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

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

**Course:** Cutting Edge Technologies Lab

**Course code:** 7CS352

**Practical No. 3**

**Exam Seat No:** 23610067

**Title of practical:**

Parallelizing loops with #pragma omp parallel for, and applying scheduling clauses

1. Print hello world using, #pragma omp parallel and #pragma omp parallel for
2. Addition of Scalar value with a vector using static scheduling
3. Addition of Scalar value with a vector using Dynamic scheduling scheduling

**Problem Statement 1:**

Print hello world using, #pragma omp parallel and #pragma omp parallel for

**Screenshots:**

**// Print hello world using, #pragma omp parallel and #pragma omp parallel for**

**#include <iostream>**

**#include <omp.h>**

**#include <iomanip>**

**using namespace std;**

**int main(){**

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

**for(int i=0; i<10; i++){**

**cout << "Hello from sequential" <<endl;**

**}**

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

**cout << " -------------------------------------------------------" << endl;**

**omp\_set\_num\_threads(4);**

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

**#pragma omp parallel**

**{**

**#pragma omp single**

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

**cout << "Hello world from thread " << omp\_get\_thread\_num() <<endl;**

**}**

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

**cout << " -------------------------------------------------------" << endl;**

**// #pragma omp parallel**

**// {**

**// #pragma omp single**

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

**// for(int i=0; i<10; i++){**

**// cout << "Hello world from thread " << omp\_get\_thread\_num() <<endl;**

**// }**

**// }**

**// cout << " -------------------------------------------------------" << endl;**

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

