Final Year B. Tech., Sem VII 2022-23

High Performance Computing Lab

PRN: 2019BTECS00036

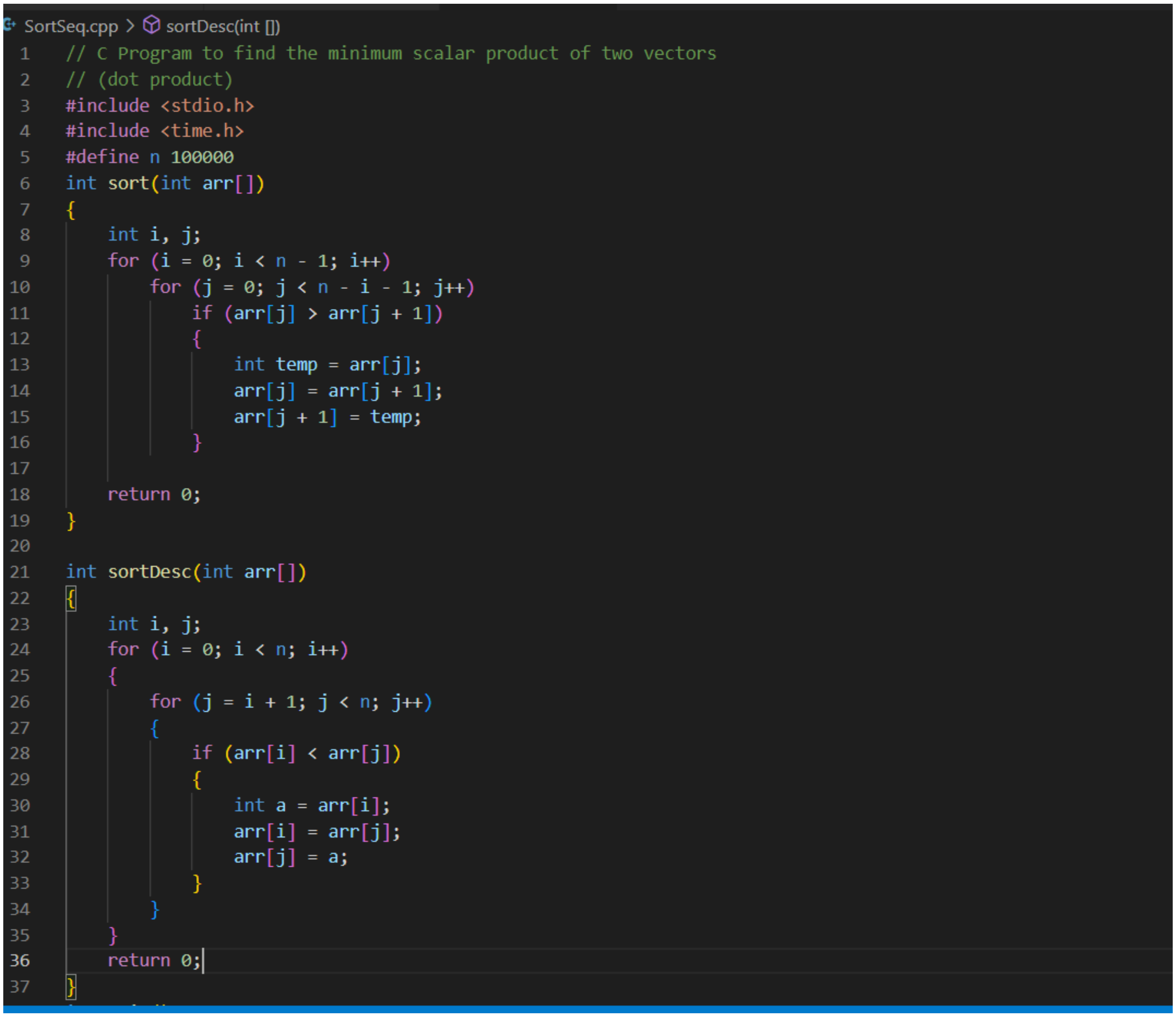
Full Name: Nikhil Danapgol

Batch: B2

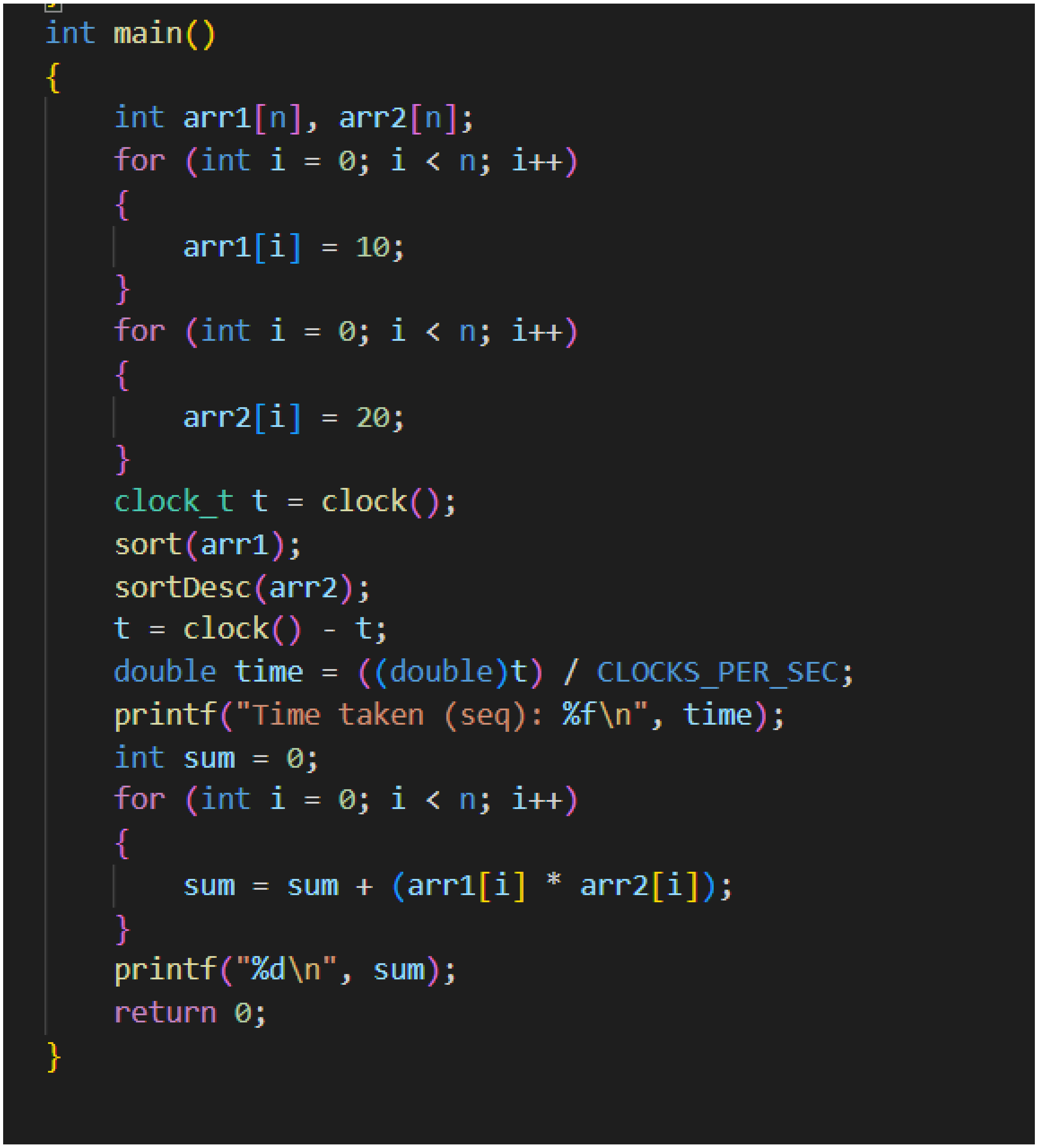
Assignment No. 4

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

Sequential:

