**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. 2**

**Que 1. What is SPMD?**

1. SPMD stands for Single Program Multiple Data.
2. It is a parallel programming style.
3. In SPMD style, tasks are split up and run simultaneously on multiple processors with different data.
4. This is done to achieve results faster.

**Que 2. What is Worksharing?**

1. A worksharing construct distributes the execution of the corresponding region among the members of the team that encounters it.
2. Threads execute portions of the region in the context of the implicit tasks that each one is executing.
3. If the team consists of only one thread then the worksharing region is not executed in parallel.
4. A worksharing region has no barrier on entry; however, an implied barrier exists at the end of the worksharing region, unless a nowait clause is specified.
5. If a nowait clause is present, an implementation may omit the barrier at the end of the worksharing region.
6. In this case, threads that finish early may proceed straight to the instructions that follow the worksharing region without waiting for the other members of the team to finish the worksharing region, and without performing a flush operation.
7. The OpenMP API defines the worksharing constructs:
8. Sections Construct
9. Single Constructs
10. Workshare constructs

**Que 3. Implement a program for Vector – Vector Addition**

**\*Parallel-**

* **Code:**

#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 B[1000];

for(int i=0;i<N;i++)B[i] = N - i;

int C[1000] = {0};

double itime, ftime, exec\_time;

itime = omp\_get\_wtime();

#pragma omp parallel for reduction(+ : C)

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

{

C[i] = A[i] + B[i];

printf("Thread: %d Index: %d\n", omp\_get\_thread\_num(),i);

}

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

{

printf("%d ", C[i]);

}

ftime = omp\_get\_wtime();

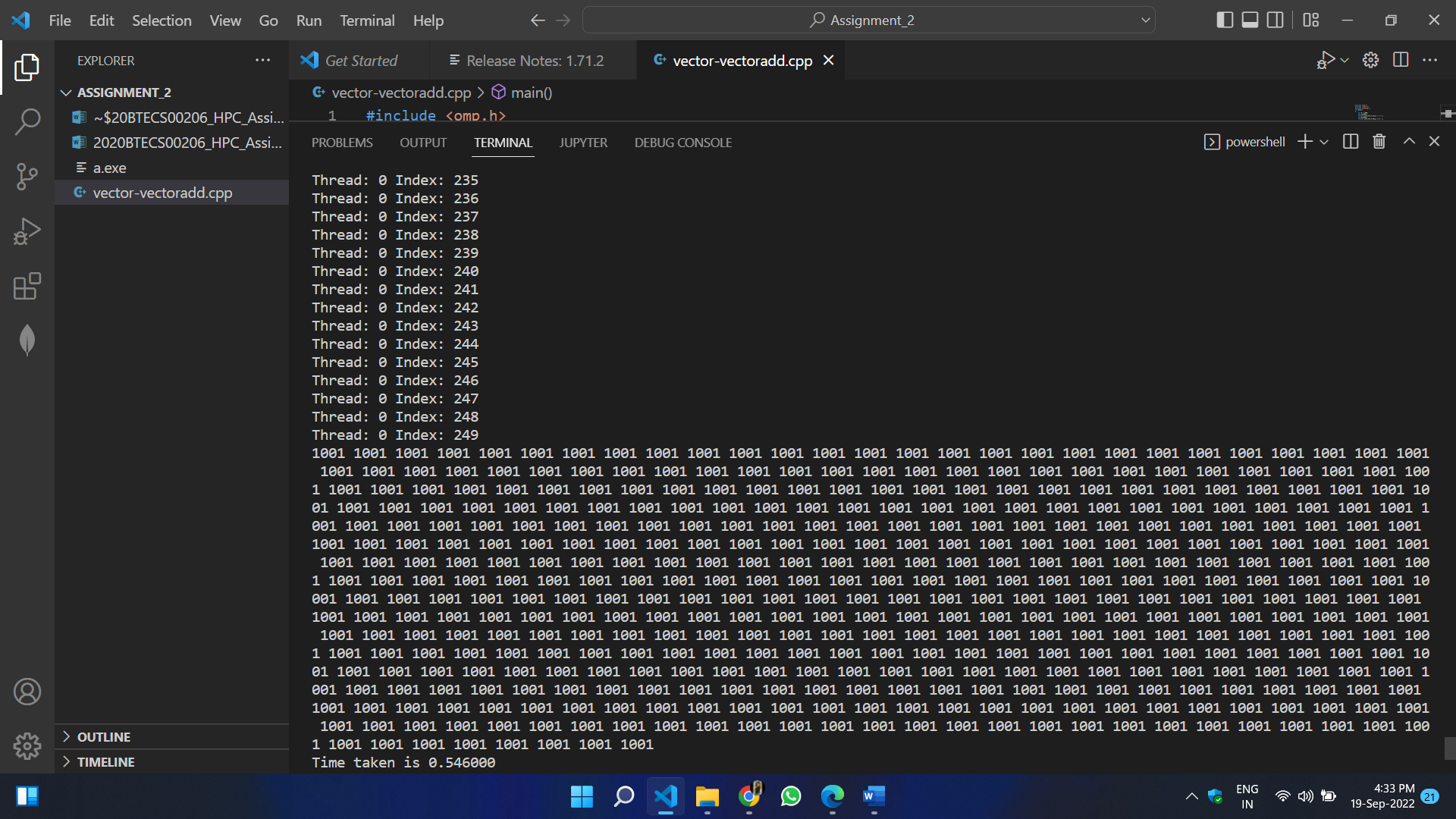
exec\_time = ftime - itime;