**#pragma omp parallel for**

**for(int i=0; i<10; i++){**

**cout << "Hello " << i << "world from thread " << omp\_get\_thread\_num() <<endl;**

**}**

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

**double seq\_time = end\_seq - start\_seq;**

**double parallel\_time = end\_parallel - start\_parallel;**

**double parallel\_for\_time = end\_parallel\_for - start\_parallel\_for;**

**cout << "------------------------------------------------------" << endl;**

**cout << fixed << setprecision(9);**

**cout << "Sequential time: "<< seq\_time<<endl;**

**cout << "Parallel time: "<<parallel\_time<<endl;**

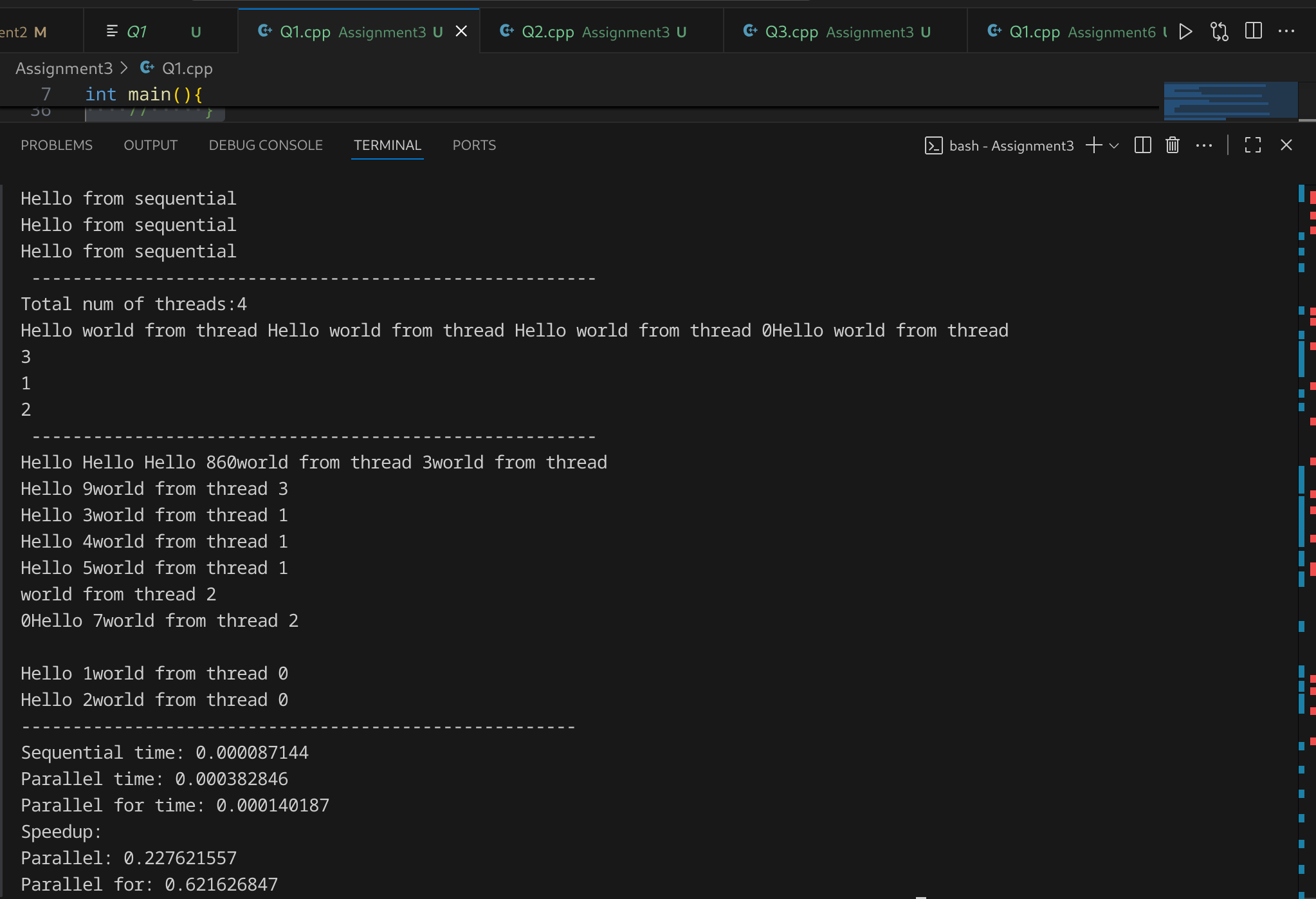
**cout << "Parallel for time: "<< parallel\_for\_time <<endl;**

**cout << "Speedup: "<<endl;**

**cout << "Parallel: " << seq\_time/parallel\_time<<endl;**

**cout << "Parallel for: " << seq\_time/parallel\_for\_time<<endl;**

**}**

****

**Information and analysis: Time analysis, Speedup analysis for all programs**

### Analysis

1. **Speedup < 1** (i.e., slower than sequential):  
   * This happens because the task (printing Hello World) is **very small** and **I/O-bound**.
   * Creating and managing multiple threads adds **overhead** that is larger than the actual work done.
2. **Why parallel for is better than parallel:**
   * In #pragma omp parallel, every thread just prints once → more overhead than useful work.
   * In #pragma omp parallel for, the loop is divided among threads, giving **slightly better workload distribution**.
3. **Real benefit of OpenMP:**
   * Speedup becomes noticeable only when **computationally heavy tasks (like matrix multiplication, FLOP-intensive calculations, etc.)** are parallelized.
   * For lightweight tasks like printing, sequential is always faster.

