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

**High Performance Computing Lab**

**PRN/ Roll No: 2020BTECS00206**

**Full Name: SAYALI YOGESH DESAI**

**Batch: B4**

**Assignment No. 3**

**Que 1. Analyse 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;

}

}

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);

int sum = 0;

for (i = 0; i < n; i++)

{

sum = sum + (arr1[i] \* arr2[i]);

}

printf("%d\n", sum);

return 0;

}

**\*Parallel:**

// C Program to find the minimum scalar product of two vectors (dot product)

#include <stdio.h>

#include <time.h>

#include<omp.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)

{

// printf("Thread ID: ",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);

int sum = 0;

for (i = 0; i < n; i++)

{

// printf("%d %d\n", arr1[i],arr2[i]);

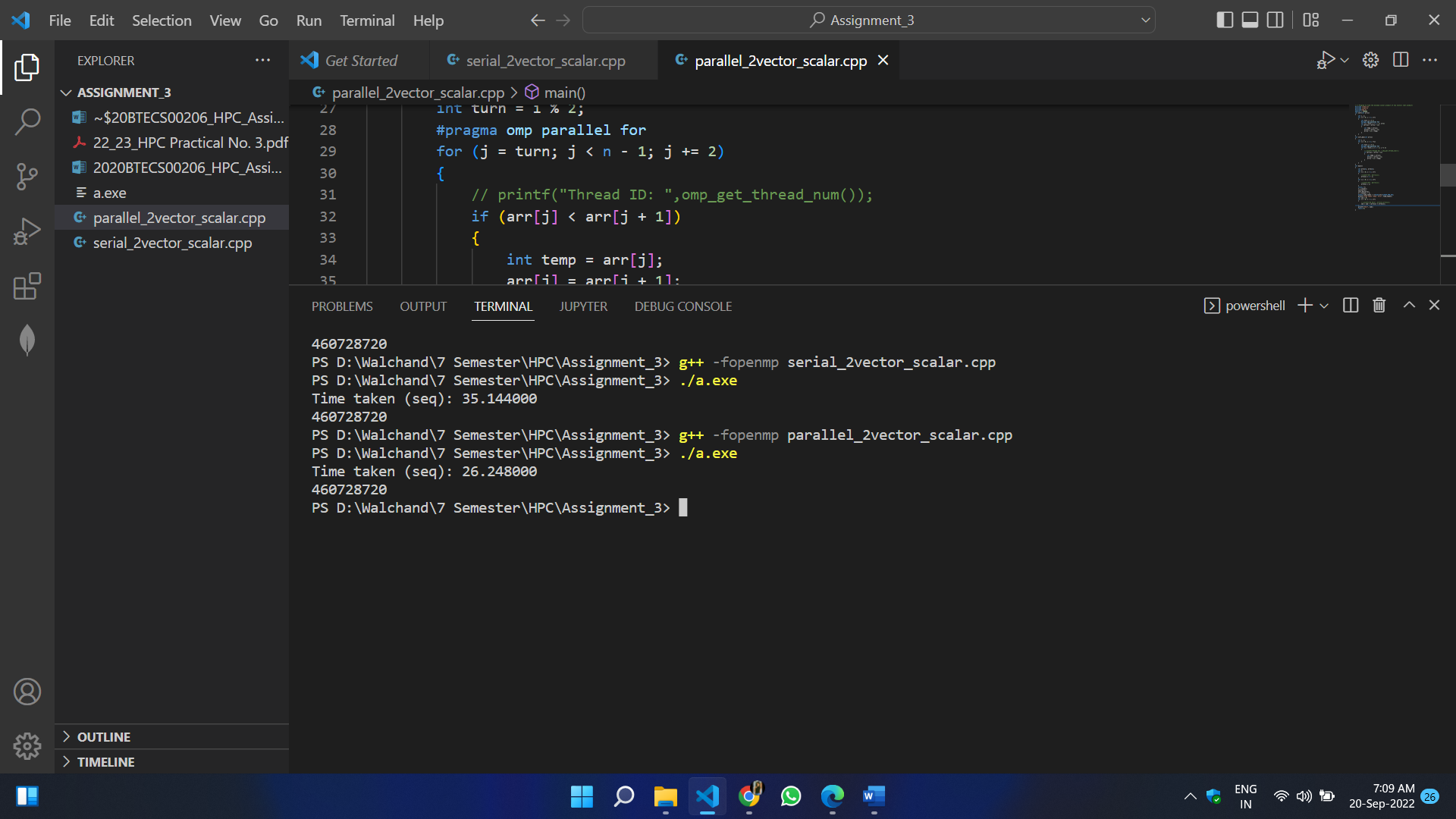
sum = sum + (arr1[i] \* arr2[i]);

}

printf("%d\n", sum);

return 0;

}



**Que 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)**

1. **For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.**
2. **Explain whether or not the scaling behaviour is as expected.**

**\*Sequential-**

* **Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 2000

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 input(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 \*\*)malloc(sizeof(int \*) \* N);

int \*\*b = (int \*\*)malloc(sizeof(int \*) \* N);

int \*\*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);

}

input(a, 1);

input(b, 1);

double start = omp\_get\_wtime();

add(a, b, c);

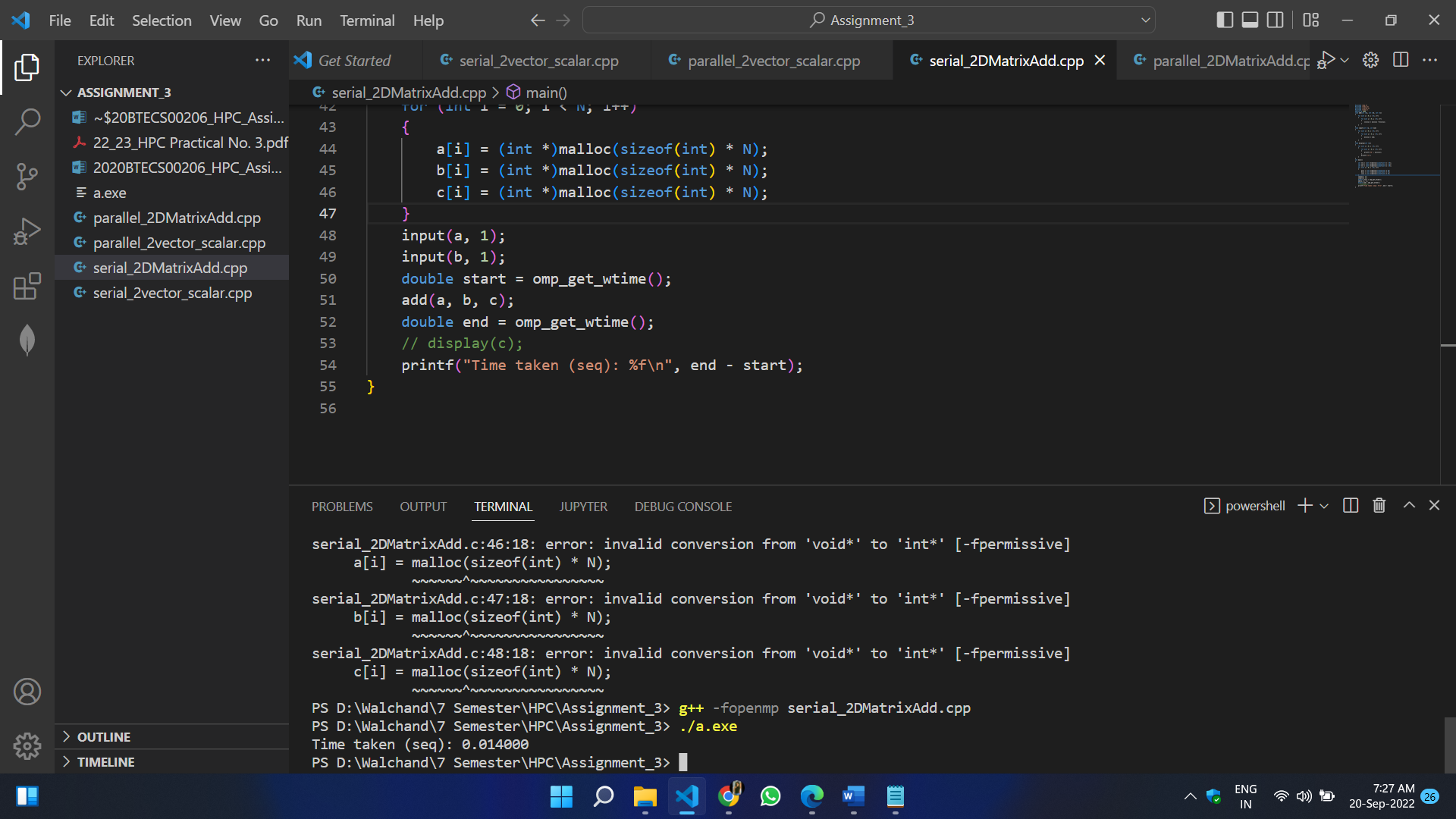
double end = omp\_get\_wtime();

// display(c);

printf("Time taken (seq): %f\n", end - start);

}

* **Output:**



**\*Parallel:**

* **Code-**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.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 input(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 \*\*)malloc(sizeof(int \*) \* N);

int \*\*b = (int \*\*)malloc(sizeof(int \*) \* N);

int \*\*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);

