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

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

**Course:** High Performance Computing Lab

**Practical No. 2**

**Exam Seat No: 22510092**

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

Implement following Programs using OpenMP with C:

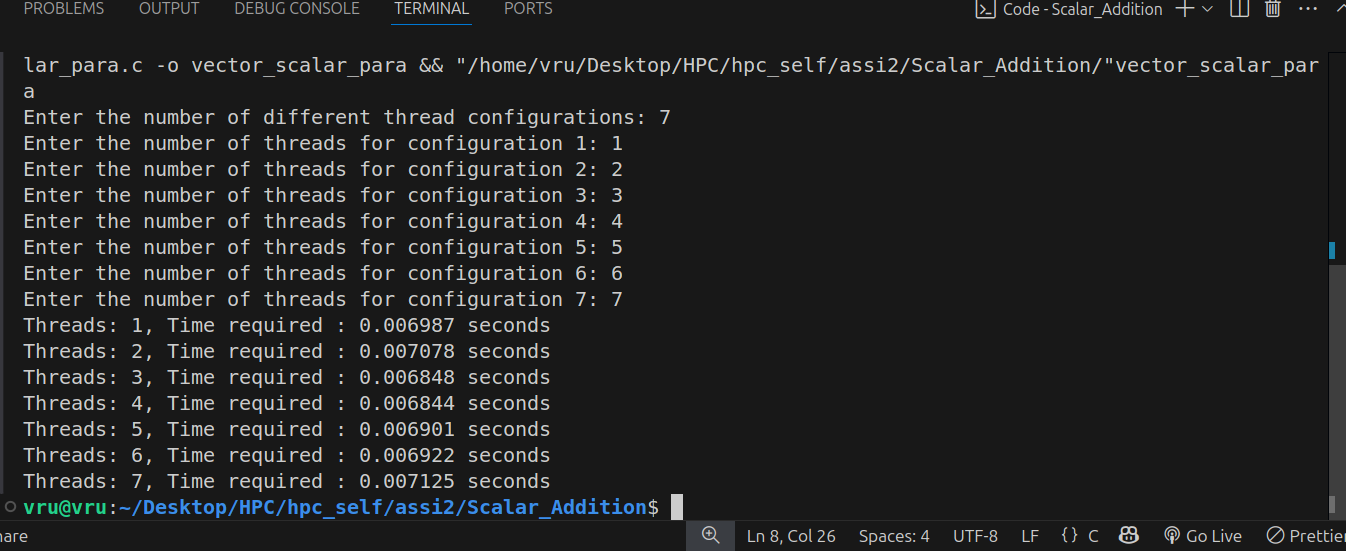
1. Vector Scalar Addition
2. Calculation of value of Pi