**Problem Statement 2:**

Addition of Scalar value with a vector using static scheduling

**Screenshots:**

**// Addition of Scalar value with a vector using static scheduling**

**#include <iostream>**

**#include <omp.h>**

**#include <vector>**

**#include <iomanip>**

**using namespace std;**

**int main(){**

**int vector\_size;**

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

**cin >> vector\_size;**

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

**int scalarVal;**

**cout << "Enter Scalar Value: " ;**

**cin >> scalarVal;**

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

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

**}**

**double start\_seq, start\_static, start\_dynamic, end\_seq, end\_static, end\_dynamic;**

**omp\_set\_num\_threads(4);**

**cout << " ----------------------------Sequential------------------------------------------- "<<endl;**

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

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

**vectorA[i] += scalarVal;**

**}**

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

**cout << " ----------------------------Static------------------------------------------- "<<endl;**

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

**#pragma omp parallel for schedule(static)**

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

**vectorA[i] += scalarVal;**

**// #pragma omp critical**

**// {**

**// cout << "Index " << i << " updated to " << vectorA[i] << " by thread " << omp\_get\_thread\_num() << endl;**

**// }**

**}**

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

**cout << " -----------------------------Dynamic------------------------------------------ "<<endl;**

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

**#pragma omp parallel for schedule(dynamic)**

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

**vectorA[i] += scalarVal;**

**// #pragma omp critical**

**// {**

**// cout << "Index " << i << " updated to " << vectorA[i] << " by thread " << omp\_get\_thread\_num() << endl;**

**// }**

**}**

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

**cout << fixed << setprecision(9);**

**double seq\_time = end\_seq - start\_seq;**

**double static\_time = end\_static - start\_static;**

**double dynamic\_time = end\_dynamic - start\_dynamic;**

**cout << "\n--------------------------- Performance Summary -----------------" << endl;**

**cout << "Sequential Time: " << seq\_time << " seconds" << endl;**

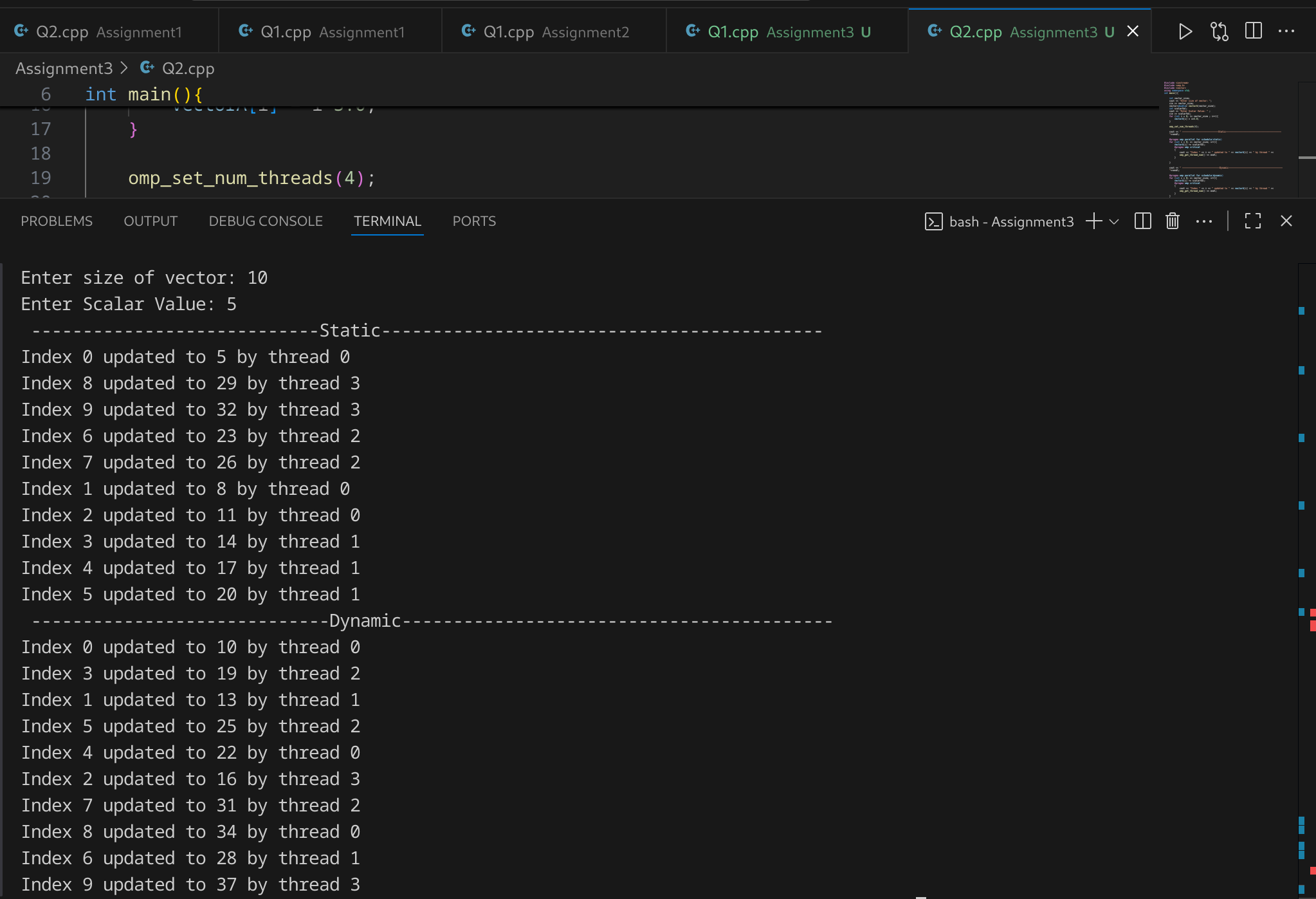
**cout << "Static Time: " << static\_time << " seconds" << endl;**

