Final Year B. Tech, Sem VII
2022-23PRN – 2020BTECS00211
Name – Aashita Narendra
Gupta High Performance
Computing LabBatch: B4

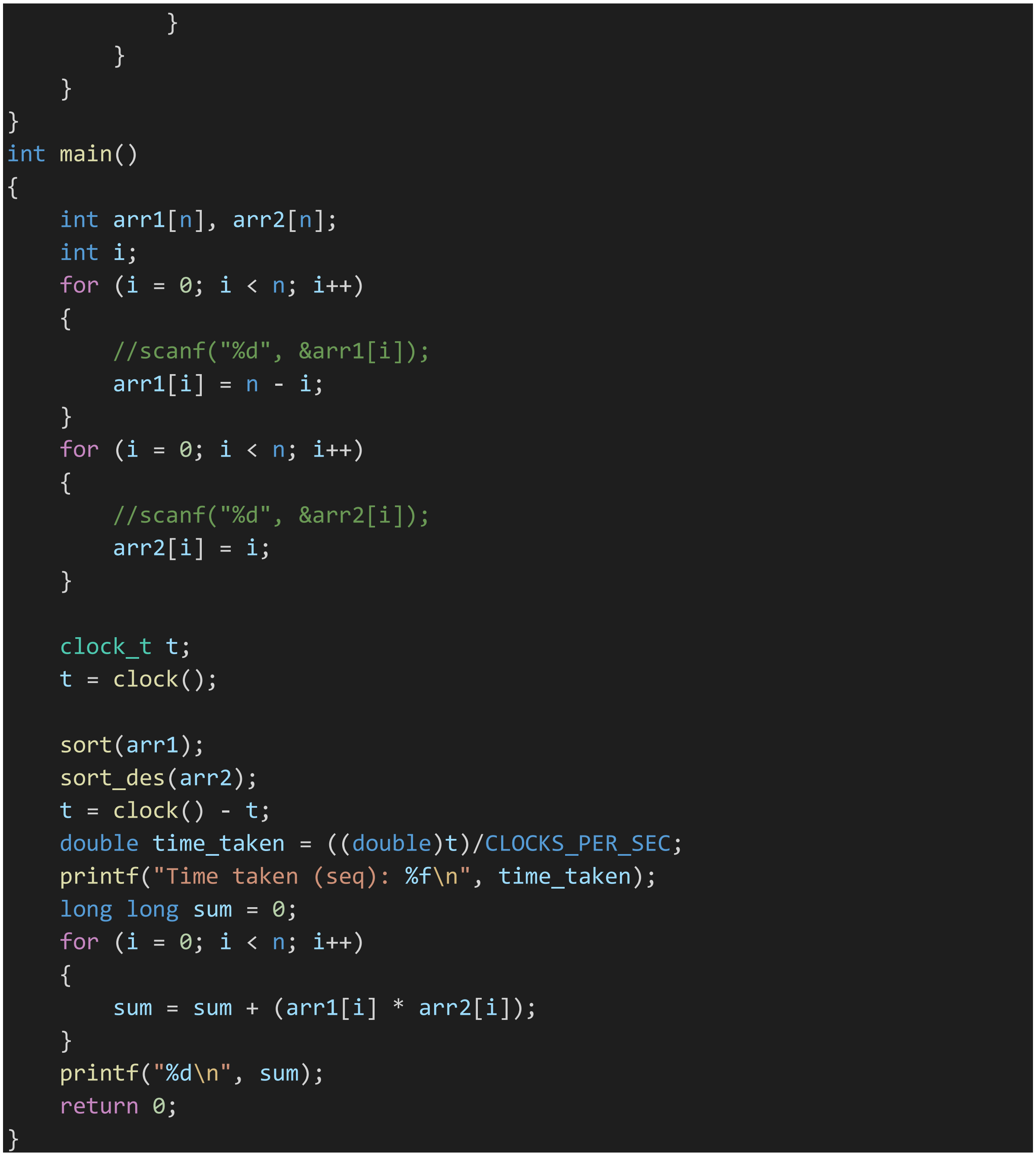
Practical No – 3

Title: Study and Implementation of schedule, nowait, reduction, ordered and collapse Clauses.

Github Link for Code - https://github.com/Aashita06/HPC_Practicals

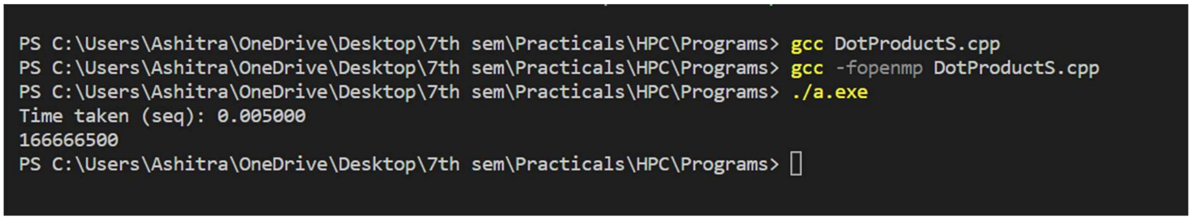
Q.1) Analyse and implement a Parallel code for below program using OpenMP.
→
Sequential code:

```c
// C Program to find the minimum scalar product of two vectors(dot product)
#include <stdio.h>
#include <time.h>
#define n 1000
int sort(int arr[])
{
    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 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()
{
    int arr1[n], arr2[n];
    int i;
    for (i = 0; i < n; i++)
    {
        //scanf("%d", &arr1[i]);
        arr1[i] = n - i;
    }
    for (i = 0; i < n; i++)
    {
        //scanf("%d", &arr2[i]);
        arr2[i] = i;
    }

    clock_t t;
    t = clock();

    sort(arr1);
    sort_des(arr2);
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC;
    printf("Time taken (seq): %f\n", time_taken);
    long long sum = 0;
    for (i = 0; i < n; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("%d\n", sum);
    return 0;
}
```

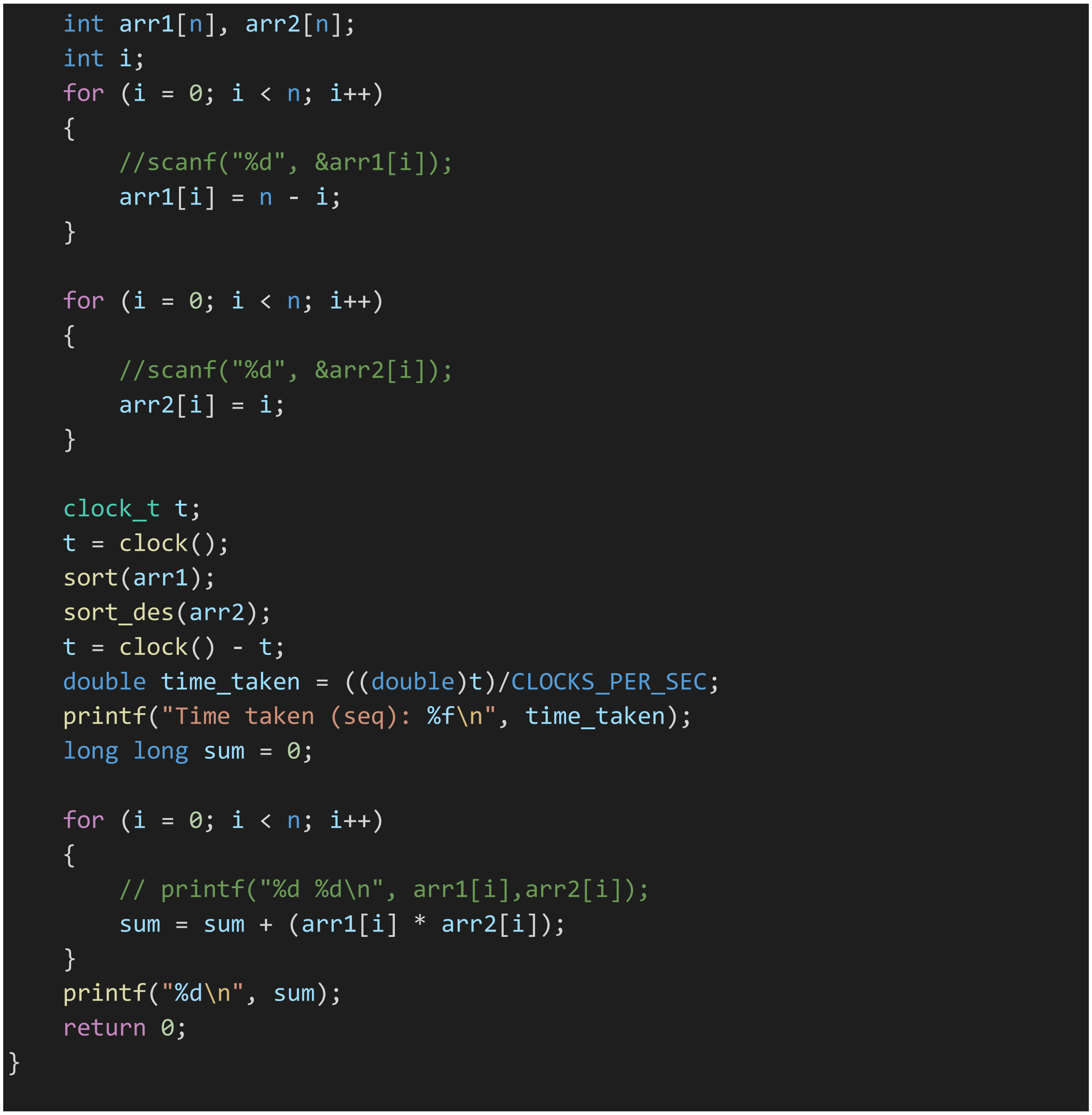
Output:



```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc DotProductS.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp DotProductS.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.005000
166666500
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> 
```

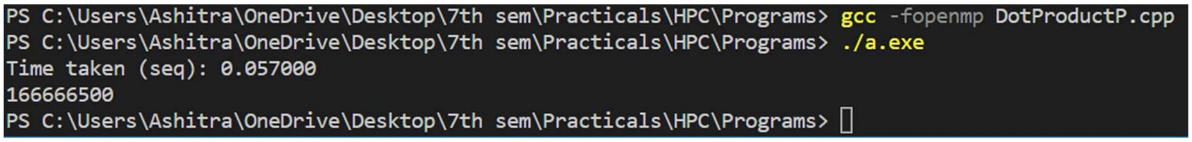
Parallel Code:

```
// C Program to find the minimum scalar product of two vectors (dot product)
#include <stdio.h>
#include <time.h>
#include<omp.h>
#define n 1000
int sort(int arr[])
{
    int i, j;

    for (i = 0; i < n; i++)
    {
        int turn = i % 2;

        #pragma omp parallel for

        for (j = turn; j < n - 1; j+=2)
            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 i, j;
    for (i = 0; i < n; ++i)
    {
        int turn = i % 2;
        #pragma omp parallel for
        for (j = turn; j < n - 1; j += 2)
        {
            // printf("Thread ID:%d",omp_get_thread_num());
            if (arr[j] < arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main()
{
```



```
    int arr1[n], arr2[n];
    int i;
    for (i = 0; i < n; i++)
    {
        //scanf("%d", &arr1[i]);
        arr1[i] = n - i;
    }

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

    clock_t t;
    t = clock();
    sort(arr1);
    sort_des(arr2);
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC;
    printf("Time taken (seq): %f\n", time_taken);
    long long sum = 0;

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

Output:



```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp DotProductP.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.057000
166666500
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ▯
```

Speedup = Ts/Tp = 0.005000/0.057000 = 0.0877192982456

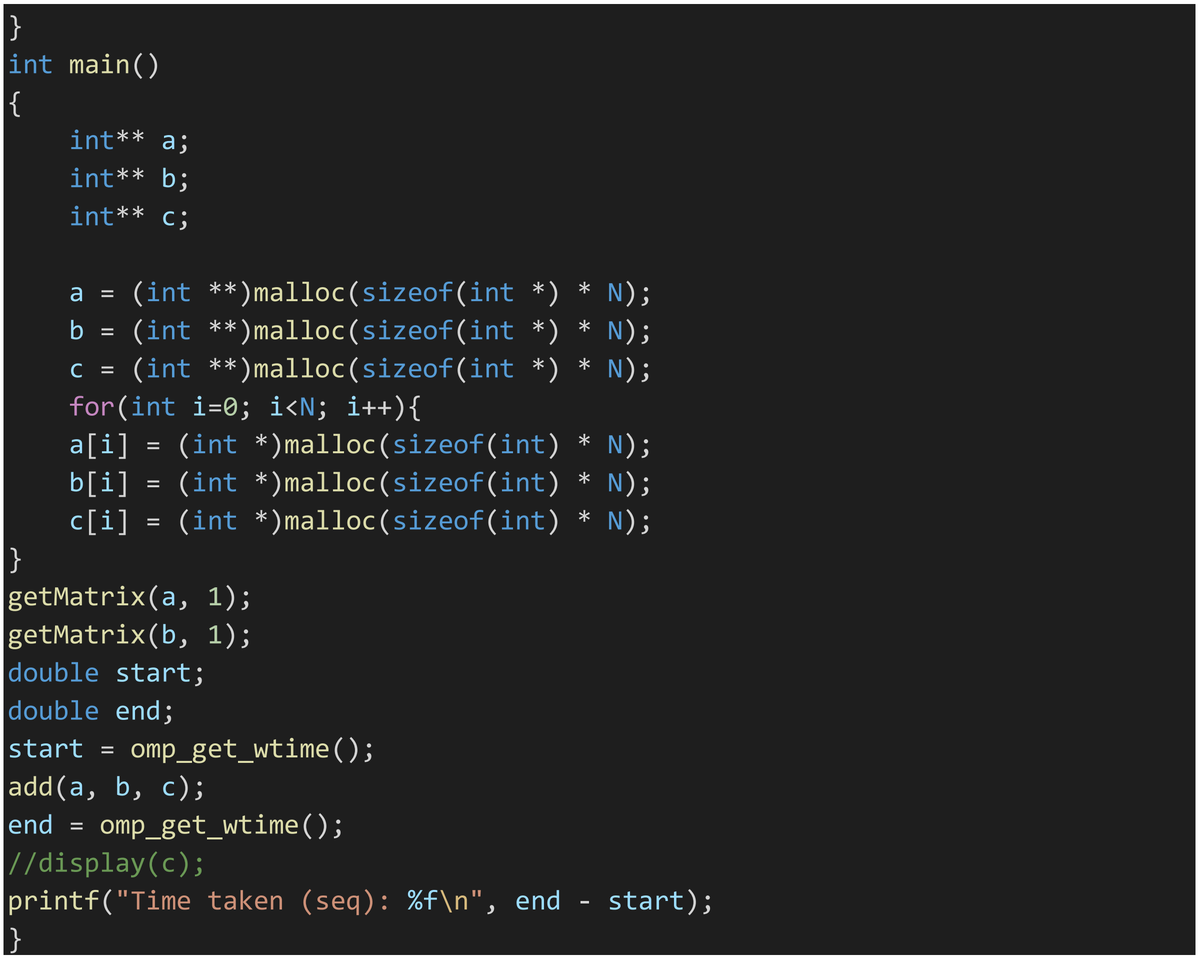
Q2) Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Usefunctions in C in calculating the execution time or use GPROF)

i. For each matrix size, change the number of threads from 2,4,8., and plot thespeedup versus the number of threads.

ii. Explain whether or not the scaling behavior is as expected.
→

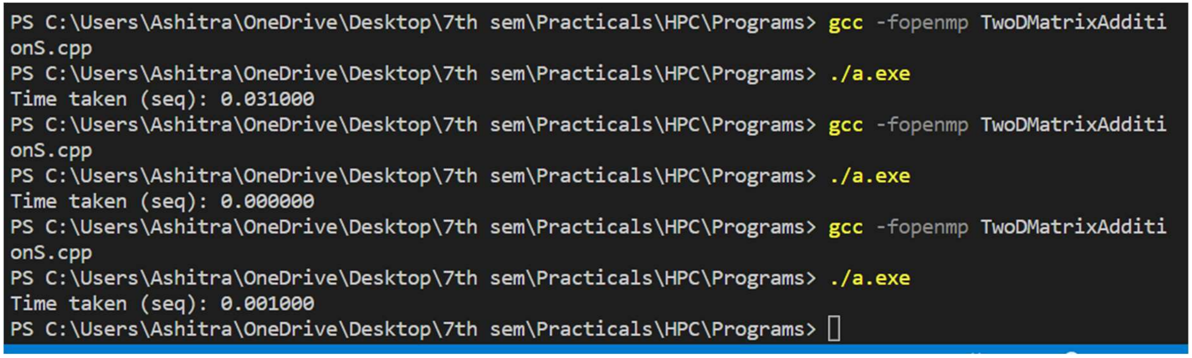
Serial Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define N 500
void add(int** a, int** b, int** c)
{
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}
void getMatrix(int** a, int num)
{
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            a[i][j] = num;
        }
    }
}
void display(int** a)
{
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
```



```
}
int main()
{
    int** a;
    int** b;
    int** c;

    a = (int **)malloc(sizeof(int *) * N);
    b = (int **)malloc(sizeof(int *) * N);
    c = (int **)malloc(sizeof(int *) * N);
    for(int i=0; i<N; i++){
    a[i] = (int *)malloc(sizeof(int) * N);
    b[i] = (int *)malloc(sizeof(int) * N);
    c[i] = (int *)malloc(sizeof(int) * N);
}
getMatrix(a, 1);
getMatrix(b, 1);
double start;
double end;
start = omp_get_wtime();
add(a, b, c);
end = omp_get_wtime();
//display(c);
printf("Time taken (seq): %f\n", end - start);
}
```