}

input(a, 1);

input(b, 1);

double start = omp\_get\_wtime();

add(a, b, c);

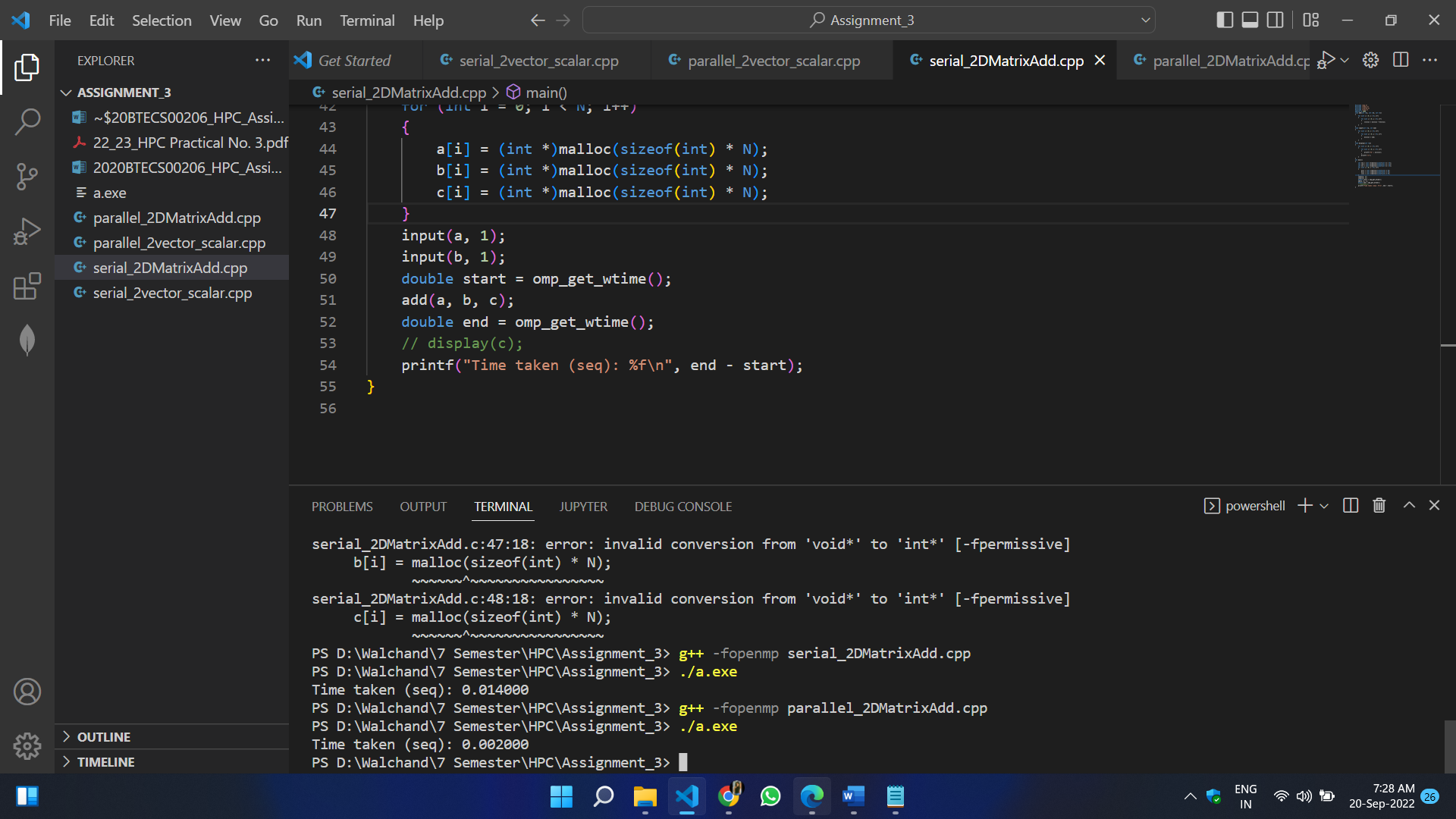
double end = omp\_get\_wtime();

// display(c);

printf("Time taken (seq): %f\n", end - start);

}

* **Output:**



**Que 3. For 1D Vector (size=200) and scalar addition, write a OpenMP code with the following:**

1. **Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyse the speedup.**
2. **Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyse the speedup.**
3. **Demonstrate the use of nowait clause**

**STATIC Schedule:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 200

int main()

{

int \*\*a=(int \*\*) malloc(sizeof(int) \* N);

int \*\*c=(int \*\*) malloc(sizeof(int) \* N);

int b = 10;

omp\_set\_num\_threads(6);

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("\n\nTime taken is %f\n", exec\_time);

}

**DYNAMIC Schedule:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 200

int main()

