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

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

**Course:** High Performance Computing Lab

**Practical No. 2**

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

Implement following Programs using OpenMP with C:

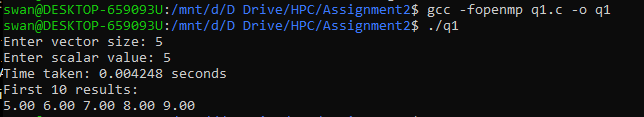
1. Vector Scalar Addition
2. Calculation of value of Pi

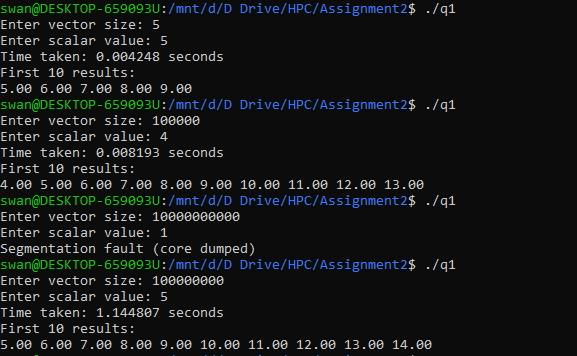
Analyse the performance of your programs for different number of threads and Data size.

**Problem Statement 1:**

**Screenshots:**

|  |
| --- |
| ***#include*** *<stdio.h>* ***#include*** *<stdlib.h>* ***#include*** *<omp.h>*  int **main**() {  int n;  float scalar;  printf("Enter vector size: ");  scanf("%d", **&**n);  printf("Enter scalar value: ");  scanf("%f", **&**scalar);   float **\***vec **=** (float**\***) malloc(n **\*** sizeof(float));  float **\***result **=** (float**\***) malloc(n **\*** sizeof(float));   for (int i **=** 0; i **<** n; i**++**) {  vec[i] **=** i **\*** 1.0f;  }   double start **=** omp\_get\_wtime();     ***#pragma omp parallel for***  for (int i **=** 0; i **<** n; i**++**) {  result[i] **=** vec[i] **+** scalar;  }   double end **=** omp\_get\_wtime();   printf("Time taken: %f seconds\n", end **-** start);   printf("First 10 results:\n");  for (int i **=** 0; i **<** 10 **&&** i **<** n; i**++**) {  printf("%.2f ", result[i]);  }  printf("\n");   free(vec);  free(result);  return 0; } |





**Information:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Index** | **Original Value** | **Scalar** | **Result** |
| 0 | 0.00 | 5 | 5.00 |
| 1 | 1.00 | 5 | 6.00 |
| 2 | 2.00 | 5 | 7.00 |
| 3 | 3.00 | 5 | 8.00 |
| 4 | 4.00 | 5 | 9.00 |

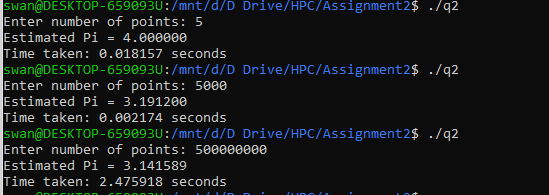
**Analysis:**

**Small vector size parallization may perform worst than sequential.**

**Problem Statement 2:**

**Screenshots:**

|  |
| --- |
| ***#include*** *<stdio.h>* ***#include*** *<stdlib.h>* ***#include*** *<omp.h>* ***#include*** *<time.h>*  int **main**() {  long long int num\_points;  printf("Enter number of points: ");  scanf("%lld", **&**num\_points);   long long int count **=** 0;   double start **=** omp\_get\_wtime();   ***#pragma omp parallel***  {  unsigned int seed **=** time(NULL) **^** omp\_get\_thread\_num();  long long int local\_count **=** 0;   ***#pragma omp for***  for (long long int i **=** 0; i **<** num\_points; i**++**) {  double x **=** (double) rand\_r(**&**seed) **/** RAND\_MAX;  double y **=** (double) rand\_r(**&**seed) **/** RAND\_MAX;  if (x**\***x **+** y**\***y **<=** 1.0) {  local\_count**++**;  }  }   ***#pragma omp atomic***  count **+=** local\_count;  }   double pi **=** (4.0 **\*** count) **/** num\_points;  double end **=** omp\_get\_wtime();   printf("Estimated Pi = %f\n", pi);  printf("Time taken: %f seconds\n", end **-** start);   return 0; } |



**Information:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Points (N)** | **Estimated Pi** | **Time Taken (seconds)** | **Observation** |
| 5 | 4.000000 | 0.018157 | Very small dataset, inaccurate estimation. Higher time due to initialization overhead compared to work done. |
| 5,000 | 3.191200 | 0.002174 | Larger sample size reduces time (less overhead effect), Pi closer to actual but still inaccurate due to low N. |
| 500,000,000 | 3.141589 | 2.475918 | Very close to actual value of π (3.141592...). More accurate but requires higher computation time due to very large dataset. |

**Analysis:**

Increasing number of points improves accuracy but increases runtime.

**Github Link:** <https://github.com/swanand2625/22510027_HPC_LAB>