Output:

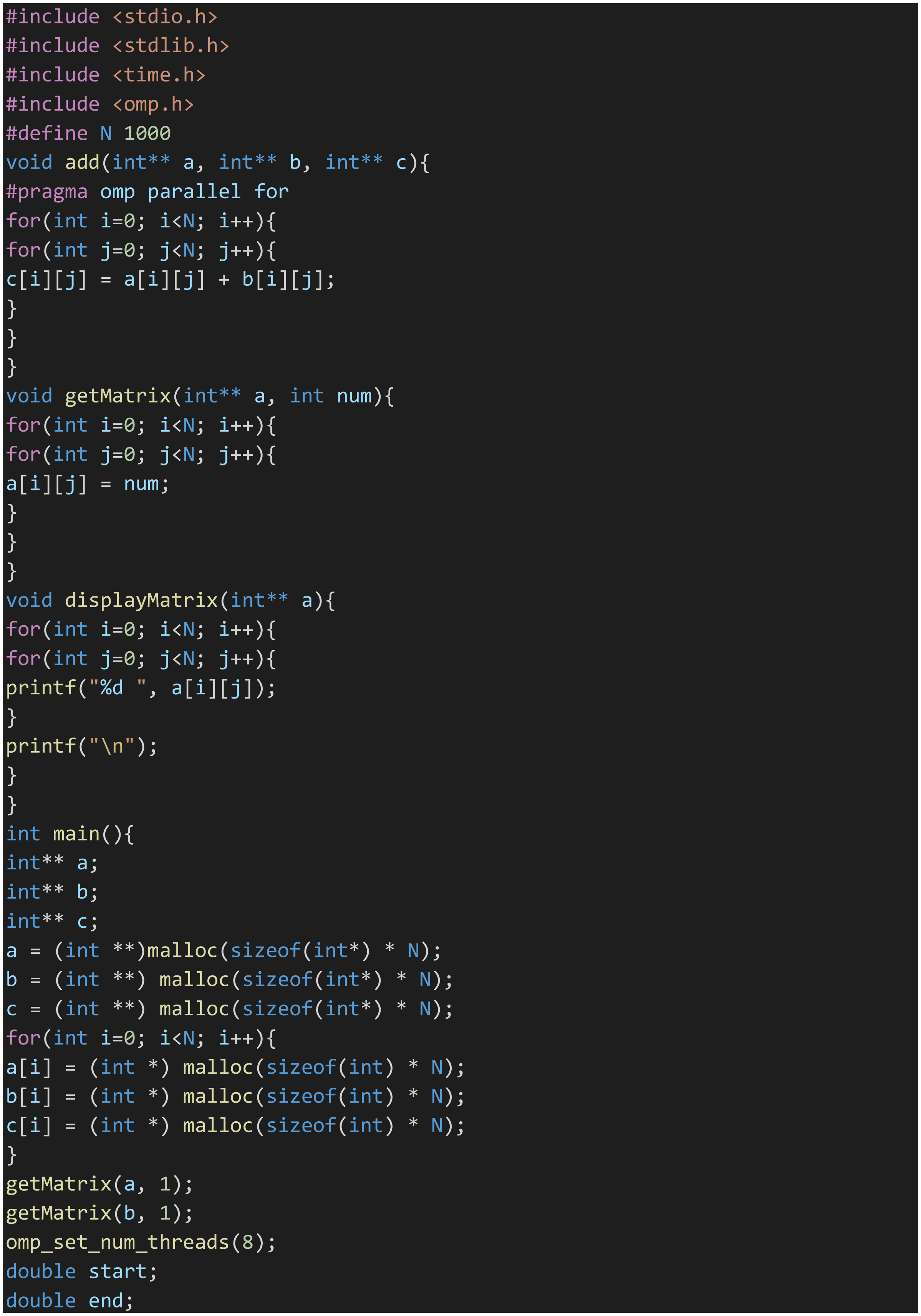


```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp TwoDMatrixAdditi
onS.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.031000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp TwoDMatrixAdditi
onS.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.000000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp TwoDMatrixAdditi
onS.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.001000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> 
```

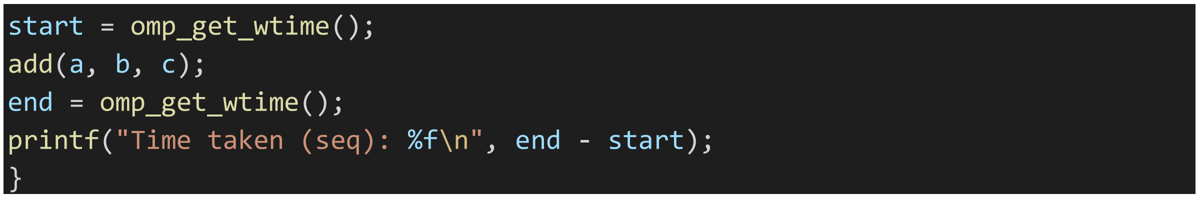
Statitics:

N	250	500	750	1000	2000
Time	0.000609	0.002412	0.007251	0.008956	0.017441

Parallel Code:

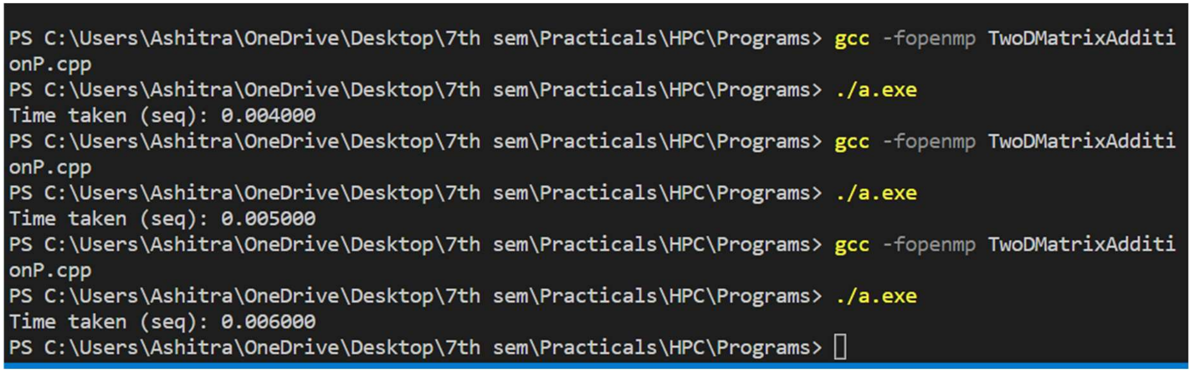


```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define N 1000
void add(int** a, int** b, int** c){
#pragma omp parallel for
for(int i=0; i<N; i++){
for(int j=0; j<N; j++){
c[i][j] = a[i][j] + b[i][j];
}
}
}
void getMatrix(int** a, int num){
for(int i=0; i<N; i++){
for(int j=0; j<N; j++){
a[i][j] = num;
}
}
}
void displayMatrix(int** a){
for(int i=0; i<N; i++){
for(int j=0; j<N; j++){
printf("%d ", a[i][j]);
}
printf("\n");
}
}
int main(){
int** a;
int** b;
int** c;
a = (int **)malloc(sizeof(int*) * N);
b = (int **) malloc(sizeof(int*) * N);
c = (int **) malloc(sizeof(int*) * N);
for(int i=0; i<N; i++){
a[i] = (int *) malloc(sizeof(int) * N);
b[i] = (int *) malloc(sizeof(int) * N);
c[i] = (int *) malloc(sizeof(int) * N);
}
getMatrix(a, 1);
getMatrix(b, 1);
omp_set_num_threads(8);
double start;
double end;
```



```
start = omp_get_wtime();
add(a, b, c);
end = omp_get_wtime();
printf("Time taken (seq): %f\n", end - start);
}
```

Output:



```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp TwoDMatrixAdditi
onP.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.004000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp TwoDMatrixAdditi
onP.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.005000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp TwoDMatrixAdditi
onP.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken (seq): 0.006000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> 
```

Threads \ N	**250**	**500**	**750**	**1000**	**2000**
2	**0.000505**	**0.001429**	**0.005316**	**0.005072**	**0.008615**
4	**0.000487**	**0.001721**	**0.004917**	**0.002474**	**0.007063**
6	**0.000462**	**0.001157**	**0.008115**	**0.022801**	**0.005138**
8	**0.031153**	**0.025150**	**0.030609**	**0.035497**	**0.004030**

Conclusion: For the smaller values of N the smaller number of thread gives optimal timeresults. But for significantly larger values of N the greater number of threads will give optimal time results.

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

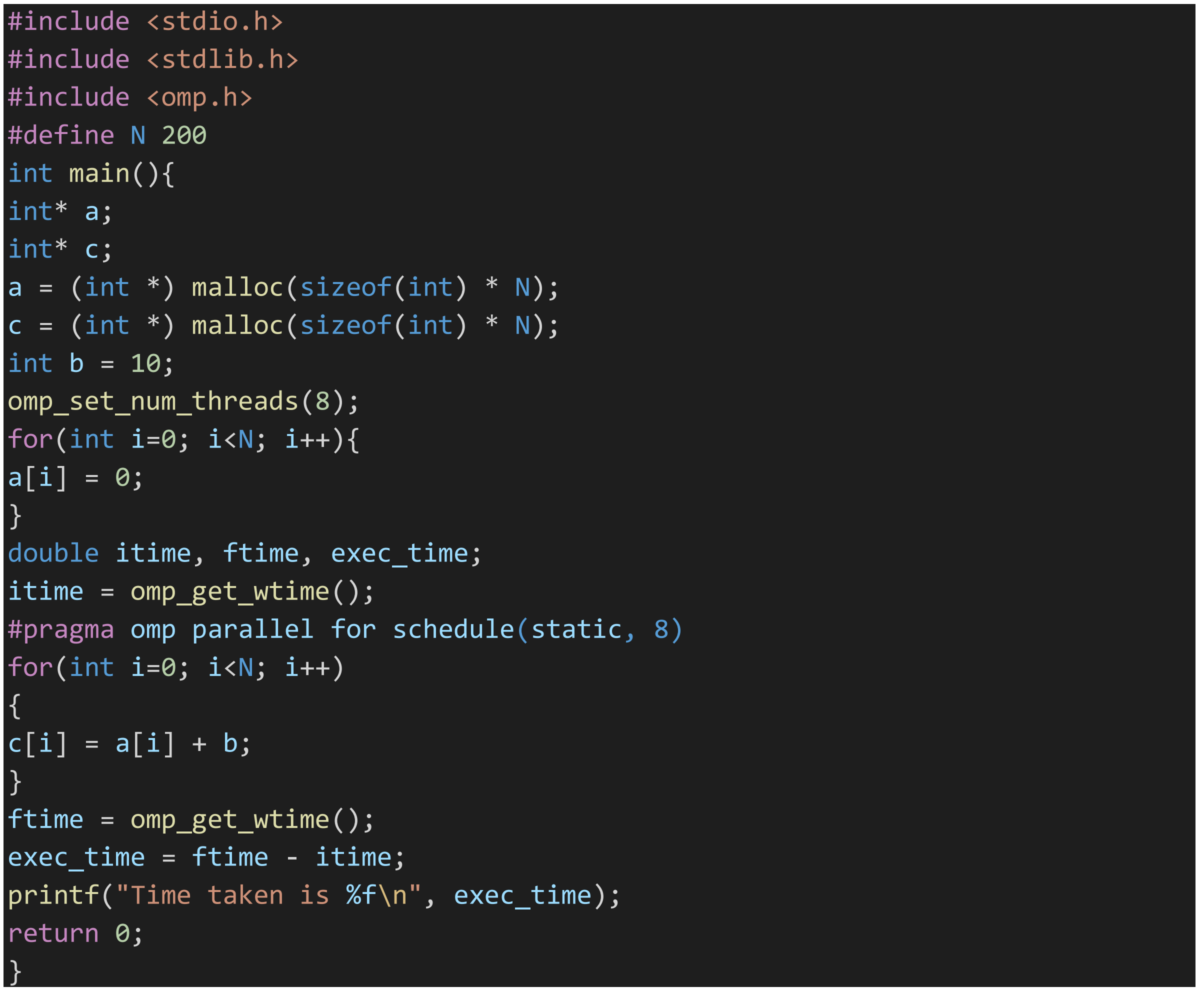
i. Use the STATIC schedule and set the loop iteration chunk size to varioussizes when changing the size of your matrix. Analyze the speedup.

