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

**Year:** 2023-24 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 2**

**Exam Seat No: 2020BTECS00037**

**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:**

#include <stdio.h>

#include <omp.h>

#define VECTOR\_SIZE 10000

int main() {

    float vector[VECTOR\_SIZE];

    int scalar = 5;

    for (int i = 0; i < VECTOR\_SIZE; i++) {

        vector[i] = i + 100.987;

    }

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

    for (int i = 0; i < VECTOR\_SIZE; i++) {

        vector[i] += scalar;

    }

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

    printf("Serial Method Time: %f seconds\n", (end\_time\_serial - start\_time\_serial));

    for (int i = 0; i < VECTOR\_SIZE; i++) {

        vector[i] = i + 100.987;

    }

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

    #pragma omp parallel for private(scalar) num\_threads(100)

        for (int i = 0; i < VECTOR\_SIZE; i++) {

                vector[i] += scalar;

            }

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

    printf("Parallel Method Time: %f seconds\n", (end\_time\_parallel - start\_time\_parallel));

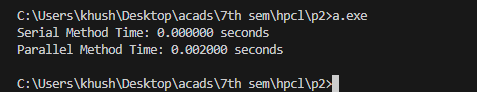
    return 0;

}

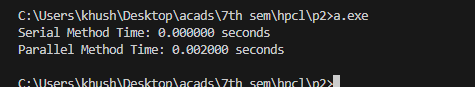
**Screenshots:**

Keeping number of threads constant and varying size of Data.

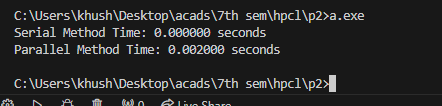
Threads = 8(default) Vector size = 100



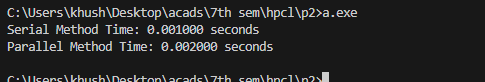
Threads = 8(default) Vector size = 1000



Threads = 8(default) Vector size = 10000

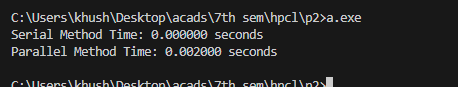


Threads = 8(default) Vector size = 100000

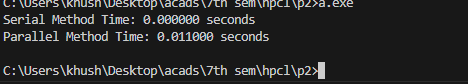


Keeping data constant and increasing number of threads.

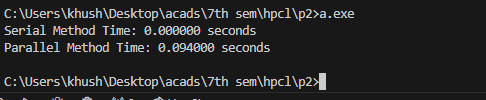
Threads = 10 Vector size = 10000



Threads = 100 Vector size = 10000



Threads = 1000 Vector size = 10000



**Information:**

Vector and scaler addition is to be performed using sequential and parallel approach. We have to analyse the time both approaches. For parallel approach analysis can be done in two ways, first by keeping data constant and varying number of threads and secondly by keeping number of threads constant and varying size of data.

**Analysis:**

1)    As we go on increasing the size of data the time it takes to execute in parallel also increases.

2)    By keeping data constant and increasing number for threads gradually increase the execution time due to increase in logical thread causes extra mapping time.

3)    Here Serial time is less than parallel because insufficient data for parallelism which causes extra overhead of communication time.

|  |  |  |  |
| --- | --- | --- | --- |
| Number of Threads | Data Size | Sequential Time | Parallel Time |
| 8 | 100 | 0.00000 | 0.002000 |
| 8 | 1000 | 0.00000 | 0.002000 |
| 8 | 10000 | 0.00000 | 0.002000 |
| 8 | 100000 | 0.00000 | 0.002000 |
| 10 | 10000 | 0.00000 | 0.002000 |
| 100 | 10000 | 0.00000 | 0.011000 |
| 1000 | 10000 | 0.00000 | 0.094000 |

**Problem Statement 2:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

int main() {

    int totalPoints = 10000000;

    int pointsInsideCircle = 0;

    double x, y;

    printf("Enter the number of terms: ");

    scanf("%d", &totalPoints);

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

    for (int i = 0; i < totalPoints; ++i) {

        x = (double)rand() / RAND\_MAX;

        y = (double)rand() / RAND\_MAX;

        if (x \* x + y \* y <= 1.0) {

            pointsInsideCircle++;

        }

    }

    double pi = 4.0 \* pointsInsideCircle / totalPoints;

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

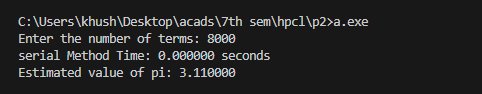
    printf("serial Method Time: %f seconds\n", (end\_time\_serial - start\_time\_serial));

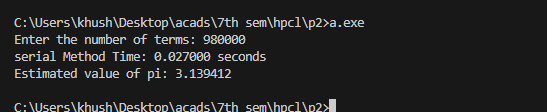
    printf("Estimated value of pi: %f\n", pi);

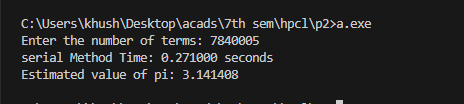
    return 0;

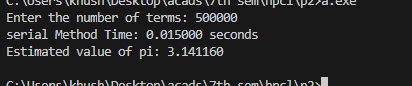
}

**Screenshots:**









**Parallel**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

int main() {

    int totalPoints = 10000000;

    int pointsInsideCircle = 0;

    double x, y;

    printf("Enter the number of terms: ");

    scanf("%d", &totalPoints);

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

    #pragma omp parallel for private(x, y) reduction(+:pointsInsideCircle)

    for (int i = 0; i < totalPoints; ++i) {

        x = (double)rand() / RAND\_MAX;

        y = (double)rand() / RAND\_MAX;

        if (x \* x + y \* y <= 1.0) {

            pointsInsideCircle++;

        }

    }

    double pi = 4.0 \* pointsInsideCircle / totalPoints;

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

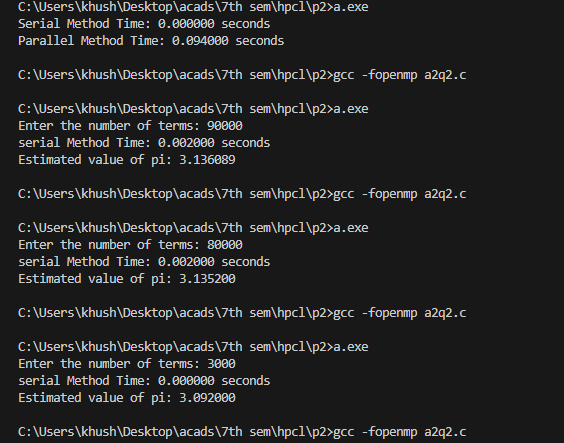
    printf("Parallel Method Time: %f seconds\n", (end\_time\_parallel - start\_time\_parallel));

    printf("Estimated value of pi: %f\n", pi);

    return 0;

}

**OUTPUT**

****