{

int \*\*a= (int \*\*)malloc(sizeof(int) \* N);

int \*\*c= (int \*\*)malloc(sizeof(int) \* N);

int b = 10;

omp\_set\_num\_threads(6);

for(int i=0; i<N; i++)

{

a[i] = 0;

}

double itime, ftime, exec\_time;

itime = omp\_get\_wtime();

#pragma omp parallel for schedule(dynamic, 2)

for(int i=0; i<N; i++)

{

c[i] = a[i] + b;

}

ftime = omp\_get\_wtime();

exec\_time = ftime - itime;

printf("\n\nTime taken is %f\n", exec\_time);

}

**NOWAIT Clause:**

**Code:**

#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++)

{

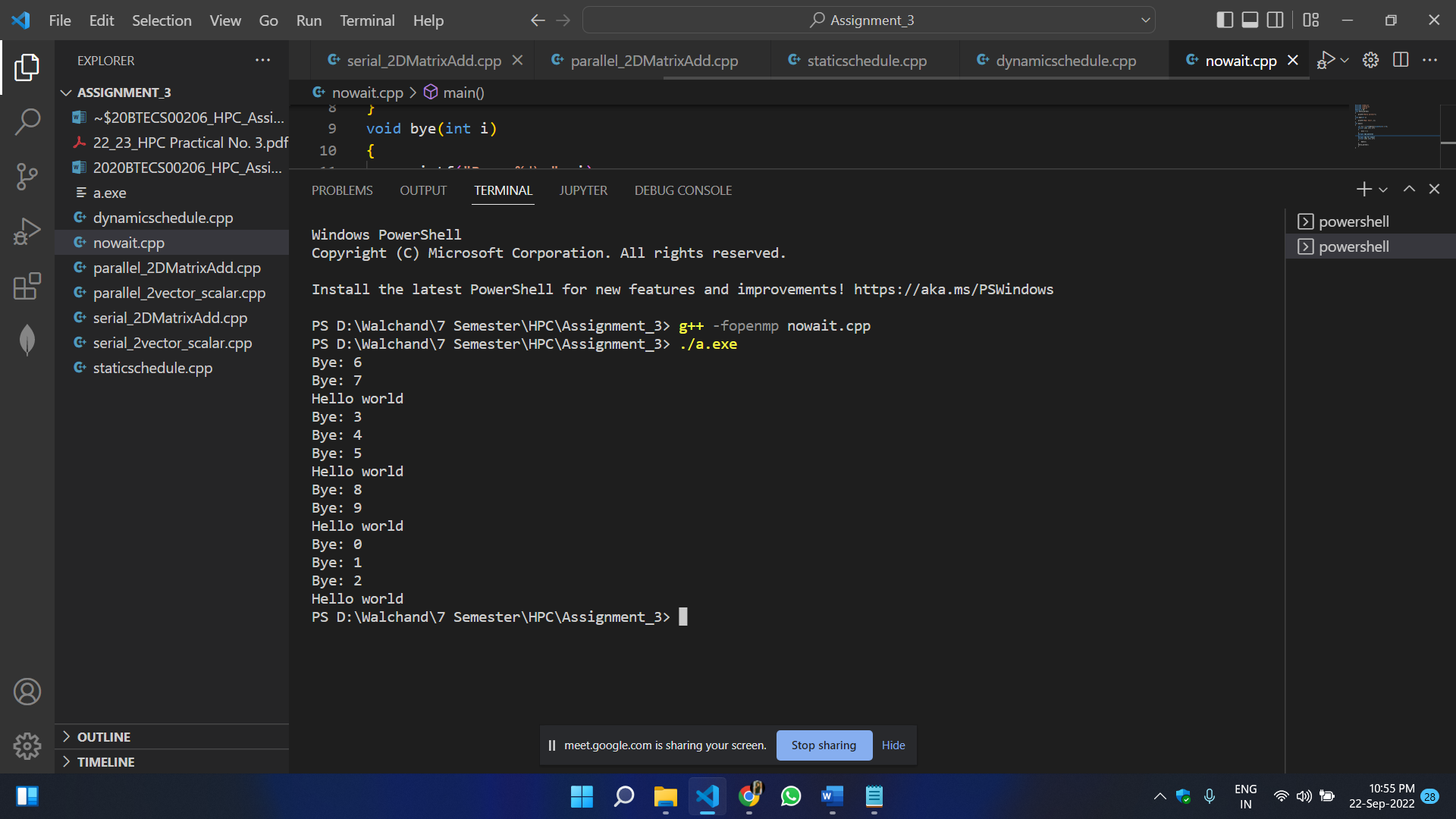
bye(i);

}

hello\_world();

}

}



**Without NOWAIT Clause:**

