**Name : Akshat Mahajan PRN : 22510003**

**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:**

**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 = 1000000;**

**int \*A = (int \*)malloc(n \* sizeof(int));**

**int \*B = (int \*)malloc(n \* sizeof(int));**

**int \*C = (int \*)malloc(n \* sizeof(int));**

**if (A == NULL || B == NULL || C == NULL) {**

**printf("Memory allocation failed!\n");**

**return 1;**

**}**

**for (int i = 0; i < n; i++) {**

**A[i] = i;**

**B[i] = i;**

**}**

**int threads[] = {1, 2, 4, 6, 8, 10};**

**for (int t = 0; t < 6; t++) {**

**omp\_set\_num\_threads(threads[t]);**

**double start\_time = omp\_get\_wtime();**

**#pragma omp parallel for**

**for (int i = 0; i < n; i++) {**

**C[i] = A[i] + B[i];**

**}**

**double end\_time = omp\_get\_wtime();**

**printf("Time taken with %d threads: %.6f seconds\n", threads[t], end\_time - start\_time);**

**}**

**free(A);**

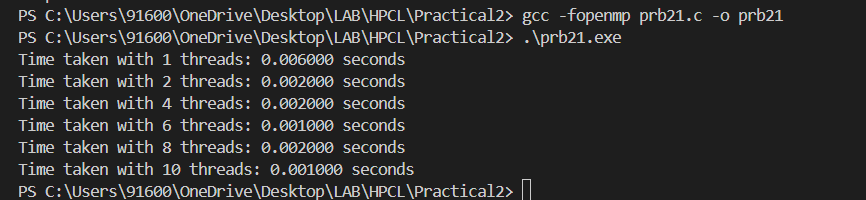
**free(B);**

**free(C);**

**return 0;**

**}**

**OUTPUT :**

****

**Analysis:**

**Observation: As the number of threads increases from 1 to 10, the execution time for the operation significantly reduces from 0.006s to 0.001s.  
Conclusion: This demonstrates effective parallelization using OpenMP, leading to improved performance and reduced runtime.**

**Problem Statement 2:**

**Screenshots:**

**#include <stdio.h>**

**#include <omp.h>**

**int main()**

**{**

**int num\_steps = 100000000;**

**double step = 1.0 / (double)num\_steps;**

**double sum;**

**double pi;**

**int threads[] = {1, 2, 4, 6, 8, 10};**

**for (int t = 0; t < 6; t++)**

**{**

**sum = 0.0;**

**omp\_set\_num\_threads(threads[t]);**

**double start\_time = omp\_get\_wtime();**

**#pragma omp parallel for reduction(+ : sum)**

**for (int i = 0; i < num\_steps; i++)**

**{**

**double x = (i + 0.5) \* step;**

**sum += 4.0 / (1.0 + x \* x);**

**}**

**pi = step \* sum;**

**double end\_time = omp\_get\_wtime();**

**printf("Pi with %d threads = %.15f | Time = %f seconds\n", threads[t], pi, end\_time - start\_time);**

**}**

**return 0;**

**}**

**OUTPUT :**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Analysis:**

**As the number of threads increases, the execution time to approximate π significantly decreases from 0.299s (1 thread) to ~0.12s (4–10 threads).  
This confirms that OpenMP parallelization with reduction improves performance, though speedup plateaus after a certain thread count due to overhead and hardware limits.**