**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: 22510046**

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

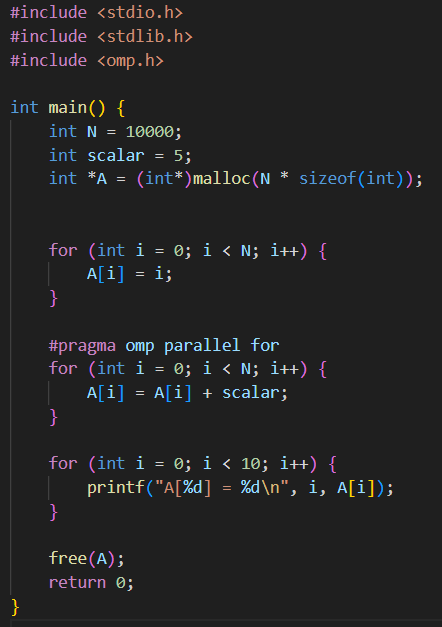
Implement following Programs using OpenMP with C:

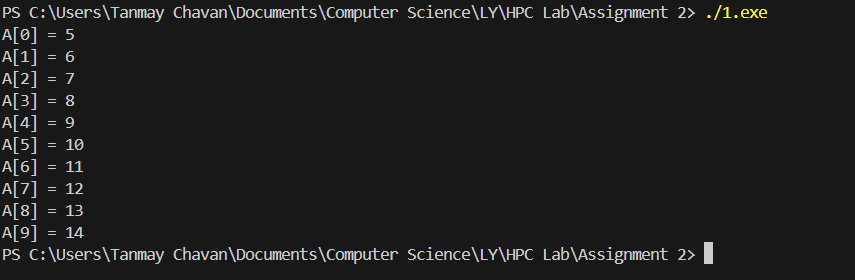
1. Vector Scalar Addition
2. Calculation of value of Pi

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

**Problem Statement 1:**

**Screenshots:**

****

****

**Information:**

The for Clause: #pragma omp parallel for: This is a very common and powerful pragma. It combines two things:

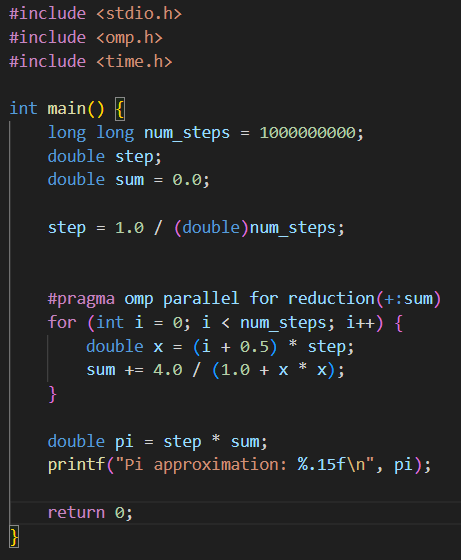
1. #pragma omp parallel: Creates a team of threads.
2. #pragma omp for: Divides the iterations of the following for loop among the threads. Each thread gets a chunk of the loop to execute.

Analysis:

We will need to measure the execution time for different numbers of threads (e.g., 1, 2, 4, 8) and different data sizes (N). We'll likely see a speedup as you increase the number of threads, but the speedup will eventually plateau due to overheads and Amdahl's Law. We should also note that a larger data size will benefit more from parallelization.

**Problem Statement 2:**

**Screenshots:**

****

****

Information:

The reduction clause is designed to solve this problem. It tells OpenMP to create a private copy of a variable for each thread. Each thread performs its own sum on its private copy. After the loop is finished, OpenMP combines all the private sums into the final shared variable using the specified operation

Analysis:

Similar to the previous problem, we analyze the performance with different numbers of threads. we should observe a clear speedup because the reduction clause correctly handles the race condition. Without reduction, the parallel version would likely be slower and produce an incorrect result due to the overhead of protecting the shared sum variable.

Github Link: