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

**Year:** 2025-26 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 2**

**PRN: 22510047**

**Name:Aadarsh Venkatesh Nandedkar**

**Batch: B8**

**Title of practical: Study and implementation of basic OpenMP clauses**

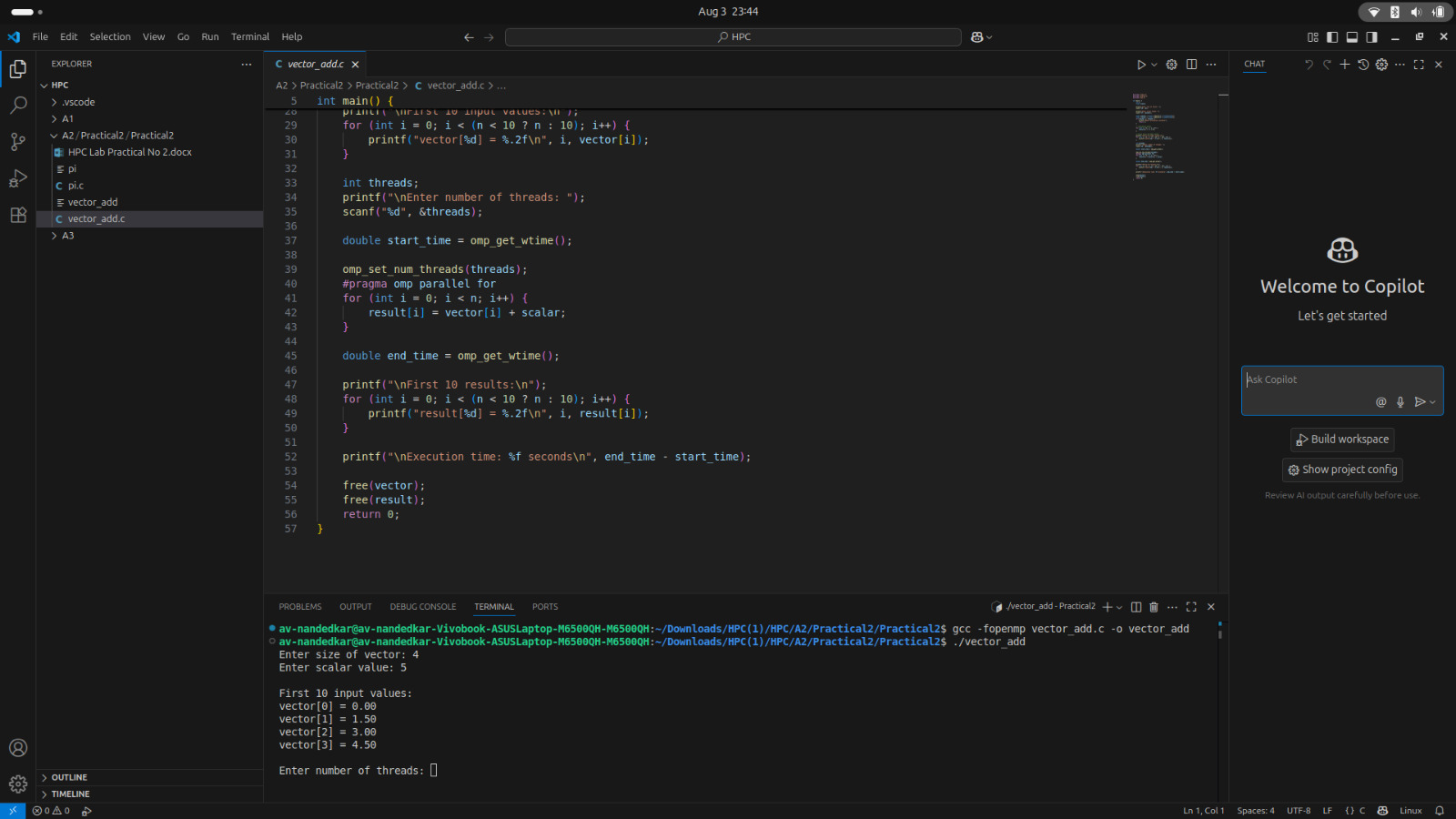
Implement following Programs using OpenMP with C:

