**Class:** Third Year B. Tech (Computer Science and Engineering)

**Year:** 2025-26 **Semester:** Odd

**Course:** Cutting Edge Technologies Lab

**Course Code:** 7CS352

**Practical No. 2**

**Exam Seat No:** 23610067

**Title of practical: Parallel region creation, thread identification using OpenMP.**

Implement following Programs using OpenMP with C:

1. Vector Vector Addition

Analyse the performance of your programs for different number of threads and Data size. Understand different ways in which you can set the number of threads for execution of parallel section, how to access thread ID and total number of threads in a code.

**Problem Statement 1:**

**Screenshots:**

**#include <iostream>**

**#include <omp.h>**

**#include <vector>**

**using namespace std;**

**int main(){**

**while(1){**

**cout<<"Enter number of threads: ";**

**int n;**

**cin >> n;**

**int vector\_size;**

**cout << "Enter size of vector: ";**

**cin >> vector\_size;**

**vector<double> vectorA(vector\_size);**

**vector<double> vectorB(vector\_size);**

**vector<double> vectorC(vector\_size);**

**for (int i = 0; i< vector\_size ; i++){**

**vectorA[i] = i\*3.0;**

**vectorB[i] = i\*2.0;**

**}**

**omp\_set\_num\_threads(n);**

**// or**

**// Environment variable: export OMP\_NUM\_THREADS=4**

**// Command line: OMP\_NUM\_THREADS=4 ./a.out**

**// Inside code: omp\_set\_num\_threads(n);**

**double start = omp\_get\_wtime();**

**#pragma omp parallel**

**{**

**#pragma omp single**

**cout << "Total number of threads: " << omp\_get\_num\_threads() << endl;**

**#pragma omp for**

**for (int i = 0; i < vector\_size ; i++){**

**vectorC[i] = vectorA[i] + vectorB[i];**

**// #pragma omp critical**

**// {**

**// cout << i << "th index Calculated by: " << omp\_get\_thread\_num() << endl;**

**// }**

**}**

**}**

**double end = omp\_get\_wtime();**

**cout << "Total time: " << end - start << endl;**

**cout << "Do you want to enter more values? (0/1) " ;**

**int flag;**

**cin >> flag;**

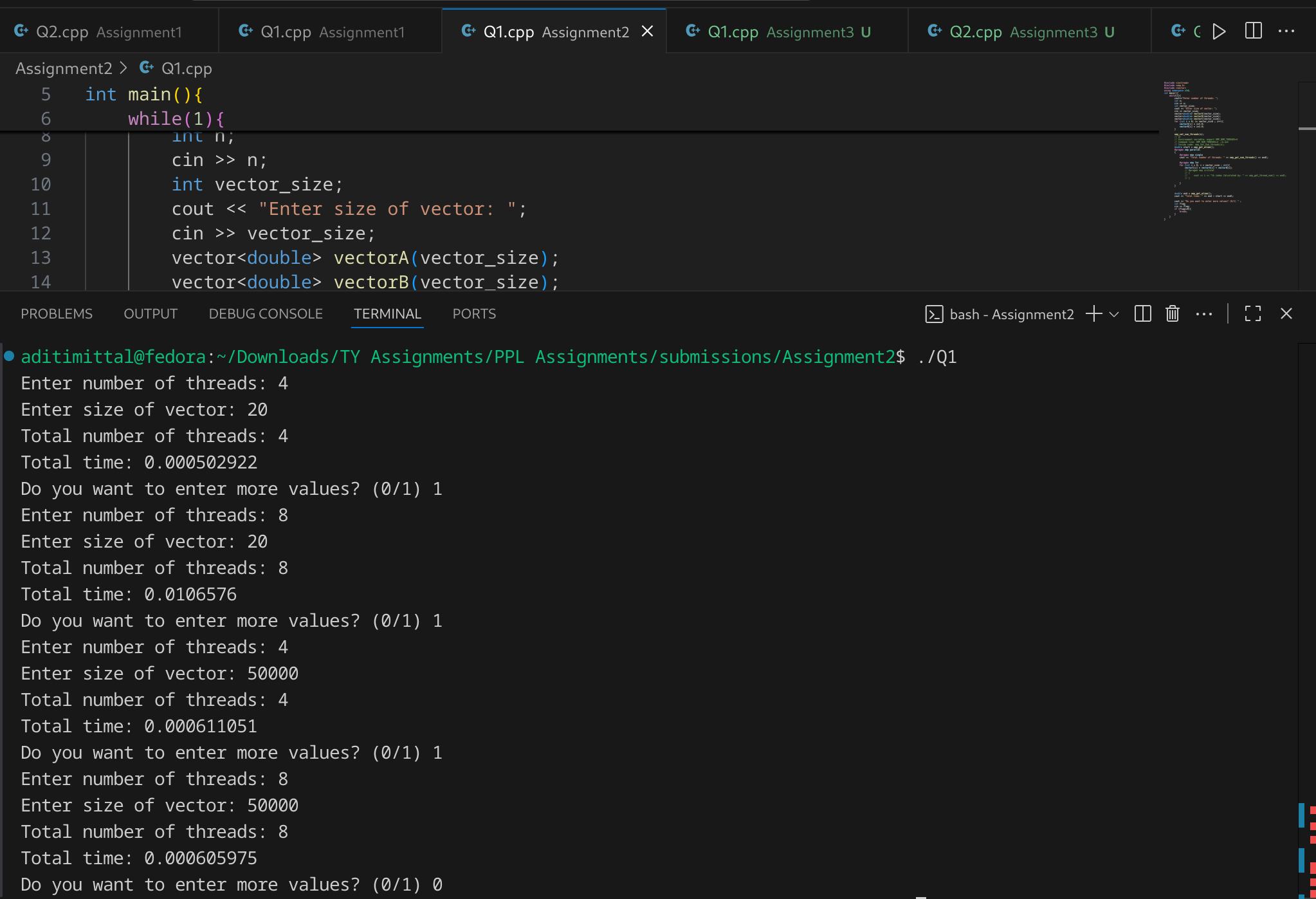
