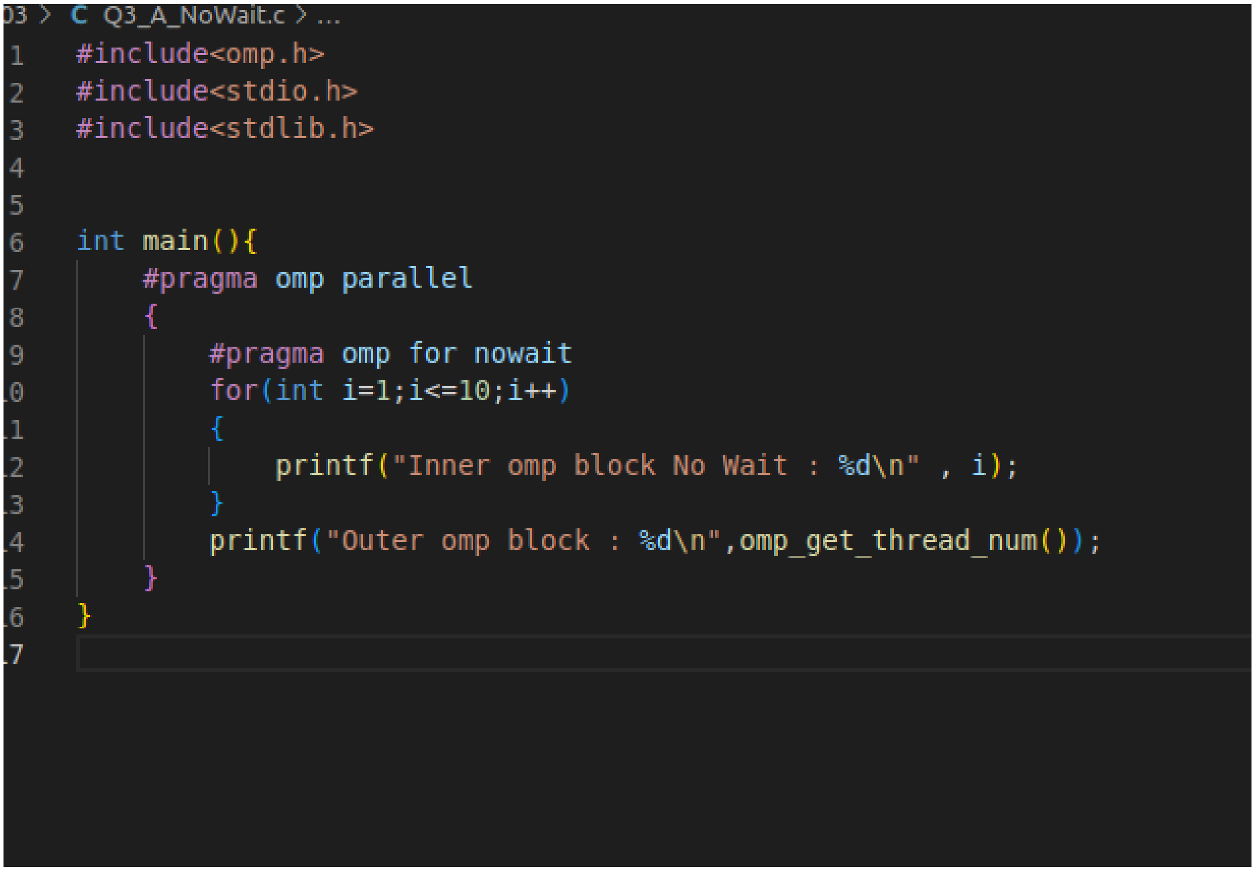
ii. Use the DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.



