**Class:** Final Year (Computer Science and Engineering)

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 2**

**Exam Seat No:**

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

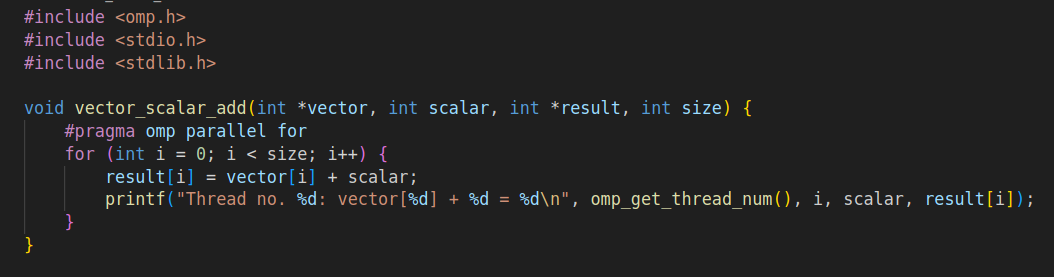
Implement following Programs using OpenMP with C:

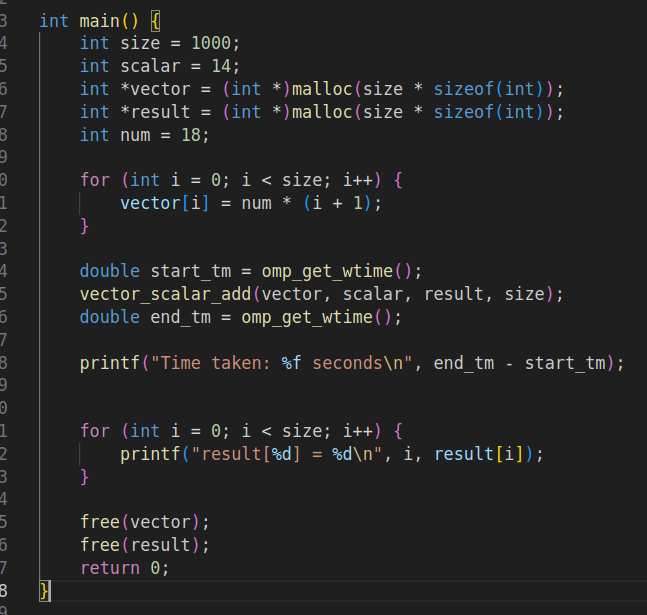
1. Vector Scalar Addition
2. Calculation of value of Pi

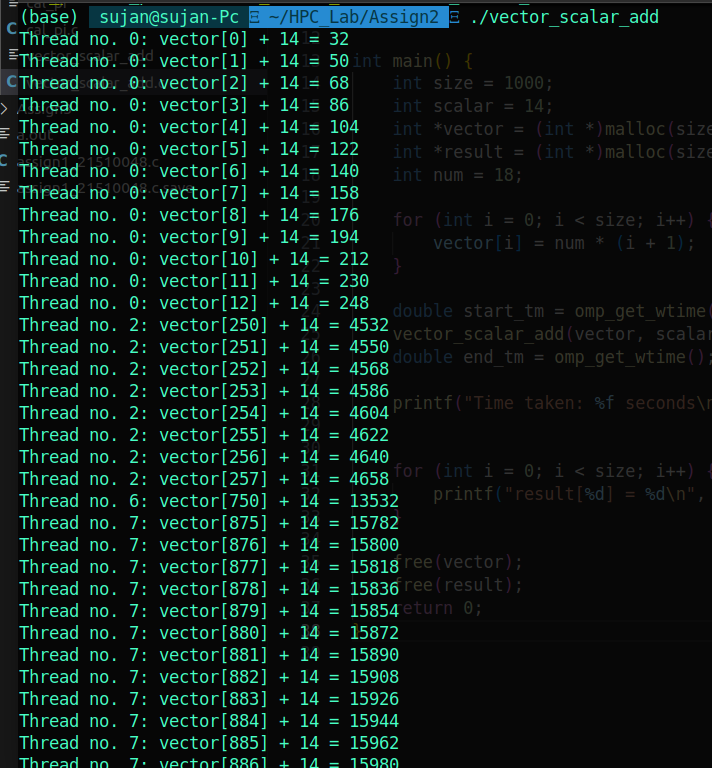
Analyse the performance of your programs for different number of threads and Data size.

**Problem Statement 1:**

**Screenshots:**

****

****



**Analysis**

### **Variable Sharing Details**

1. vector:
   * **Shared**: The pointer vector is shared among all threads. The array it points to is accessible to all threads. The threads access the elements of vector in the for loop.
2. scalar:
   * **Shared**: The variable scalar is shared among all threads. Since scalar is not modified by the threads, it is safe for all threads to read the same value.
3. result:
   * **Shared**: The pointer result is shared among all threads. The array it points to is accessible to all threads. Each thread writes to a different index of the result array, so there are no conflicts. However, the result array itself is shared.
4. i:
   * **Private**: The loop variable i in the for loop is private to each thread. Each thread has its own instance of i and does not interfere with other threads' i values. This is a default behavior for loop variables in OpenMP parallel loops.

**Problem Statement 2:**

**Screenshots:**

**Information:**

**Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| Threads | Points | Estimated Value of Pi | Time Taken (seconds) |
| 2 | 500 | 3.136 | 0.000184 |
| 2 | 1000 | 3.07 | 0.000024 |
| 2 | 2000 | 3.09 | 0.000043 |
| 2 | 5000 | 3.132 | 0.000143 |
| 4 | 500 | 3.184 | 0.000114 |
| 4 | 1000 | 3.180 | 0.000015 |
| 4 | 2000 | 3.114 | 0.000024 |
| 4 | 5000 | 3.1296 | 0.000055 |
| 8 | 500 | 3.128 | 0.001822 |
| 8 | 1000 | 3.168 | 0.000062 |
| 8 | 2000 | 3.170 | 0.000044 |
| 8 | 5000 | 3.124 | 0.000113 |
| 16 | 500 | 3.032 | 0.000661 |
| 16 | 1000 | 3.14 | 0.000202 |
| 16 | 2000 | 3.16 | 0.000121 |
| 16 | 5000 | 3.124 | 0.000208 |

**Analysis:**

**Accuracy: The accuracy of the Pi approximation improves with increasing the number of points but varies less with the number of threads once you reach a certain data size.**

* **Performance**: Optimal performance is achieved by balancing the number of threads with the size of the data. Too few threads may not fully utilize the available cores, while too many threads may introduce significant overhead, especially with smaller datasets.

**Github Link:**