**if (flag==0){**

**break;**

**}**

**}**

**}**

****

**Information:**

**OpenMP Directives Used:**

1. #pragma omp parallel → Creates a parallel region where multiple threads execute simultaneously.
2. omp\_get\_num\_threads() → Retrieves the total number of threads participating.
3. omp\_get\_thread\_num() → Gives the **Thread ID** (0 to N-1).
4. #pragma omp for → Distributes loop iterations across available threads.
5. omp\_set\_num\_threads(n) → Manually sets the number of threads.  
   * Can also be set via **environment variable** (export OMP\_NUM\_THREADS=4) or **command line** (OMP\_NUM\_THREADS=4 ./a.out).

**Thread Identification:**

* Each thread is given a unique **thread ID**.

**Performance Measurement:**

* Execution time is measured using omp\_get\_wtime().
* Time varies with **vector size** and **number of threads**.

**Analysis:**

The result of vector addition is always correct because each thread works on different indices (no race conditions).

For small vector sizes, overhead of thread creation may dominate, so **speedup is negligible**.

### Case 1: Small Vector (N = 20)

* For **4 threads** → Time = 0.000367365 s
* For **8 threads** → Time = 0.000839803 s

**Observation:**

* Execution time is very small because vector size is too small.
* With **more threads (8)**, execution time actually **increases**, since overhead of thread creation and synchronization dominates compared to the actual work.
* **Conclusion:** For small data sizes, parallelization is inefficient.

For **large vectors** (e.g., size 50,000 or 100,000), using more threads (up to the number of logical cores) allows better parallel utilization, resulting in **slightly reduced execution time**.

**Github Link:**

[**https://github.com/aditimittal38/Parallel-Programming-Lab.git**](https://github.com/aditimittal38/Parallel-Programming-Lab.git)