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

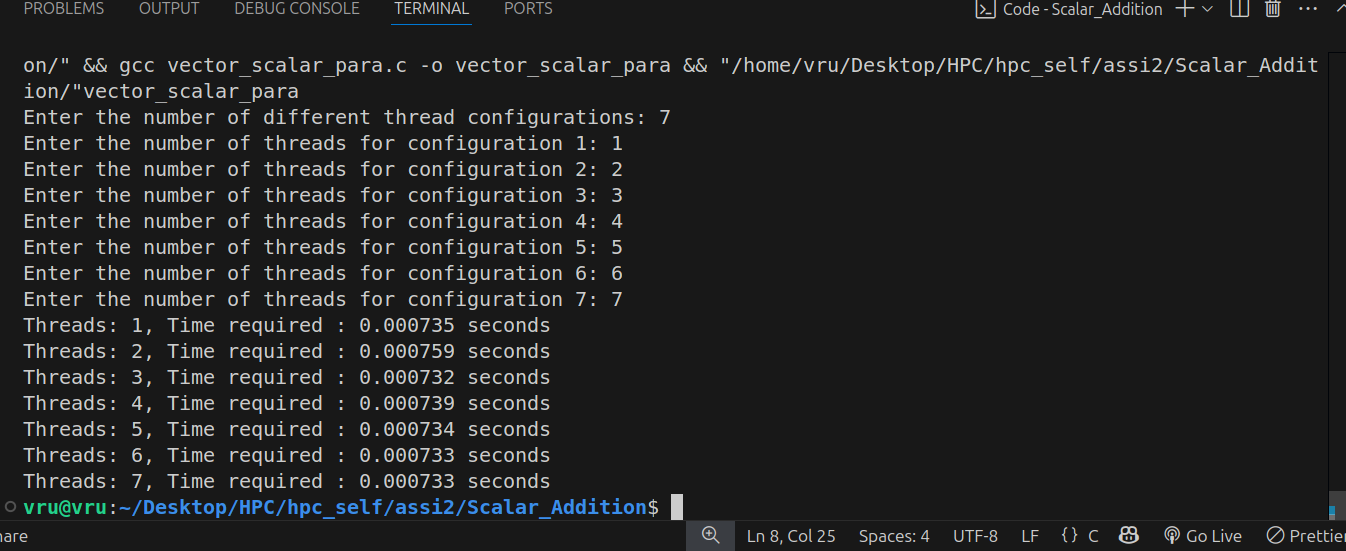
**Problem Statement 1:**

**Screenshots:**

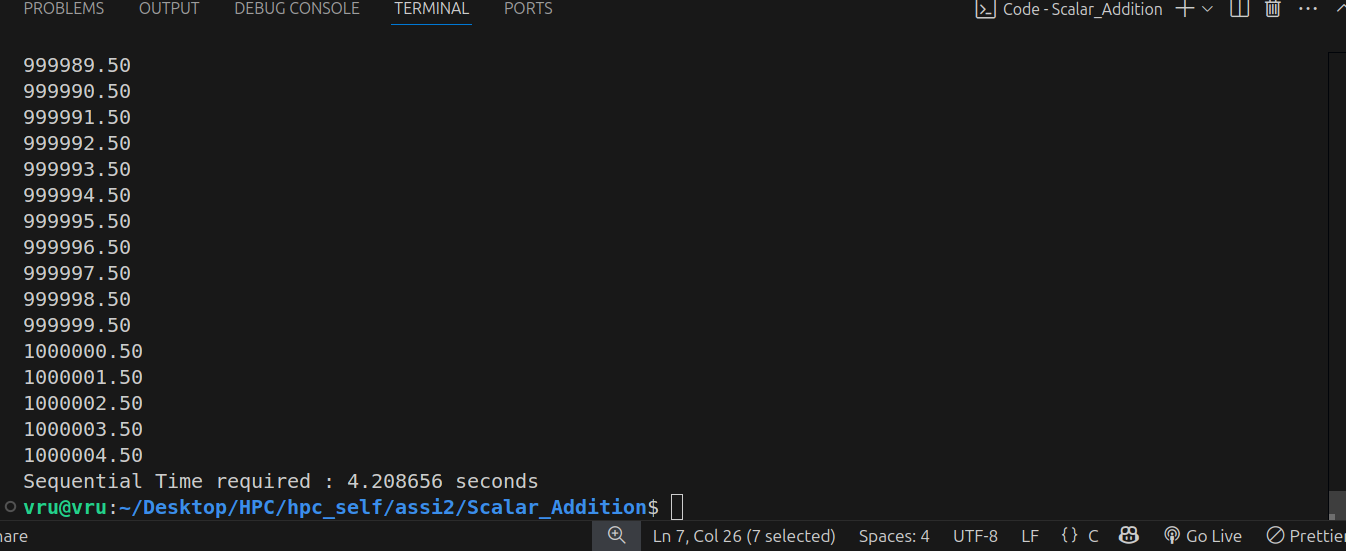
**1 – Parallel Addition For 1000000**



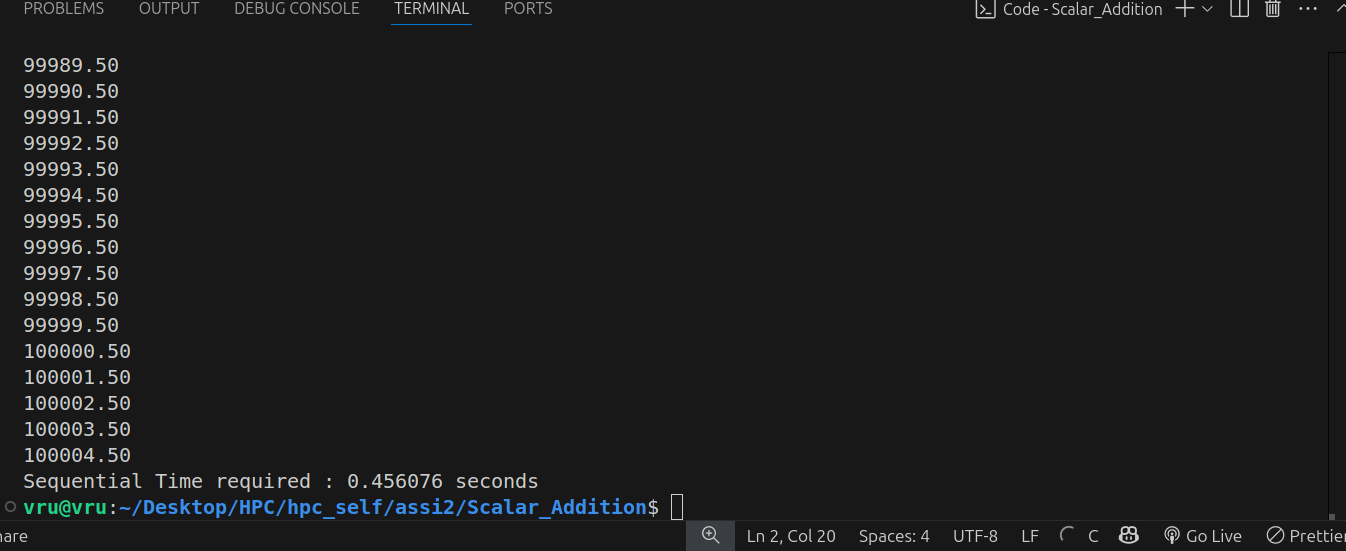
**For 10000 :**



**2 – Sequential Addition For 1000000**



**For 100000**



**Information:** I tried on different values such as 1000000, 100000 etc, for both parallel and sequential execution.

**Analysis:**

1. Sequential Execution:

- Time increases with vector size.

- Sequential execution is straightforward but becomes impractical for large datasets due to longer processing times.

2. Parallel Execution:

- Parallel execution is faster than sequential for larger vector sizes, demonstrating the benefits of utilizing multiple threads.

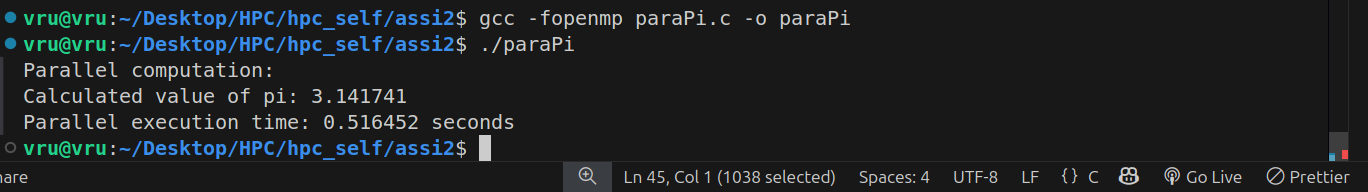
- The performance improvement is more significant with larger datasets and more threads.