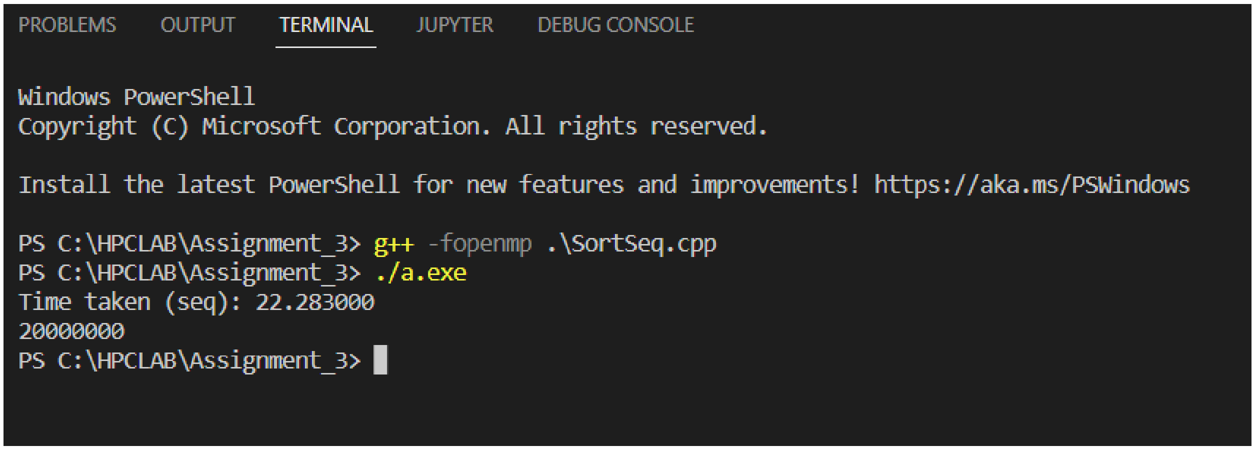


```
// C Program to find the minimum scalar product of two vectors
// (dot product)
#include <stdio.h>
#include <time.h>
#define n 100000
int sort(int arr[])
{
    int i, j;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }

    return 0;
}

int sortDesc(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;
            }
        }
    }
    return 0;
}
```



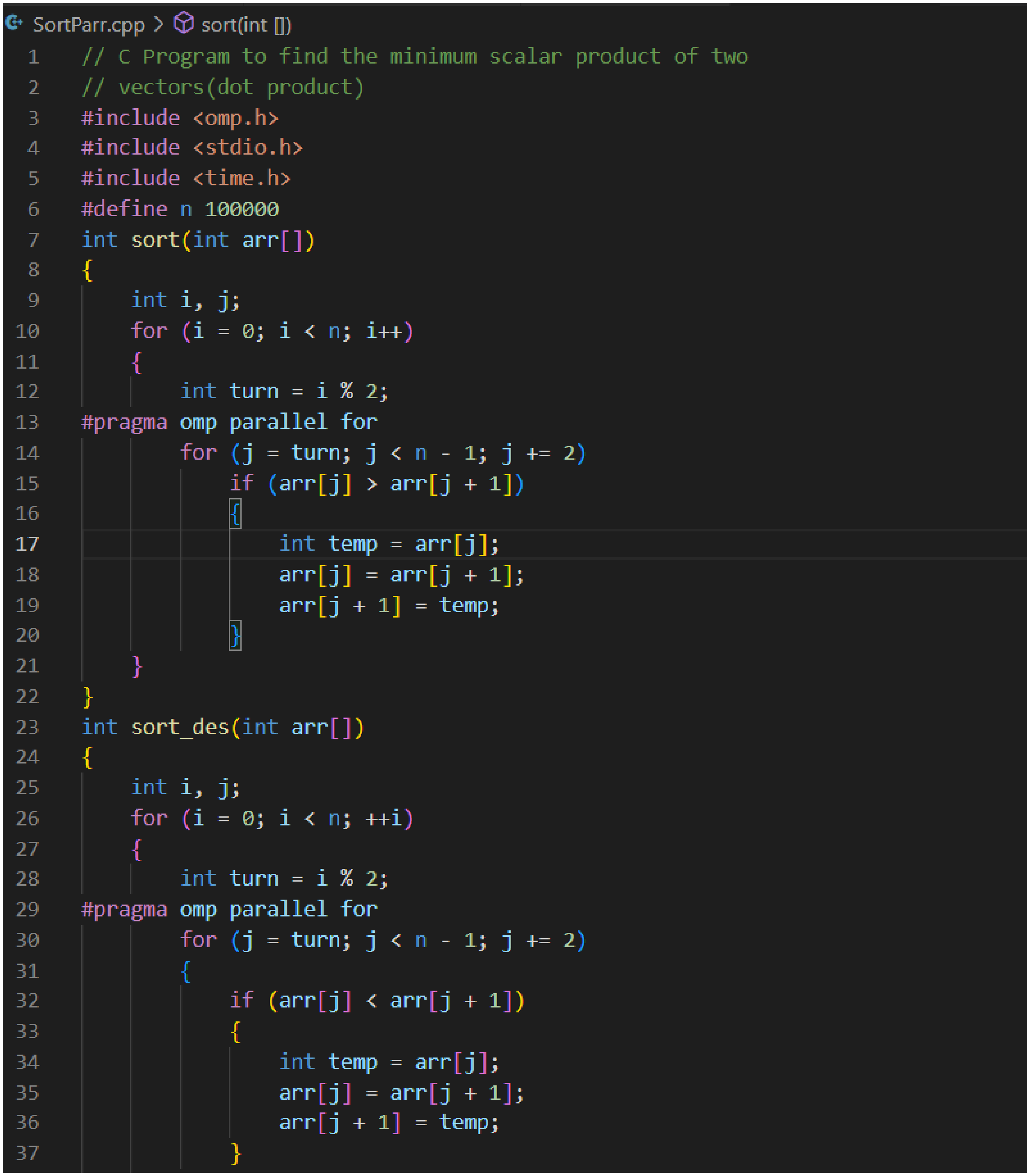
```
int main()
{
    int arr1[n], arr2[n];
    for (int i = 0; i < n; i++)
    {
        arr1[i] = 10;
    }
    for (int i = 0; i < n; i++)
    {
        arr2[i] = 20;
    }
    clock_t t = clock();
    sort(arr1);
    sortDesc(arr2);
    t = clock() - t;
    double time = ((double)t) / CLOCKS_PER_SEC;
    printf("Time taken (seq): %f\n", time);
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("%d\n", sum);
    return 0;
}
```

Output:

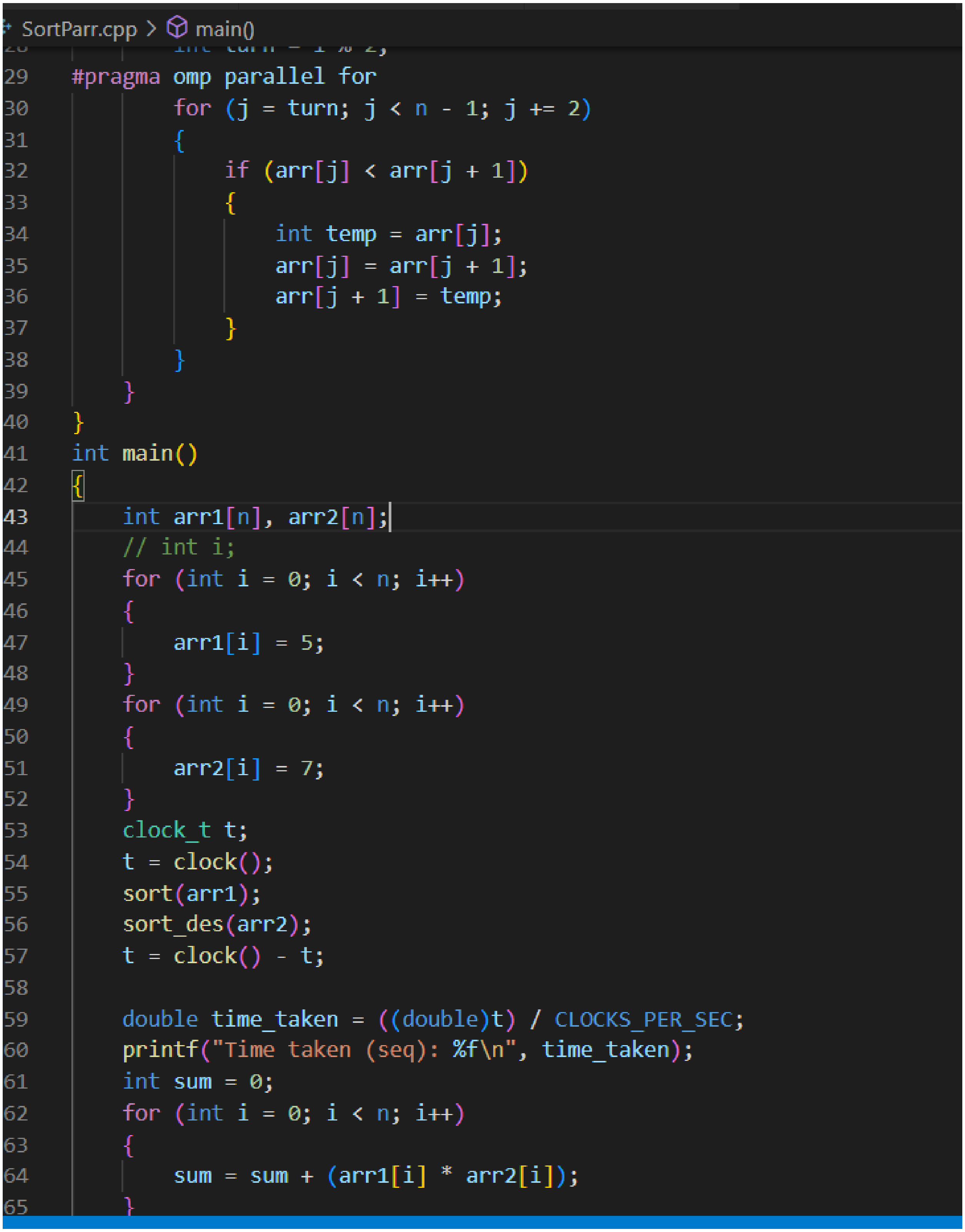


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\HPCLAB\Assignment_3> g++ -fopenmp .\SortSeq.cpp
PS C:\HPCLAB\Assignment_3> ./a.exe
Time taken (seq): 22.283000
20000000
PS C:\HPCLAB\Assignment_3> 
```

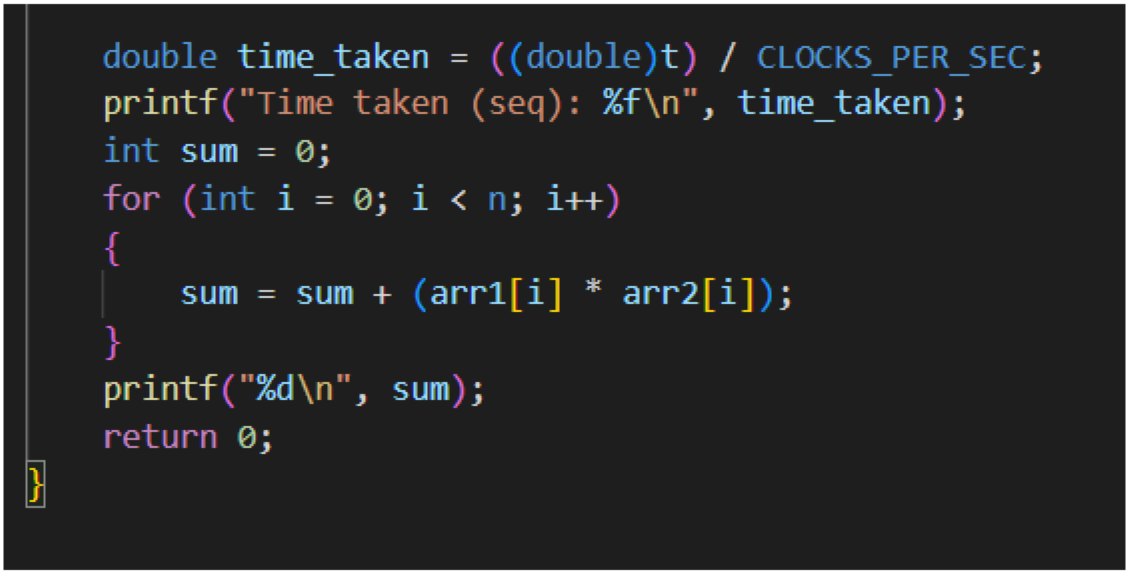
Parallel:



```
// C Program to find the minimum scalar product of two
// vectors(dot product)
#include <omp.h>
#include <stdio.h>
#include <time.h>
#define n 100000
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)
        {
            if (arr[j] < arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
```

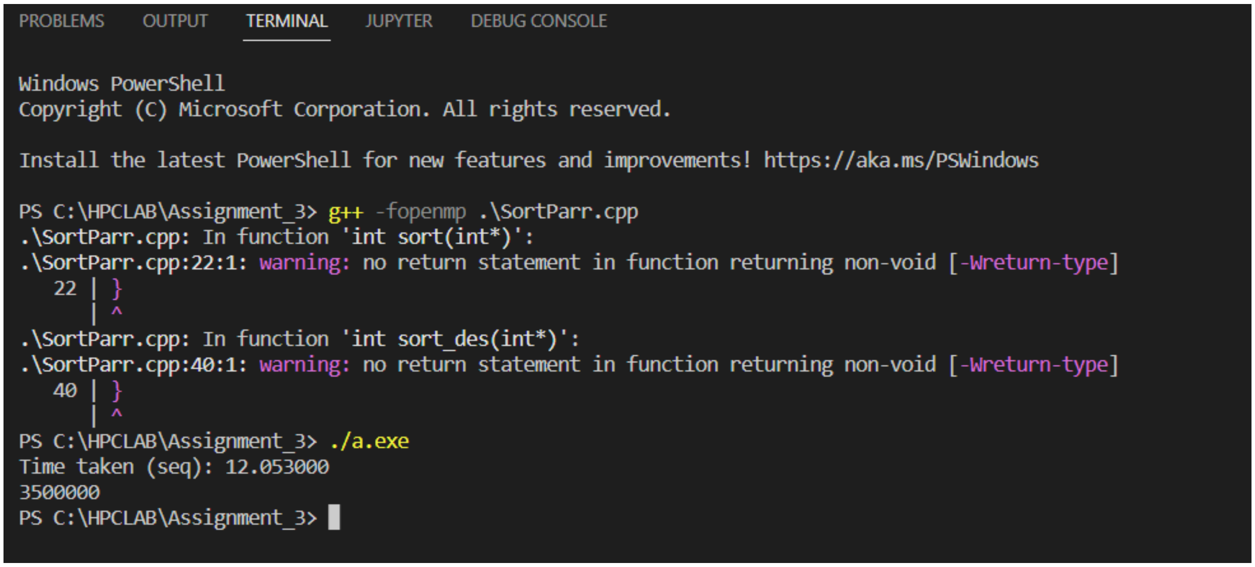