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

printf("\n");

}

* **Output:**



**\*Sequential-**

* **Code:**

#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 B[1000];

for(int i=0;i<N;i++)B[i] = N - i;

int C[1000] = {0};

double itime, ftime, exec\_time;

itime = omp\_get\_wtime();

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

{

C[i] = A[i] + B[i];

printf("Thread: %d Index: %d\n", omp\_get\_thread\_num(),i);

}

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

{

printf("%d ", C[i]);

}

ftime = omp\_get\_wtime();

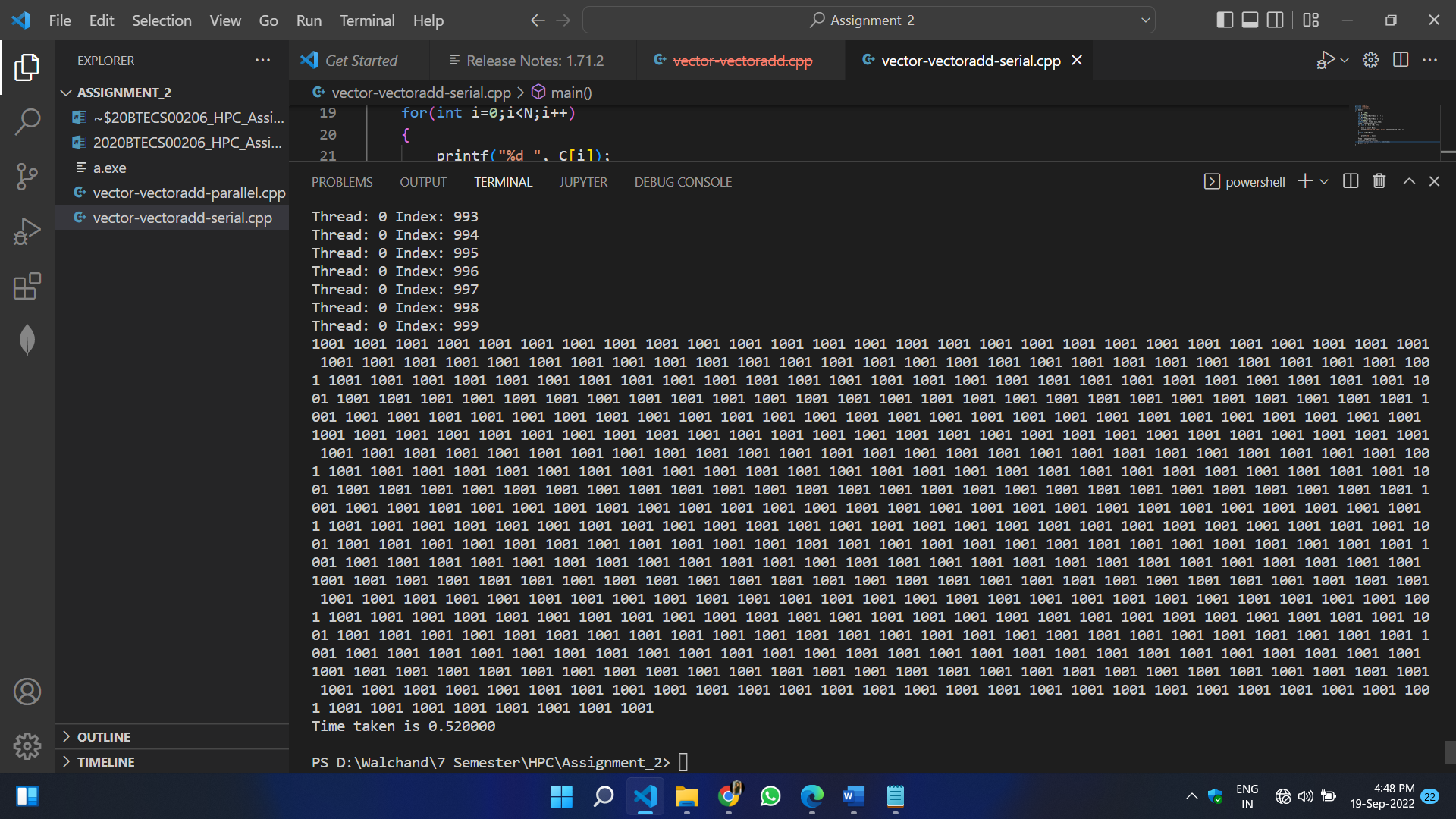
exec\_time = ftime - itime;