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

iii. Demonstrate the use of nowait clause.

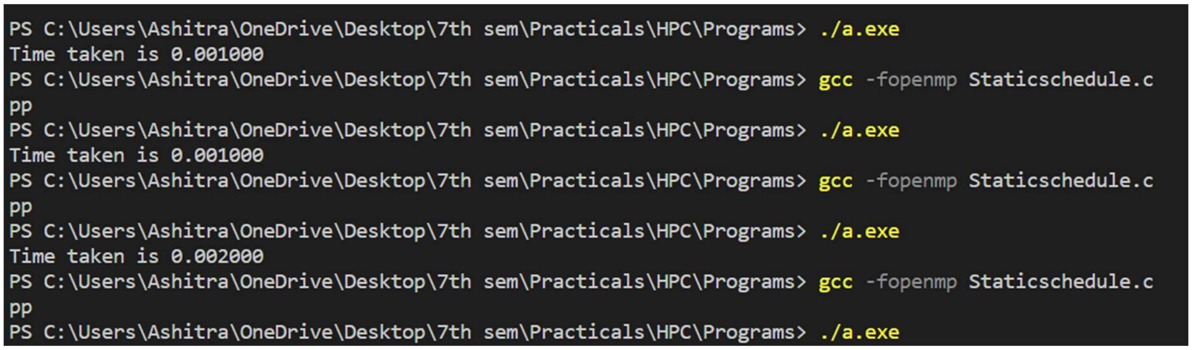
→

Static Schedule:



```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 200
int main(){
int* a;
int* c;
a = (int *) malloc(sizeof(int) * N);
c = (int *) malloc(sizeof(int) * N);
int b = 10;
omp_set_num_threads(8);
for(int i=0; i<N; i++){
a[i] = 0;
}
double itime, ftime, exec_time;
itime = omp_get_wtime();
#pragma omp parallel for schedule(static, 8)
for(int i=0; i<N; i++)
{
c[i] = a[i] + b;
}
ftime = omp_get_wtime();
exec_time = ftime - itime;
printf("Time taken is %f\n", exec_time);
return 0;
}
```

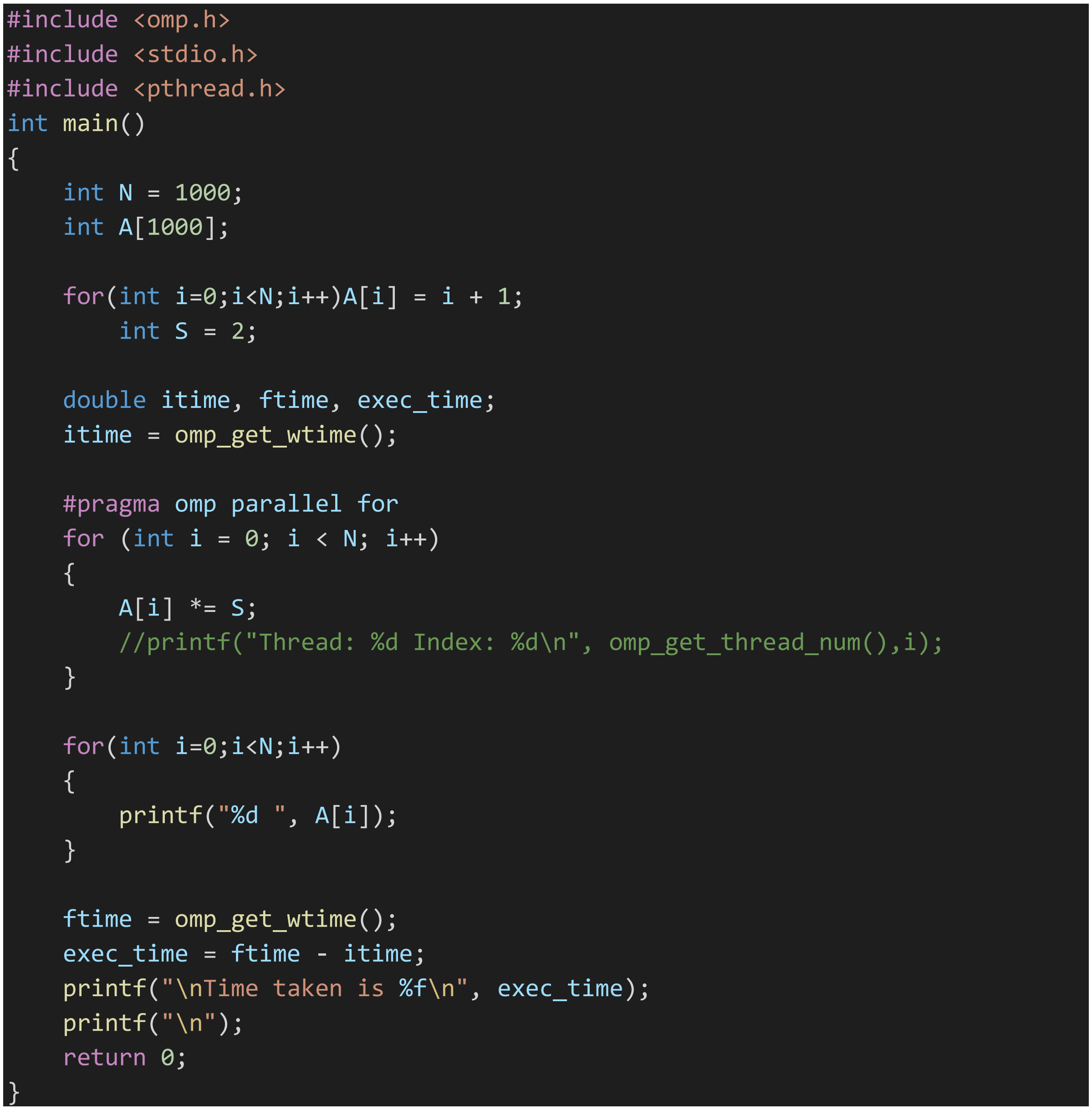
Output:



```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken is 0.001000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp Staticschedule.c
pp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken is 0.001000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp Staticschedule.c
pp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken is 0.002000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp Staticschedule.c
pp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
```