```
        int turn = i % 2;
#pragma omp parallel for
        for (j = turn; j < n - 1; j += 2)
        {
            if (arr[j] < arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
int main()
{
    int arr1[n], arr2[n];
    // int i;
    for (int i = 0; i < n; i++)
    {
        arr1[i] = 5;
    }
    for (int i = 0; i < n; i++)
    {
        arr2[i] = 7;
    }
    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);
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
```



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

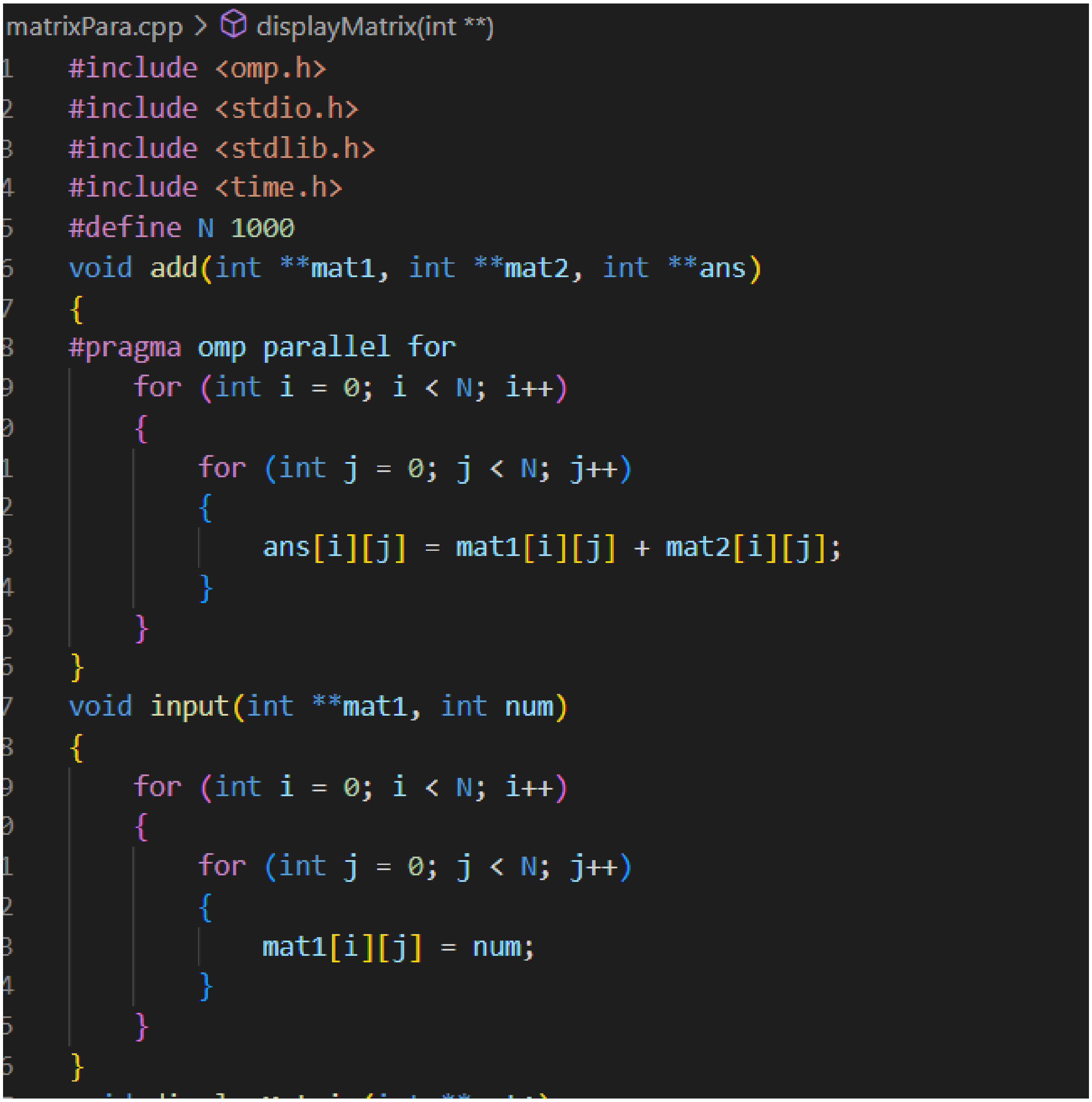
Output:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\HPCLAB\Assignment_3> g++ -fopenmp .\SortParr.cpp
.\SortParr.cpp: In function 'int sort(int*)':
.\SortParr.cpp:22:1: warning: no return statement in function returning non-void [-Wreturn-type]
   22 | }
      | ^
.\SortParr.cpp: In function 'int sort_des(int*)':
.\SortParr.cpp:40:1: warning: no return statement in function returning non-void [-Wreturn-type]
   40 | }
      | ^
PS C:\HPCLAB\Assignment_3> ./a.exe
Time taken (seq): 12.053000
3500000
PS C:\HPCLAB\Assignment_3> █
```

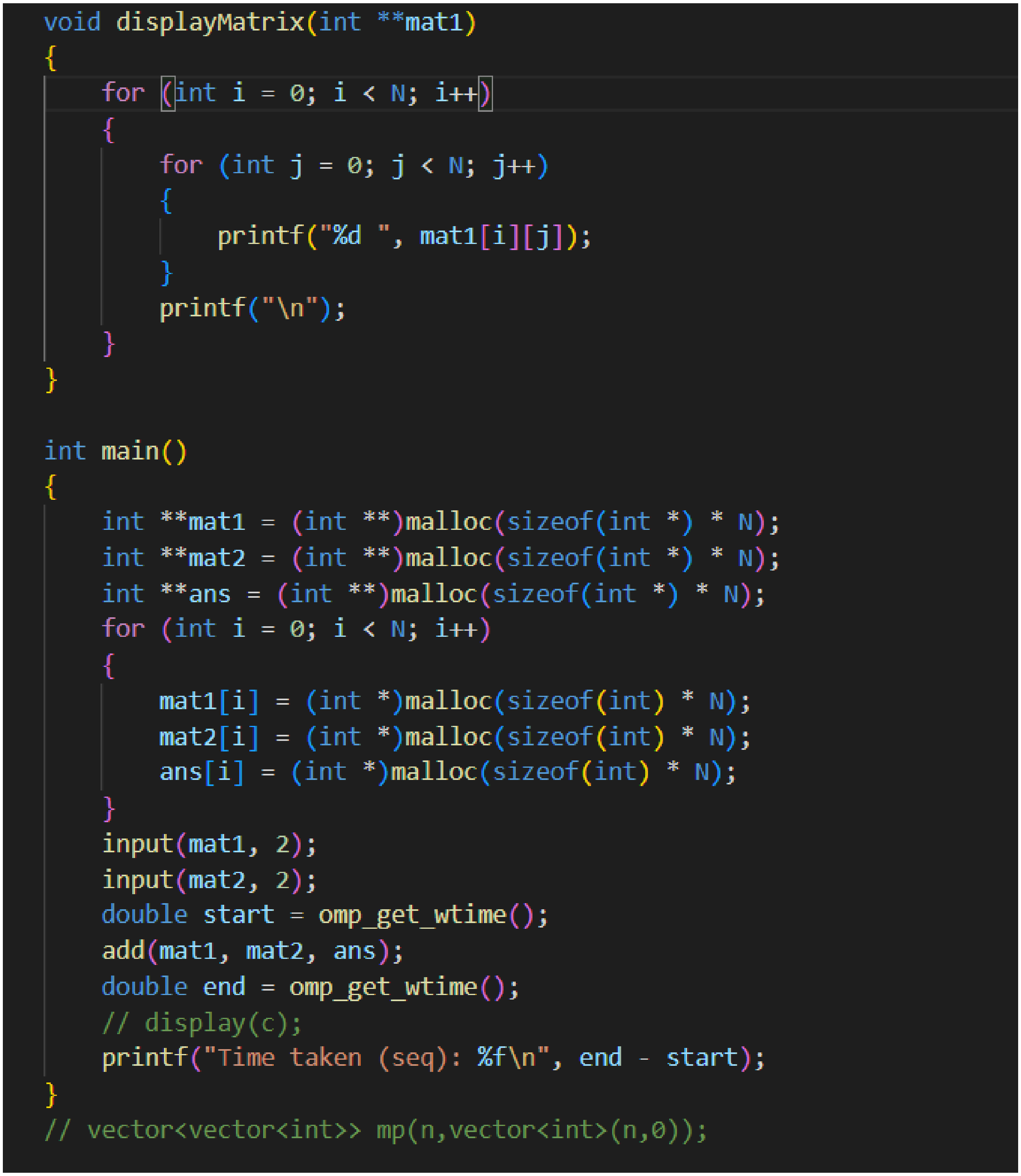
Q2: Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculating the execution time or use GPROF)

Parallel:



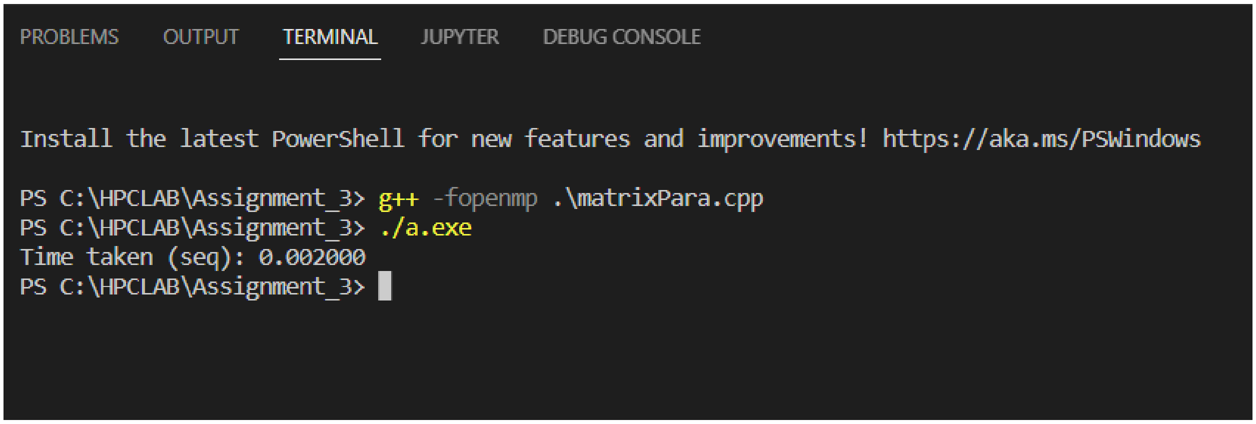
matrixPara.cpp > displayMatrix(int **)

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 1000
void add(int **mat1, int **mat2, int **ans)
{
#pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            ans[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}
void input(int **mat1, int num)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            mat1[i][j] = num;
        }
    }
}
```



```c
void displayMatrix(int **mat1)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", mat1[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int **mat1 = (int **)malloc(sizeof(int *) * N);
    int **mat2 = (int **)malloc(sizeof(int *) * N);
    int **ans = (int **)malloc(sizeof(int *) * N);
    for (int i = 0; i < N; i++)
    {
        mat1[i] = (int *)malloc(sizeof(int) * N);
        mat2[i] = (int *)malloc(sizeof(int) * N);
        ans[i] = (int *)malloc(sizeof(int) * N);
    }
    input(mat1, 2);
    input(mat2, 2);
    double start = omp_get_wtime();
    add(mat1, mat2, ans);
    double end = omp_get_wtime();
    // display(c);
    printf("Time taken (seq): %f\n", end - start);
}
// vector<vector<int>> mp(n,vector<int>(n,0));
```

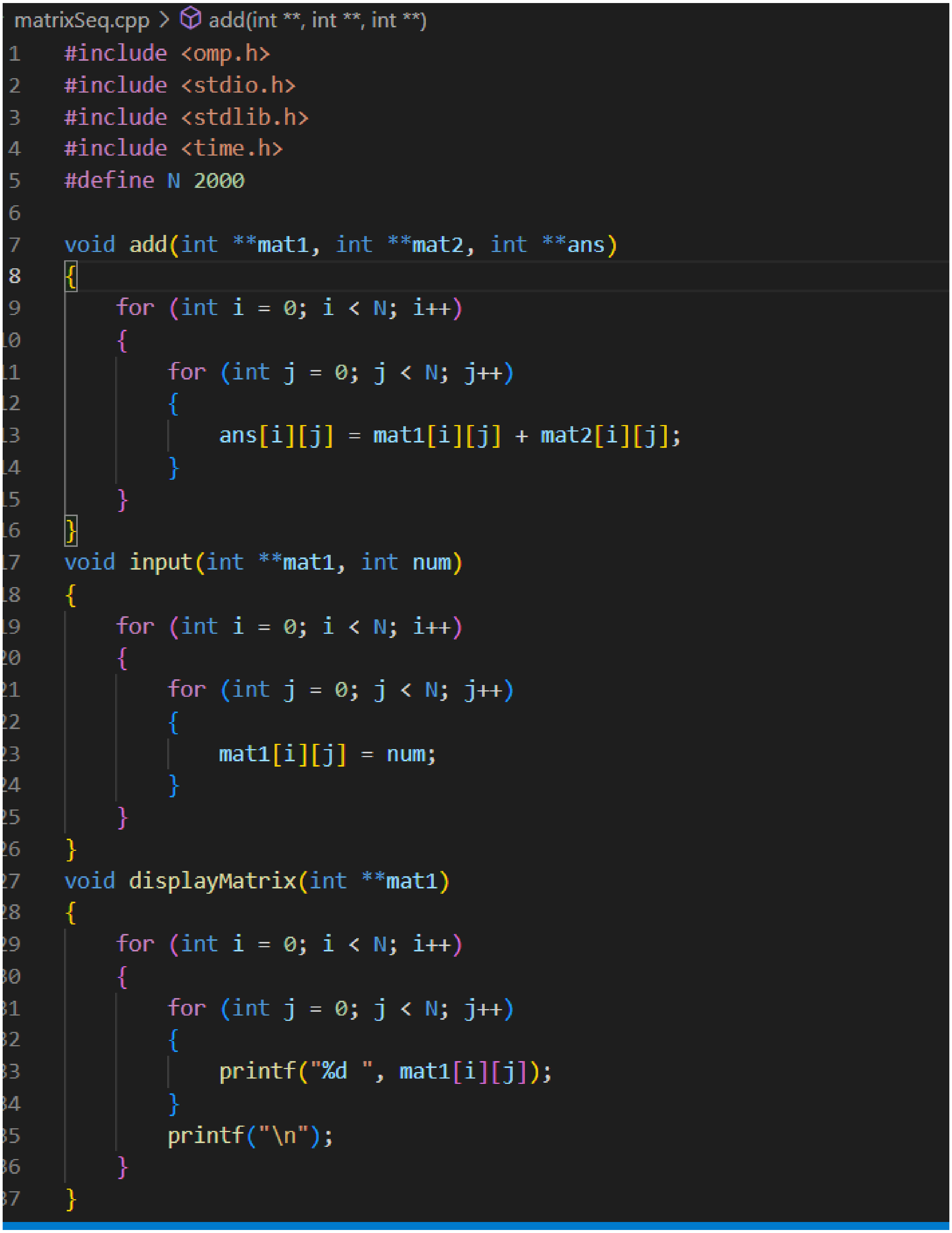
Output:



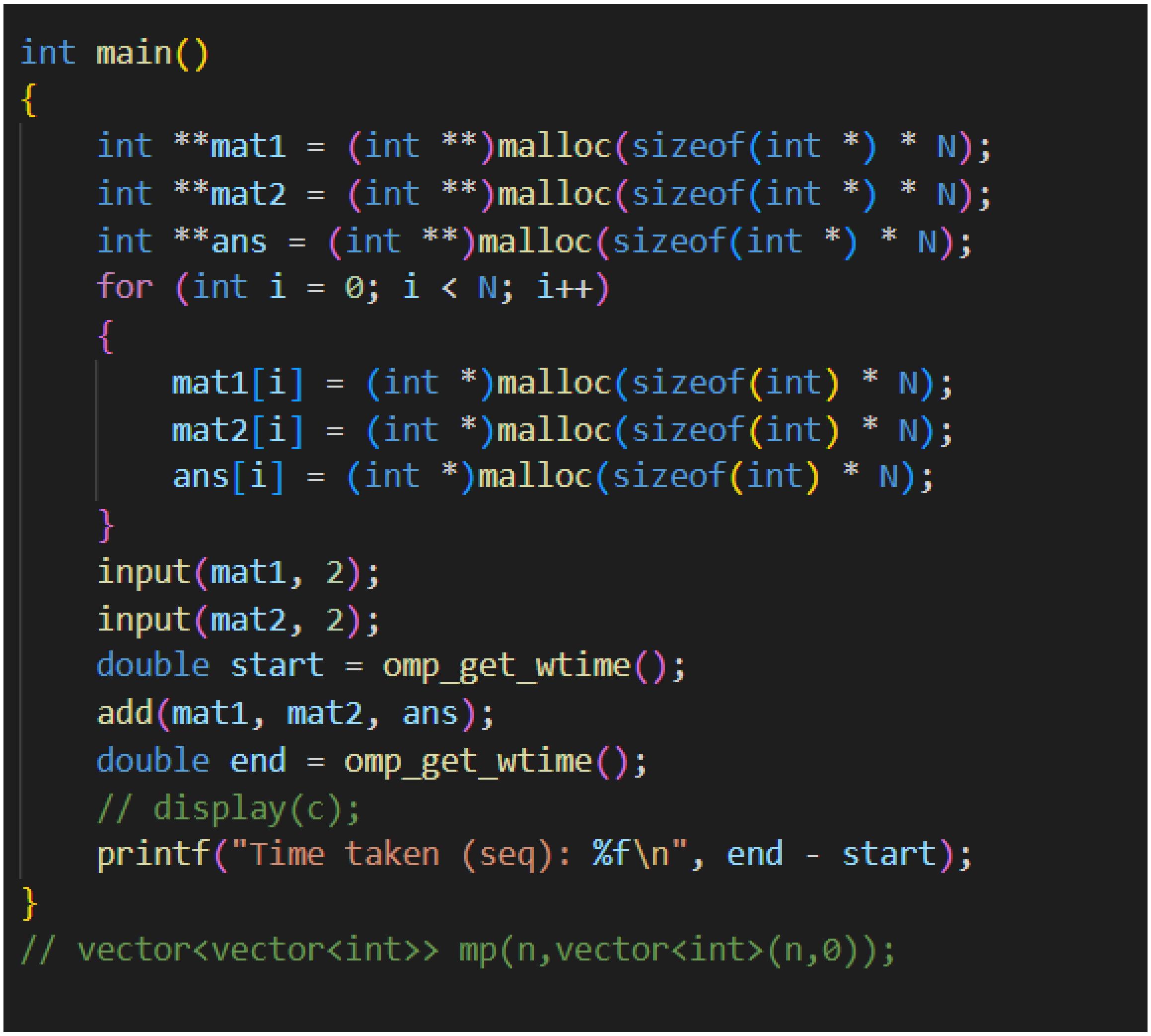
```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\HPCLAB\Assignment_3> g++ -fopenmp .\matrixPara.cpp
PS C:\HPCLAB\Assignment_3> ./a.exe
Time taken (seq): 0.002000
PS C:\HPCLAB\Assignment_3>
```

Threads/N:	**250**	**500**	**750**	**1000**	**2000**
2	**0.001000**	**0.001000**	**0.002000**	**0.004000**	**0.015000**
4	**0.001000**	**0.001000**	**0.002000**	**0.003000**	**0.007000**
6	**0.001000**	**0.002000**	**0.002000**	**0.003000**	**0.010000**
8	**0.001000**	**0.002000**	**0.003000**	**0.003000**	**0.009000**

Sequential:

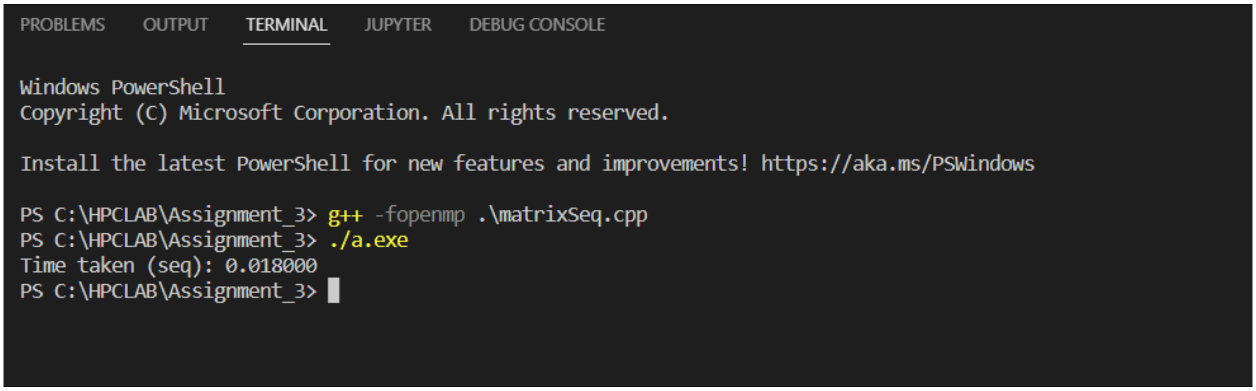