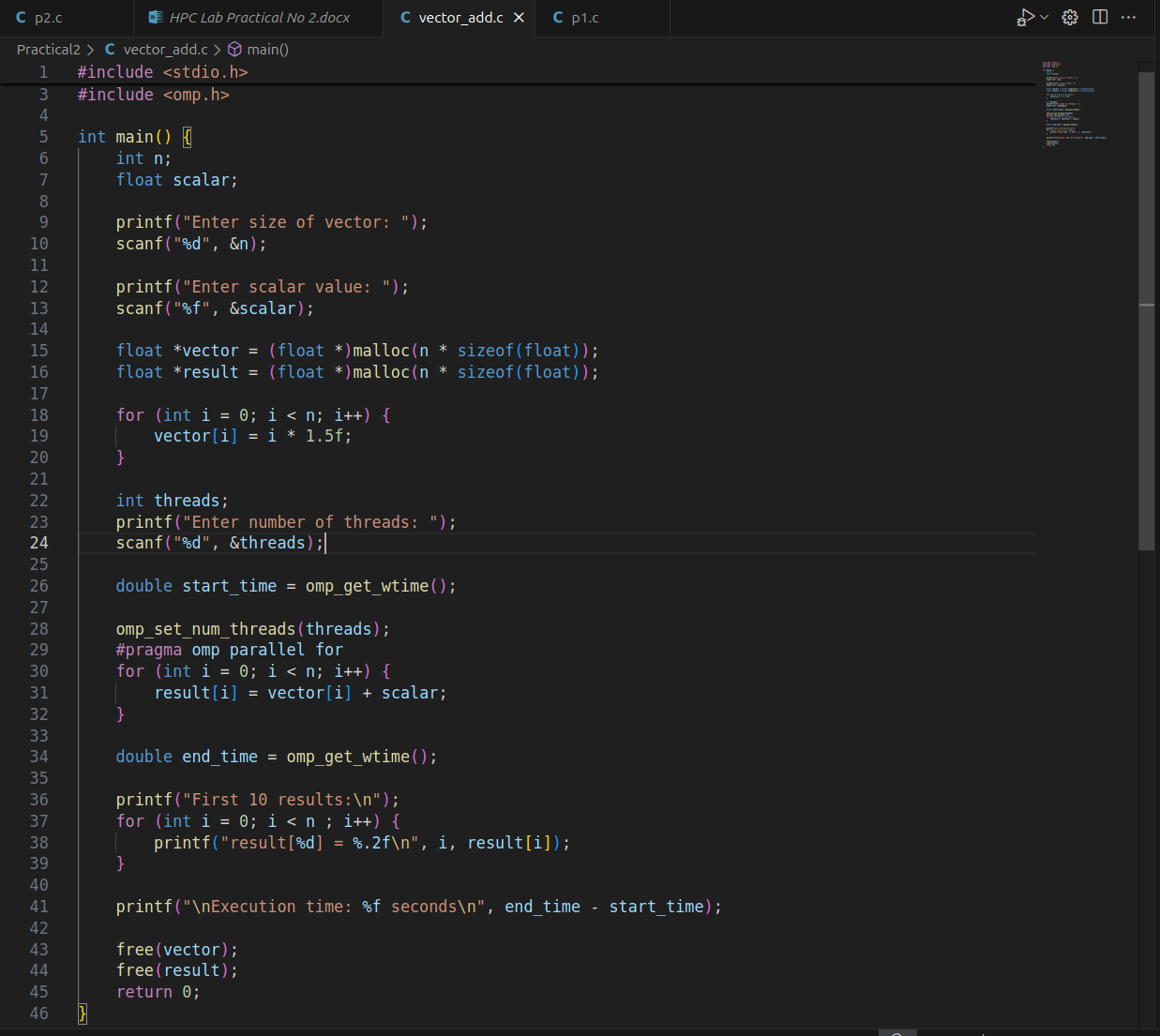
1. Vector Scalar Addition
2. Calculation of value of Pi

Analyse the performance of your programs for different number of threads and Data size.

**Problem Statement 1:** Vector Scalar Addition

**Screenshots:**



****

**Information & Analysis:**

## OpenMP Basics Used in the Program

### a) #pragma omp parallel for

* This directive parallelizes the for loop, distributing iterations across multiple threads.
* Each thread computes a portion of the loop iterations, improving performance for large vectors.