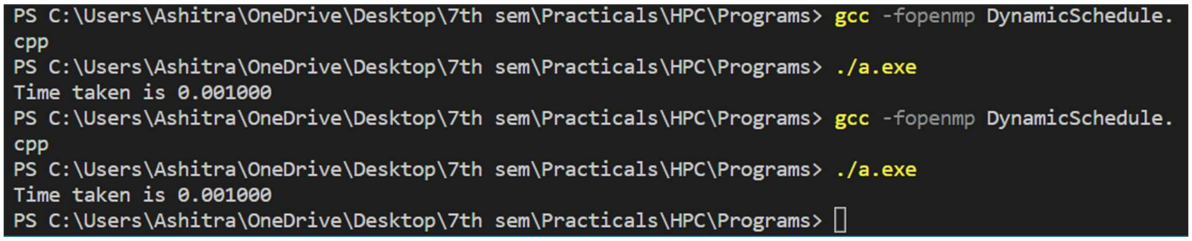
Chunk Size	**2**	**4**	**6**	**8**
Time	**0.001000**	**0.000268**	**0.000246**	**0.000279**

Dynamic Schedule:



```
#include <omp.h>
#include <stdio.h>
#include <pthread.h>
int main()
{
    int N = 1000;
    int A[1000];

    for(int i=0;i<N;i++)A[i] = i + 1;
        int S = 2;

    double itime, ftime, exec_time;
    itime = omp_get_wtime();

    #pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        A[i] *= S;
        //printf("Thread: %d Index: %d\n", omp_get_thread_num(),i);
    }

    for(int i=0;i<N;i++)
    {
        printf("%d ", A[i]);
    }

    ftime = omp_get_wtime();
    exec_time = ftime - itime;
    printf("\nTime taken is %f\n", exec_time);
    printf("\n");
    return 0;
}
```

Output:



```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp DynamicSchedule.
cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken is 0.001000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp DynamicSchedule.
cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Time taken is 0.001000
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> 
```

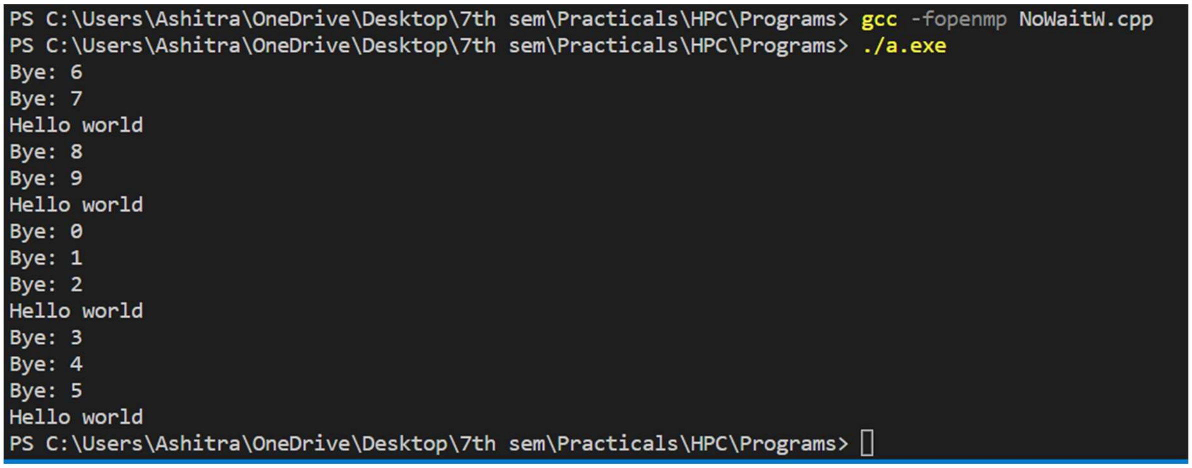
Chunk Size	2	4	6	8
Time	0.000589	0.000265	0.000275	0.000248