```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 2000

void add(int **mat1, int **mat2, int **ans)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            ans[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}
void input(int **mat1, int num)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            mat1[i][j] = num;
        }
    }
}
void displayMatrix(int **mat1)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", mat1[i][j]);
        }
        printf("\n");
    }
}
```



```
int main()
{
    int **mat1 = (int **)malloc(sizeof(int *) * N);
    int **mat2 = (int **)malloc(sizeof(int *) * N);
    int **ans = (int **)malloc(sizeof(int *) * N);
    for (int i = 0; i < N; i++)
    {
        mat1[i] = (int *)malloc(sizeof(int) * N);
        mat2[i] = (int *)malloc(sizeof(int) * N);
        ans[i] = (int *)malloc(sizeof(int) * N);
    }
    input(mat1, 2);
    input(mat2, 2);
    double start = omp_get_wtime();
    add(mat1, mat2, ans);
    double end = omp_get_wtime();
    // display(c);
    printf("Time taken (seq): %f\n", end - start);
}
// vector<vector<int>> mp(n,vector<int>(n,0));
```

Output:



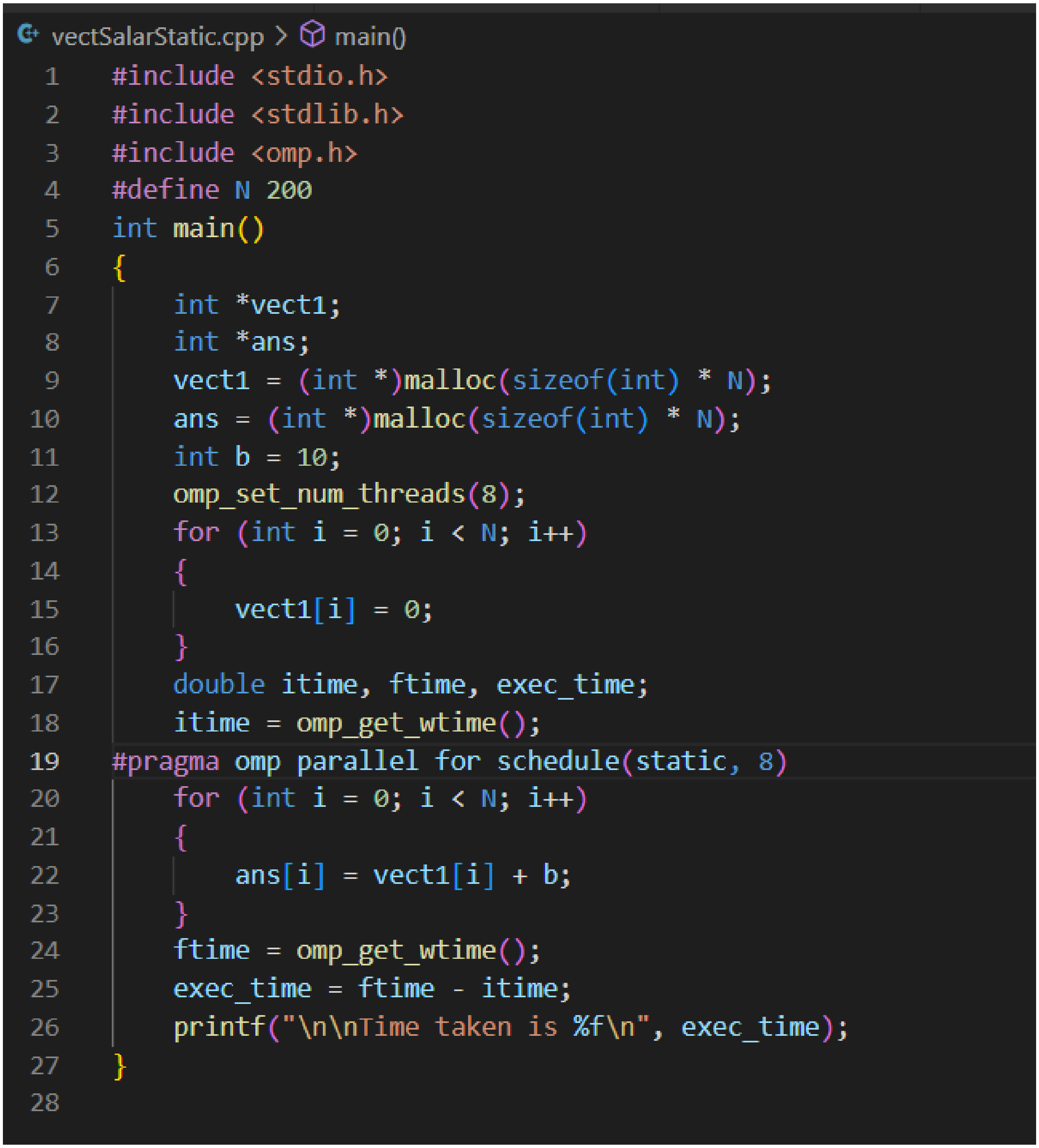
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\HPCLAB\Assignment_3> g++ -fopenmp .\matrixSeq.cpp
PS C:\HPCLAB\Assignment_3> ./a.exe
Time taken (seq): 0.018000
PS C:\HPCLAB\Assignment_3> █
```

N	**250**	**500**	**750**	**1000**	**2000**
Time	**0.000000**	**0.002000**	**0.002000**	**0.003000**	**0.020000**

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.