**cout << "Dynamic Time: " << dynamic\_time << " seconds" << endl;**

**cout << "\nSpeedup (Static vs Seq): " << seq\_time / static\_time << endl;**

**cout << "Speedup (Dynamic vs Seq): " << seq\_time / dynamic\_time << endl;**

**}**

****

**Problem Statement 3:**

Addition of Scalar value with a vector using dynamic scheduling

**Screenshots:**

**// Addition of Scalar value with a vector using static scheduling**

**#include <iostream>**

**#include <omp.h>**

**#include <vector>**

**using namespace std;**

**int main(){**

**int vector\_size;**

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

**cin >> vector\_size;**

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

**int scalarVal;**

**cout << "Enter Scalar Value: " ;**

**cin >> scalarVal;**

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

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

**}**

**omp\_set\_num\_threads(4);**

**cout << " -----------------------------Dynamic------------------------------------------ "<<endl;**

**#pragma omp parallel for schedule(dynamic)**

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

**vectorA[i] += scalarVal;**

**#pragma omp critical**

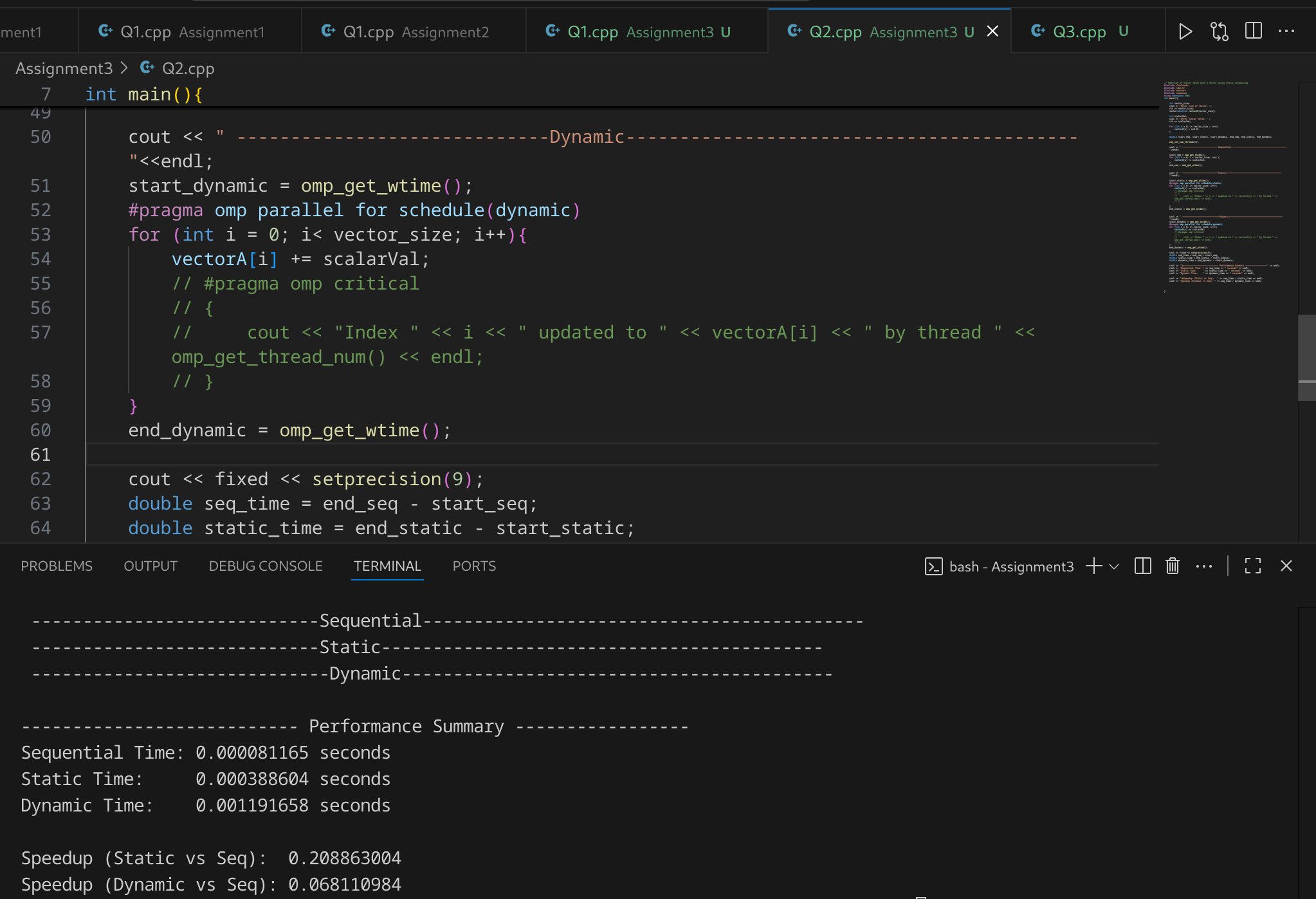
**{**

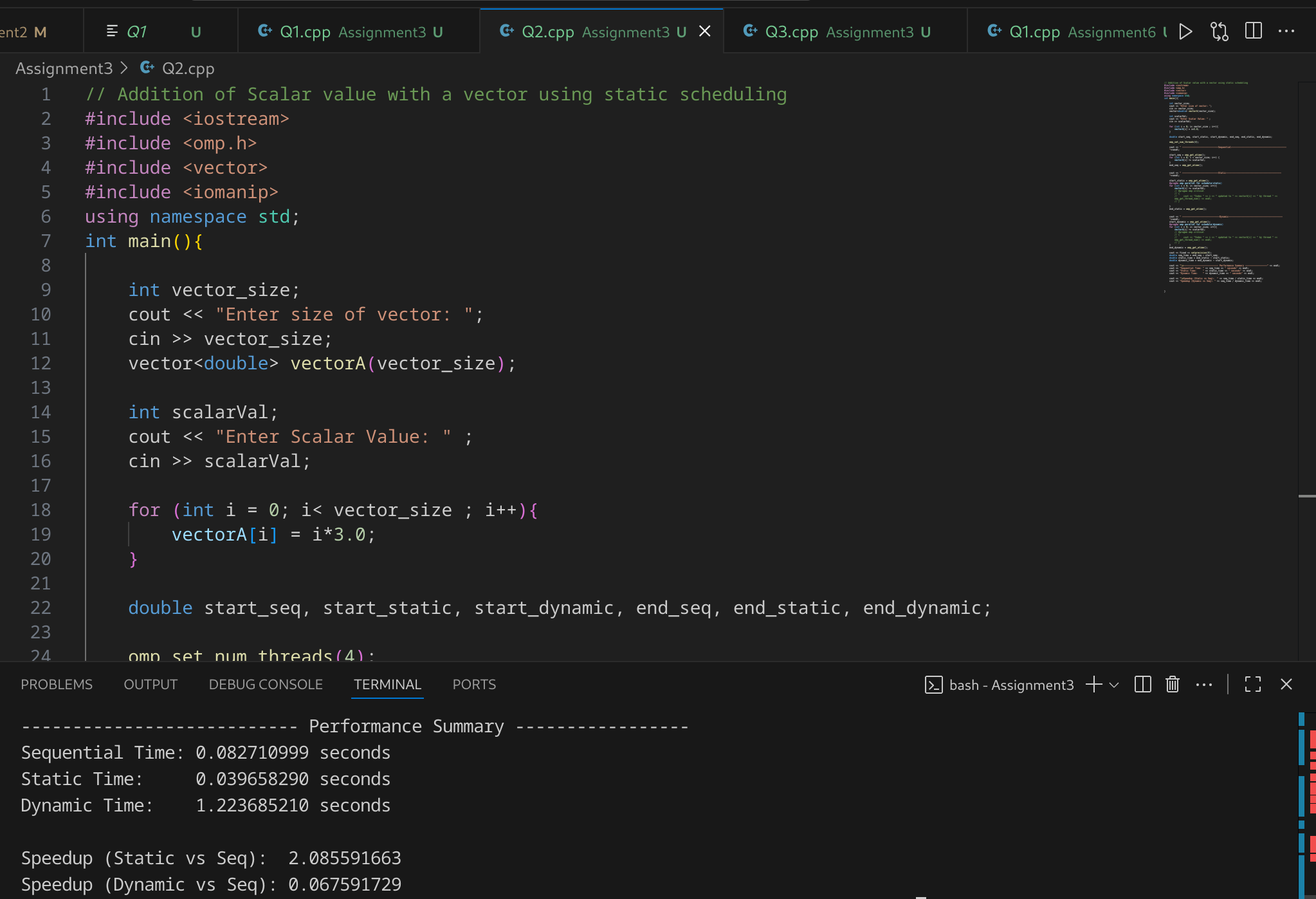
**cout << "Index " << i << " updated to " << vectorA[i] << " by thread " << omp\_get\_thread\_num() << endl;**

**}**

**}**

**}**

**For greater value of n: n= 1000000**

****

**Information and analysis:**

1. **Sequential is the fastest here (for lower values of n)**
   * Sequential loop: **8.1e-5 sec** (very small).
   * Static parallel: **3.9e-4 sec** (~5× slower).
   * Dynamic parallel: **1.19e-3 sec** (~15× slower).
2. Overheads (thread creation, scheduling, synchronization) dominate because the per-iteration work is *too trivial* (+scalarVal).
3. **Static vs Dynamic**
   * **Static scheduling is faster than dynamic** here.
   * Reason:  
     + Static assigns iterations *evenly* to threads once, so minimal scheduling overhead.
     + Dynamic keeps reassigning chunks → more overhead.
   * Since the loop iterations are uniform and cheap, static is the better fit.
4. Inference: Use **static scheduling** for balanced and lightweight workloads. Use **dynamic scheduling** only if iterations are uneven or unpredictable.
5. **Speedup values < 1**
   * Both speedups are much less than 1 (slowdown, not speedup).
   * This confirms that **parallelization is not worth it for small/light problems**.
   * For larger vectors (say 1e6 or more), the cost of actual computation grows and may start overcoming the OpenMP overhead, giving **speedup > 1**.

### Conclusion

* For **small workloads**, sequential execution outperforms parallel due to thread management and scheduling overhead.
* **Static scheduling** is more efficient than dynamic in this case, since the workload is uniform and predictable.
* **Dynamic scheduling** incurs higher scheduling overhead, making it slower.
* **Speedup < 1** indicates that parallelization is not beneficial here.
* Parallelization is effective only when problem size is large enough or computations per iteration are heavy.

**Github Link:**

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