### b) omp\_set\_num\_threads(threads)

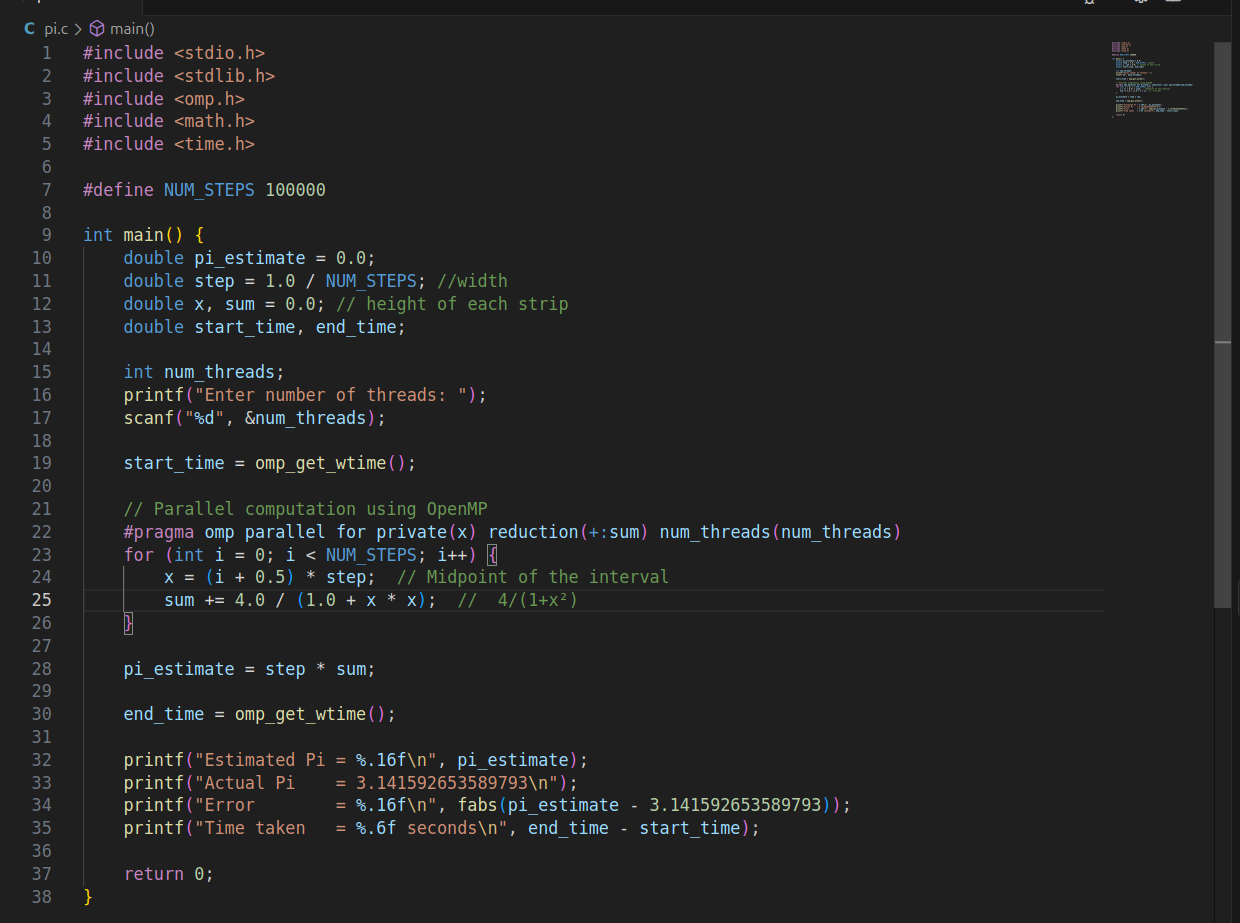
* Explicitly sets the number of threads to use.
* If more threads than available CPU cores are requested, OpenMP may limit actual thread usage.