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

printf("\n");

}

* **Output:**



**Que 3. Implement a program for Vector – Scalar Multiplication**

**\*Parallel-**

* **Code:**

#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();

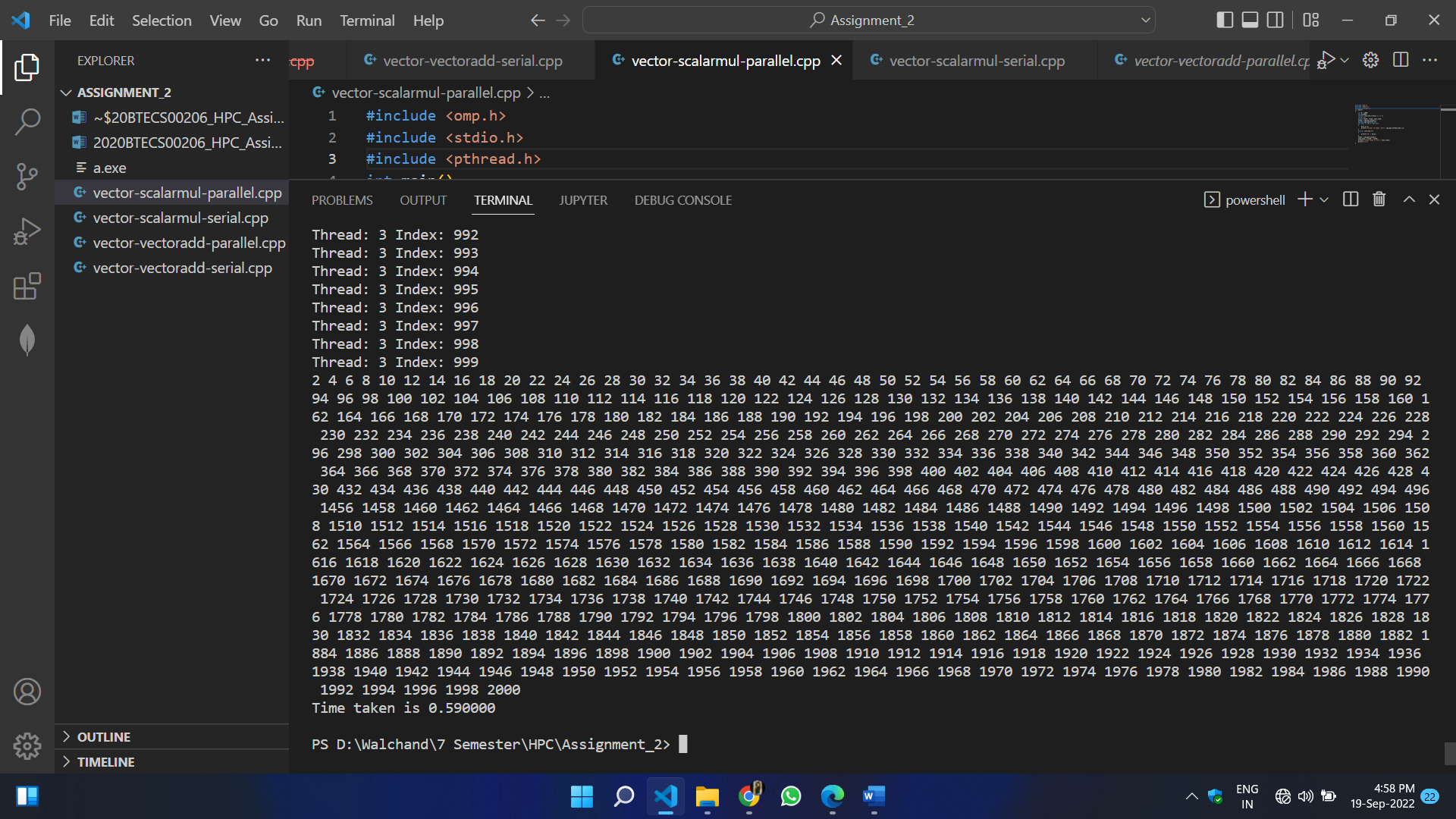
exec\_time = ftime - itime;

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

printf("\n");

}

* **Output:**



**\*Sequential-**

* **Code:**

#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();

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");

}

* **Output:**