**- The optimal number of threads depends on the hardware (number of CPU cores) and the size of the data.**

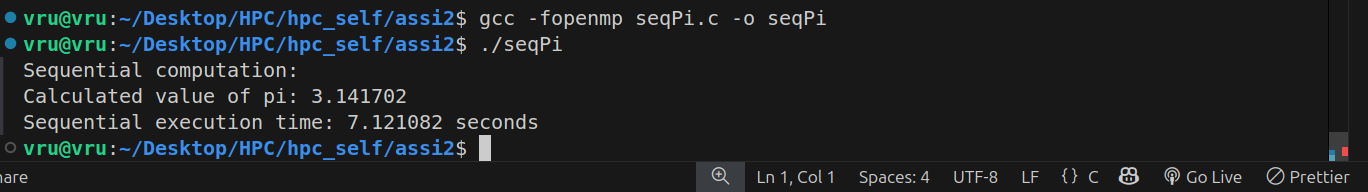
**Problem Statement 2:**

**Screenshots:**

**Parallel --**



**Sequential --**



**Information:**

**Monte Carlo Pi estimation** uses random point generation to approximate the value of π. Parallelizing it with OpenMP significantly reduces execution time by distributing computations across multiple threads.

**Analysis:**

****A**ccuracy:**

* Both versions provide an approximate value of π (close to 3.14159).
* **Accuracy depends on the number of points; more points → higher accuracy.**

**Performance Comparison:**

* The sequential version uses a single core and takes significantly more time.
* The parallel version distributes the computation across multiple threads using OpenMP and significantly reduces the execution time.

**Github Link:** [**https://github.com/Vru01/HPC\_22510092**](https://github.com/Vru01/HPC_22510092)