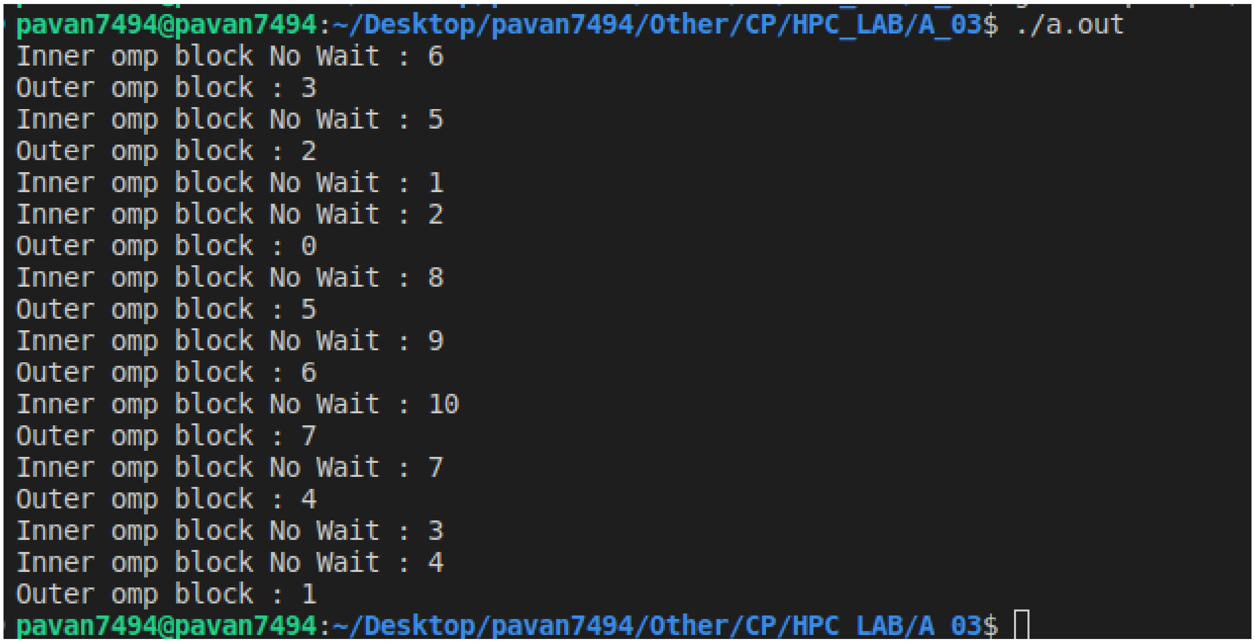
```c
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>


int main(){

    int a[200];

    for(int i=0;i<200;i++){
        a[i] = rand()*1000;
    }

    int scalar  = 10;


    double startTime = omp_get_wtime();

    #pragma omp parallel for schedule(dynamic , 8)
    for(int i=0;i<16;i++){
        a[i] = a[i] + scalar;
        printf("%d -> %d\n",i,omp_get_thread_num());
    }

    double endTime = omp_get_wtime();

    printf("\nExecution Time : %f" , endTime - startTime);

    return 0;


}

```

Chunk Size	2	4	6	8
Runtime	0.032318	0.022752	0.023094	0.033200

iii. Demonstrate the use of nowait clause.



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


int main(){
    #pragma omp parallel
    {
        #pragma omp for nowait
        for(int i=1;i<=10;i++)
        {
            printf("Inner omp block No Wait : %d\n" , i);
        }
        printf("Outer omp block : %d\n",omp_get_thread_num());
    }
}

```



```
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC_LAB/A_03$ ./a.out
Inner omp block No Wait : 6
Outer omp block : 3
Inner omp block No Wait : 5
Outer omp block : 2
Inner omp block No Wait : 1
Inner omp block No Wait : 2
Outer omp block : 0
Inner omp block No Wait : 8
Outer omp block : 5
Inner omp block No Wait : 9
Outer omp block : 6
Inner omp block No Wait : 10
Outer omp block : 7
Inner omp block No Wait : 7
Outer omp block : 4
Inner omp block No Wait : 3
Inner omp block No Wait : 4
Outer omp block : 1
pavan7494@pavan7494:~/Desktop/pavan7494/Other/CP/HPC LAB/A 03$ 
```