1.Static Schedule:

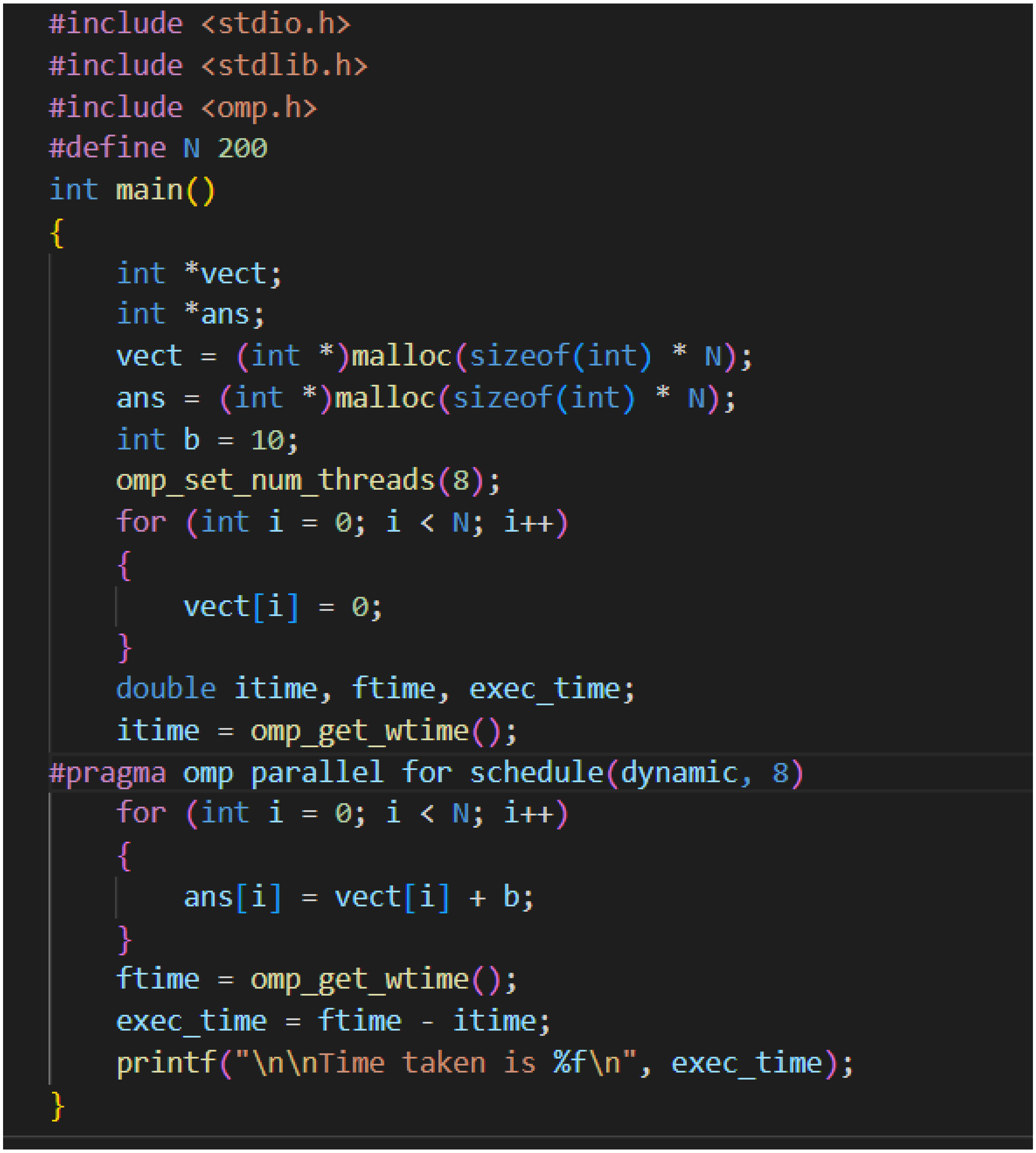


```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 200
int main()
{
    int *vect1;
    int *ans;
    vect1 = (int *)malloc(sizeof(int) * N);
    ans = (int *)malloc(sizeof(int) * N);
    int b = 10;
    omp_set_num_threads(8);
    for (int i = 0; i < N; i++)
    {
        vect1[i] = 0;
    }
    double itime, ftime, exec_time;
    itime = omp_get_wtime();
#pragma omp parallel for schedule(static, 8)
    for (int i = 0; i < N; i++)
    {
        ans[i] = vect1[i] + b;
    }
    ftime = omp_get_wtime();
    exec_time = ftime - itime;
    printf("\n\nTime taken is %f\n", exec_time);
}

```

Chunk Size: **2** **4** **6** **8**

Time:	**0.001000**	**0.001000**	**0.001000**	**0.001000**

2.Dynamic Schedule:



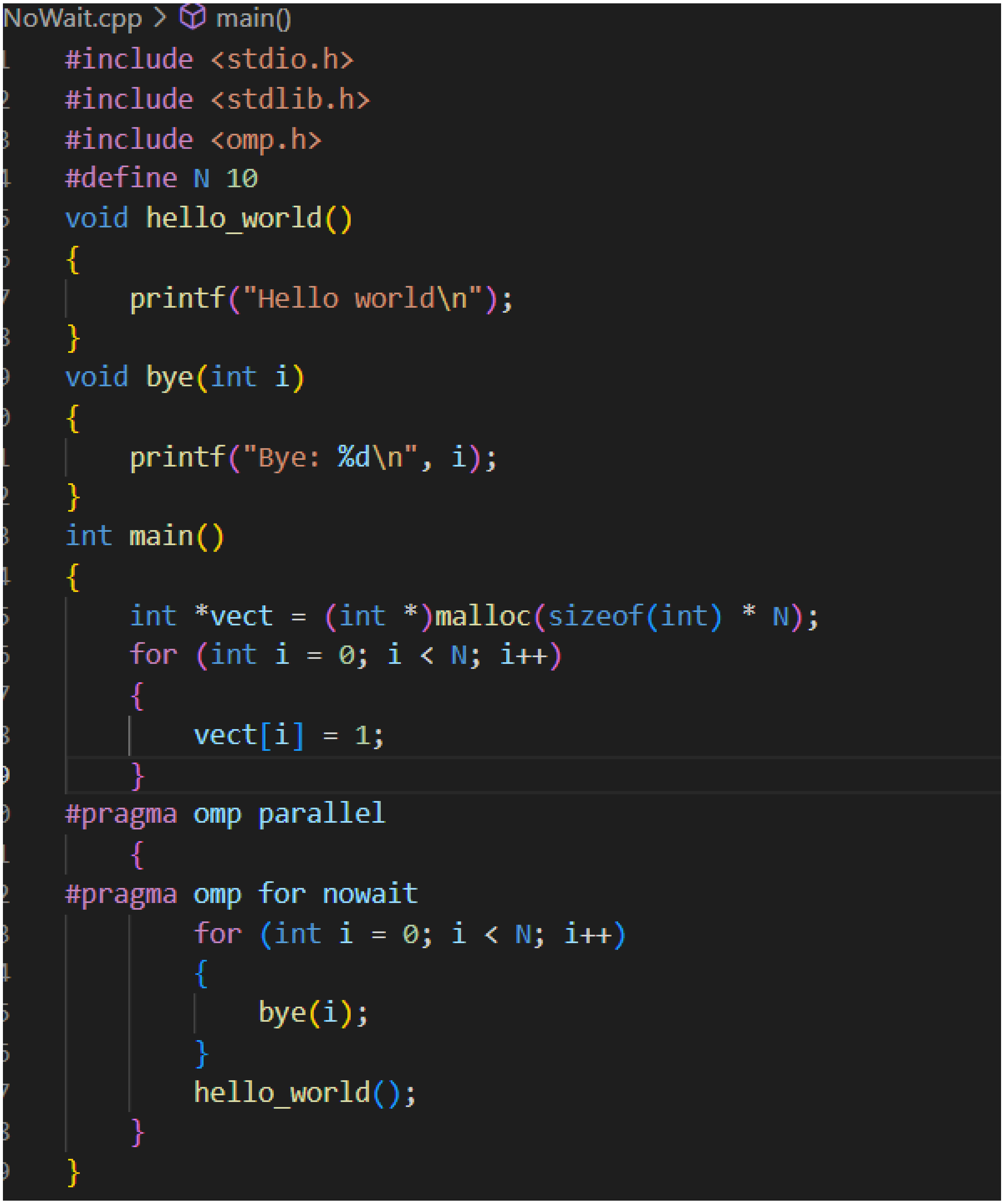
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 200
int main()
{
    int *vect;
    int *ans;
    vect = (int *)malloc(sizeof(int) * N);
    ans = (int *)malloc(sizeof(int) * N);
    int b = 10;
    omp_set_num_threads(8);
    for (int i = 0; i < N; i++)
    {
        vect[i] =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