ii) Nowait clause

Code: With Nowait clause

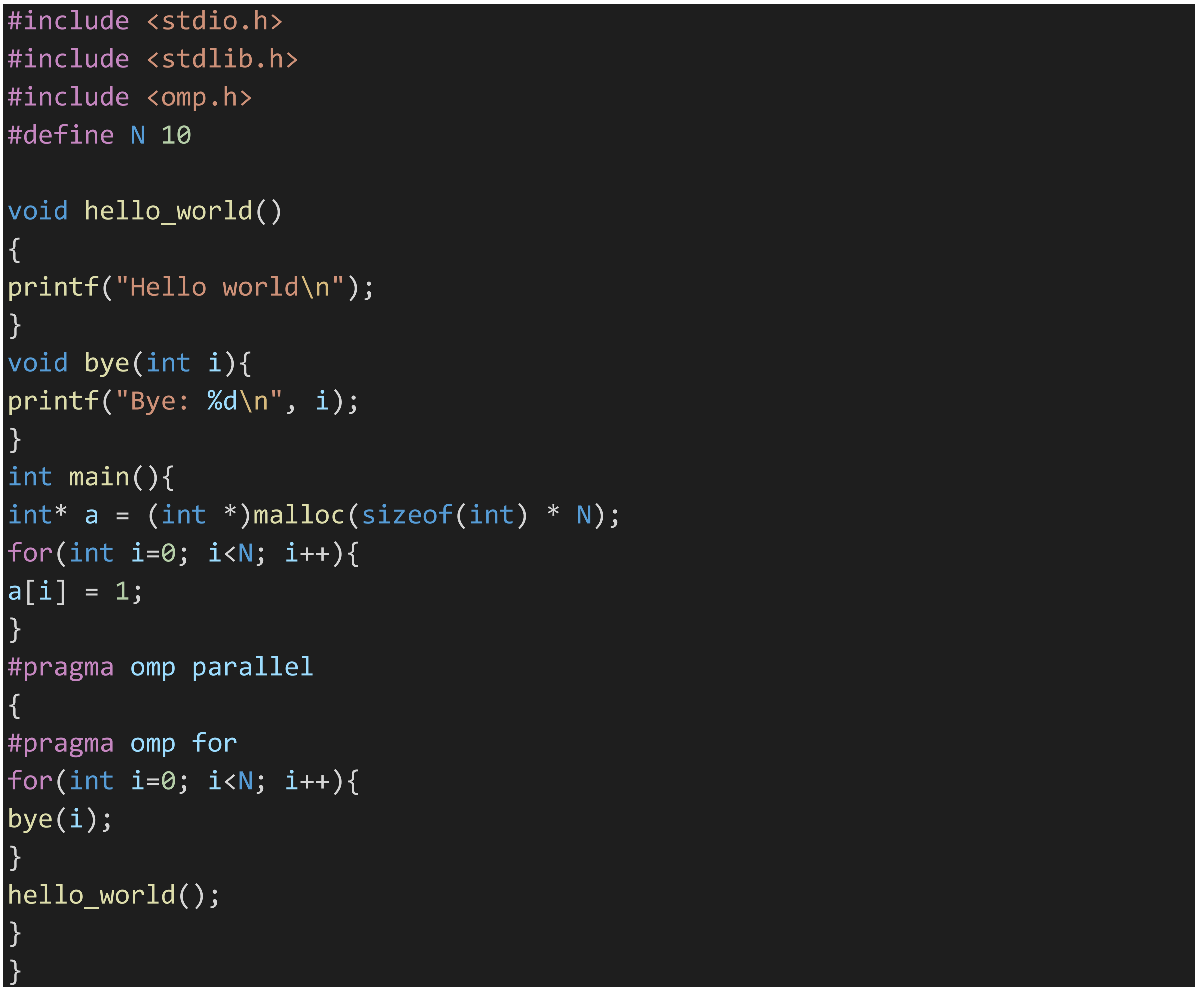
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 10

void hello_world()
{
printf("Hello world\n");
}
void bye(int i){
printf("Bye: %d\n", i);
}
int main(){
int* a = (int *)malloc(sizeof(int) * N);
for(int i=0; i<N; i++){
a[i] = 1;
}
#pragma omp parallel
{
#pragma omp for nowait
for(int i=0; i<N; i++){
bye(i);
}
hello_world();
}
}
```

Output:



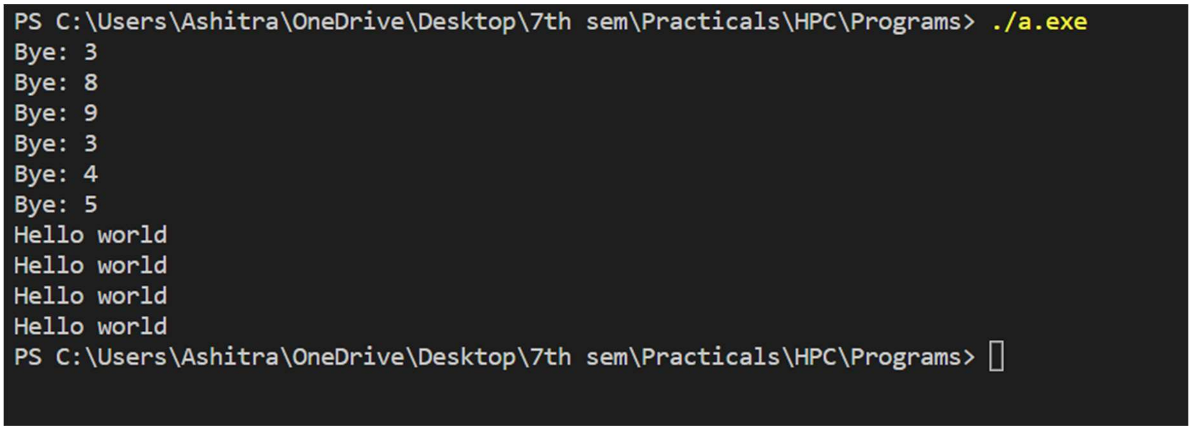
```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> gcc -fopenmp NoWaitW.cpp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Bye: 6
Bye: 7
Hello world
Bye: 8
Bye: 9
Hello world
Bye: 0
Bye: 1
Bye: 2
Hello world
Bye: 3
Bye: 4
Bye: 5
Hello world
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> 
```

Code:
Without Nowait clause



```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 10

void hello_world()
{
printf("Hello world\n");
}
void bye(int i){
printf("Bye: %d\n", i);
}
int main(){
int* a = (int *)malloc(sizeof(int) * N);
for(int i=0; i<N; i++){
a[i] = 1;
}
#pragma omp parallel
{
#pragma omp for
for(int i=0; i<N; i++){
bye(i);
}
hello_world();
}
}
```

Output:



```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> ./a.exe
Bye: 3
Bye: 8
Bye: 9
Bye: 3
Bye: 4
Bye: 5
Hello world
Hello world
Hello world
Hello world
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\HPC\Programs> 
```