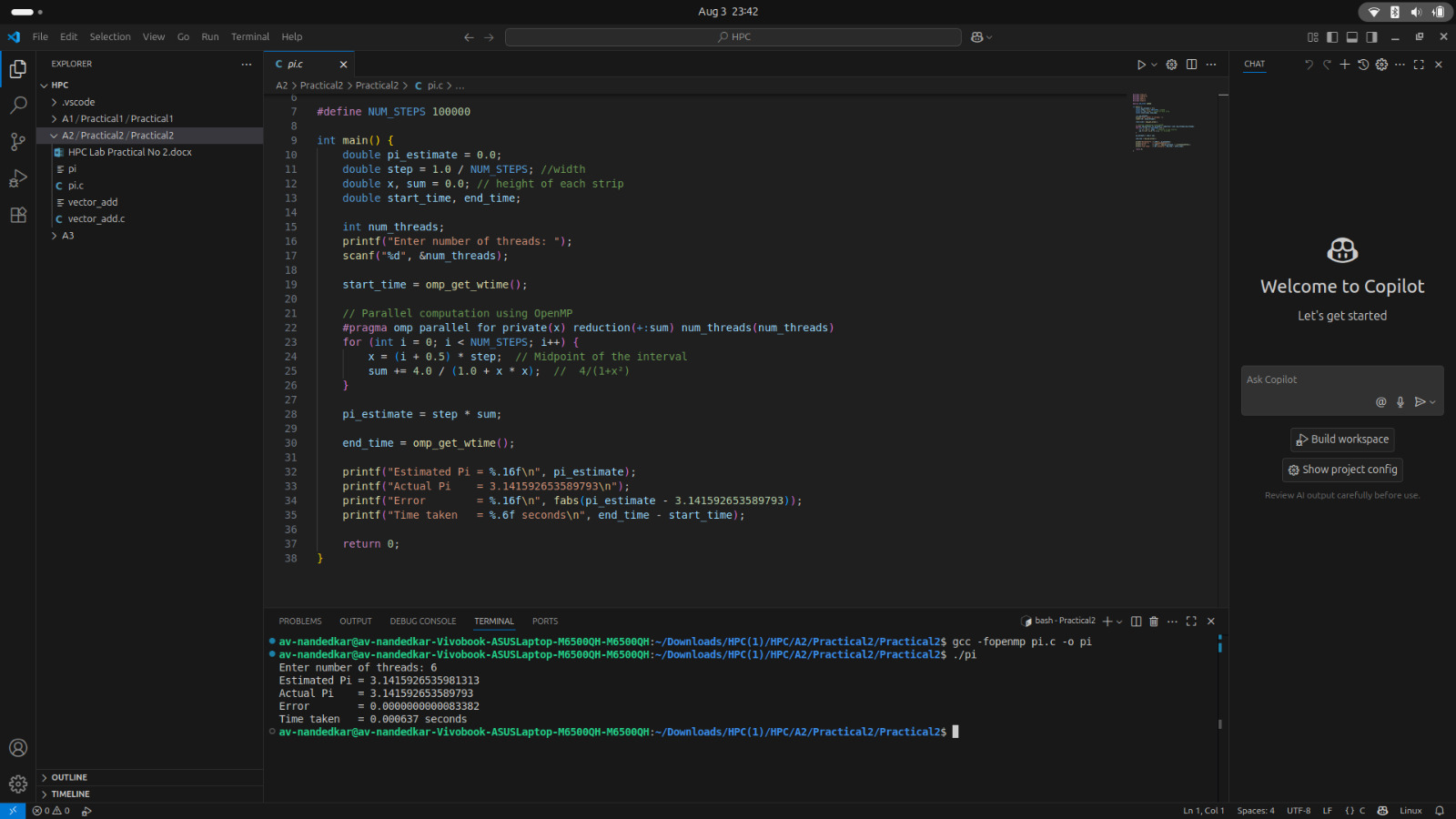
### c) omp\_get\_wtime()

* Used to measure clock time (real time taken for execution).
* Helps in comparing parallel vs. serial performance.

**Problem Statement 2:** Calculation of value of Pi

**Screenshots:**

****



**Information:**

### OpenMP Parallelization

#pragma omp parallel for: Splits the loop across multiple CPU threads for faster computation.

private(x): Each thread gets its own copy of x (no conflicts between threads).

reduction(+:sum): Combines partial sums from all threads safely (no race condition).

num\_threads(num\_threads): Sets the number of threads manually (user input).

**Analysis:**

How OpenMP Helps:

Parallelizes the loop (#pragma omp parallel for), splitting work across threads.

Uses reduction(+:sum) to safely combine results.

Thread Count Impact

Too few threads (1-2): Near-serial speed (minimal speedup).

Optimal threads (=CPU cores): Best performance.

Too many threads (>cores): Slower due to overhead (OS context switching).

Accuracy Depends on NUM\_STEPS (more steps = better precision), not threads.

**Github Link:**

<https://github.com/av-nandedkar/HPC>