Chunk Size:	**2**	**4**	**6**	**8**
Time:	**0.002000**	**0.001000**	**0.001000**	**0.001000**

Pros: The dynamic scheduling type is appropriate when the iterations require different computational costs. This means that the iterations are not as balance as static methods between each other.

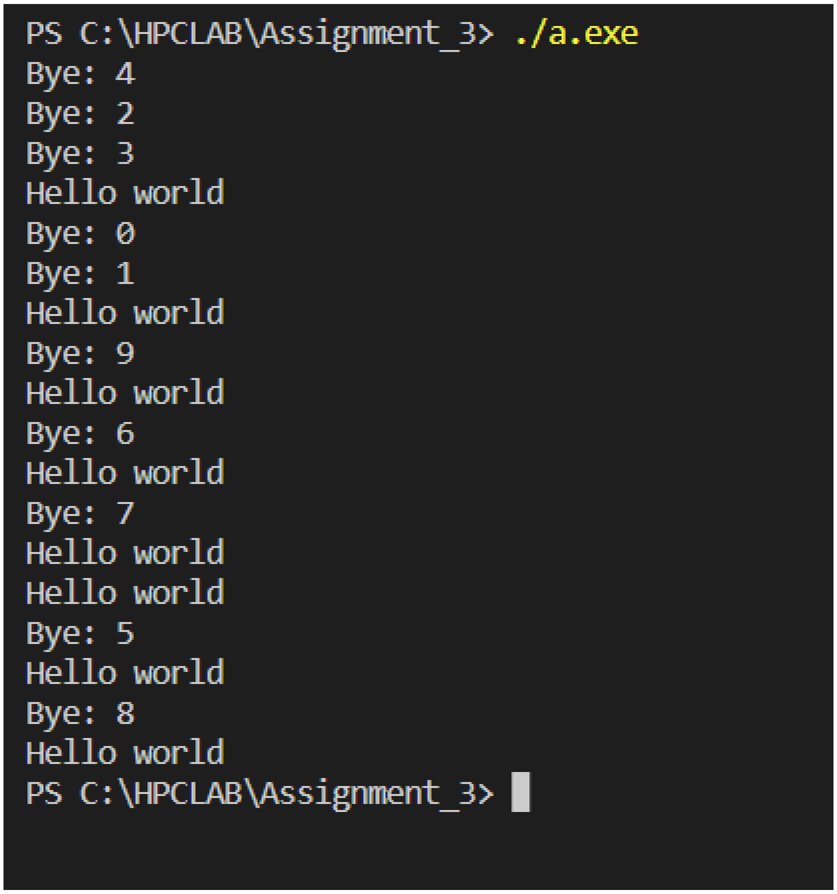
Cons: The dynamic scheduling type has higher overhead then the static scheduling type because it dynamically distributes the iterations during the runtime.

3.Nowait Clause:



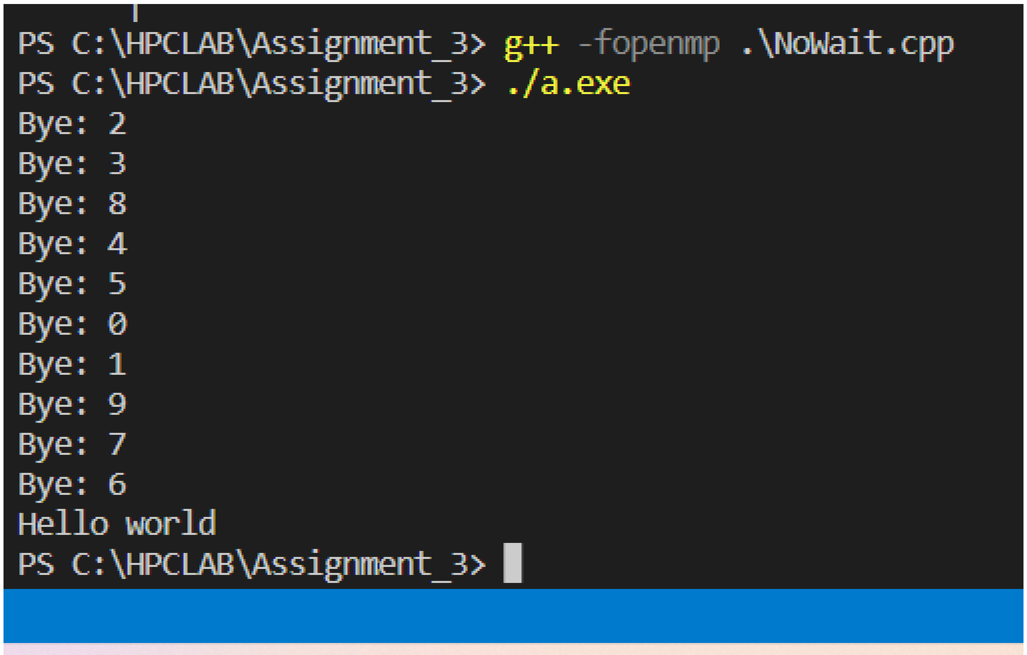
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 10
void hello_world()
{
    printf("Hello world\n");
}
void bye(int i)
{
    printf("Bye: %d\n", i);
}
int main()
{
    int *vect = (int *)malloc(sizeof(int) * N);
    for (int i = 0; i < N; i++)
    {
        vect[i] = 1;
    }
#pragma omp parallel
    {
#pragma omp for nowait
        for (int i = 0; i < N; i++)
        {
            bye(i);
        }
        hello_world();
    }
}
```

Output:



```
PS C:\HPCLAB\Assignment_3> ./a.exe
Bye: 4
Bye: 2
Bye: 3
Hello world
Bye: 0
Bye: 1
Hello world
Bye: 9
Hello world
Bye: 6
Hello world
Bye: 7
Hello world
Hello world
Bye: 5
Hello world
Bye: 8
Hello world
PS C:\HPCLAB\Assignment_3>
```

Without NoWait Clause:



```
PS C:\HPCLAB\Assignment_3> g++ -fopenmp .\NoWait.cpp
PS C:\HPCLAB\Assignment_3> ./a.exe
Bye: 2
Bye: 3
Bye: 8
Bye: 4
Bye: 5
Bye: 0
Bye: 1
Bye: 9
Bye: 7
Bye: 6
Hello world
PS C:\HPCLAB\Assignment_3>
```

GitHub:

https://github.com/nd22052000/HPC