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

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

**Course:** High Performance Computing Lab

**Practical No. 6**

**Name:** Sumit Narake

**PRN:**2020BTECS00023

**Batch :** B2

**Title of practical: Implementation of OpenMP programs.**

Implement following Programs using OpenMP with C:

1. Implementation of Prefix sum.
2. Implementation of Matrix-Vector Multiplication.

**Problem Statement 1:**

**Serial Code:**

#include <stdio.h>

#include <stdlib.h> // Include the stdlib.h header for atoi function

#include <time.h>

void prefixSum(int arr[], int n)

{

int prefixSum[n];

prefixSum[0] = arr[0];

for (int i = 1; i < n; i++)

{

prefixSum[i] = prefixSum[i - 1] + arr[i];

}

printf("Prefix Sum Array:\n");

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

{

printf("%d ", prefixSum[i]);

}

printf("\n");

}

int main(int argc, char \*argv[])

{

if (argc != 2)

{

printf("Usage: %s [array\_size]\n", argv[0]);

return 1;

}

int n = atoi(argv[1]);

clock\_t start = clock();

printf("The number of elements in the array: %d\n", n);

printf("The elements of the array:\n");

int arr[n];

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

{

arr[i] = 10;

printf("%d ", arr[i]);

}

printf("\n");

prefixSum(arr, n);

clock\_t end = clock();

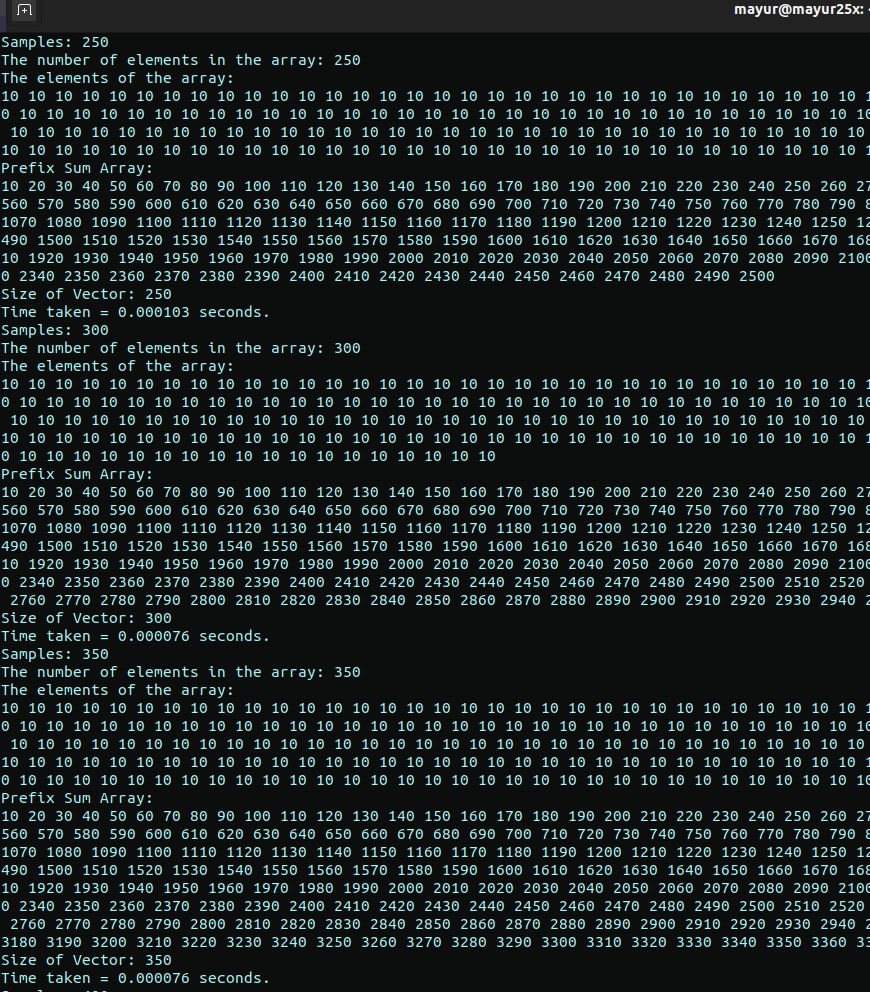
printf("Size of Vector: %d\n", n);

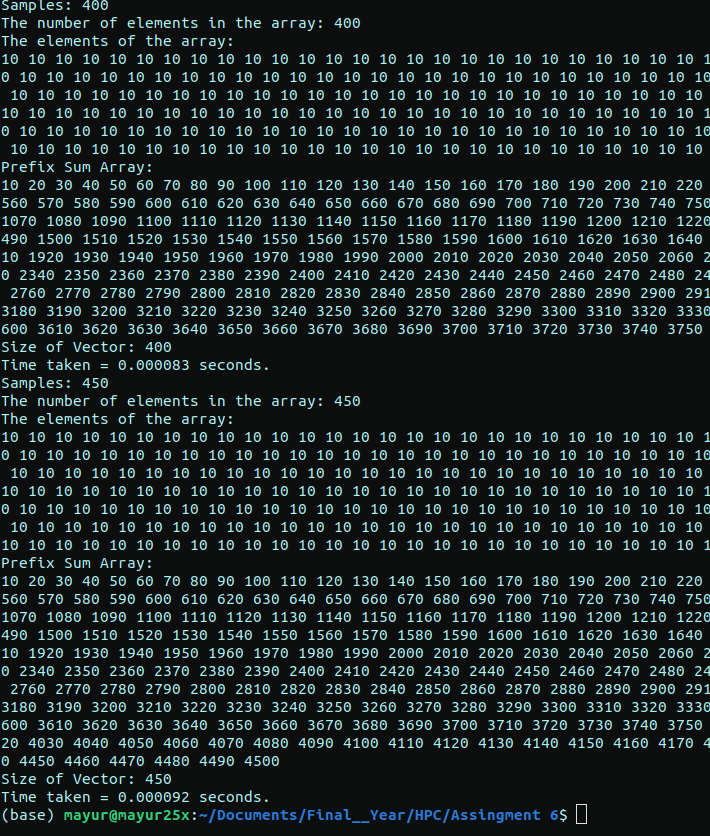
printf("Time taken = %f seconds.\n", (double)(end - start) / CLOCKS\_PER\_SEC);

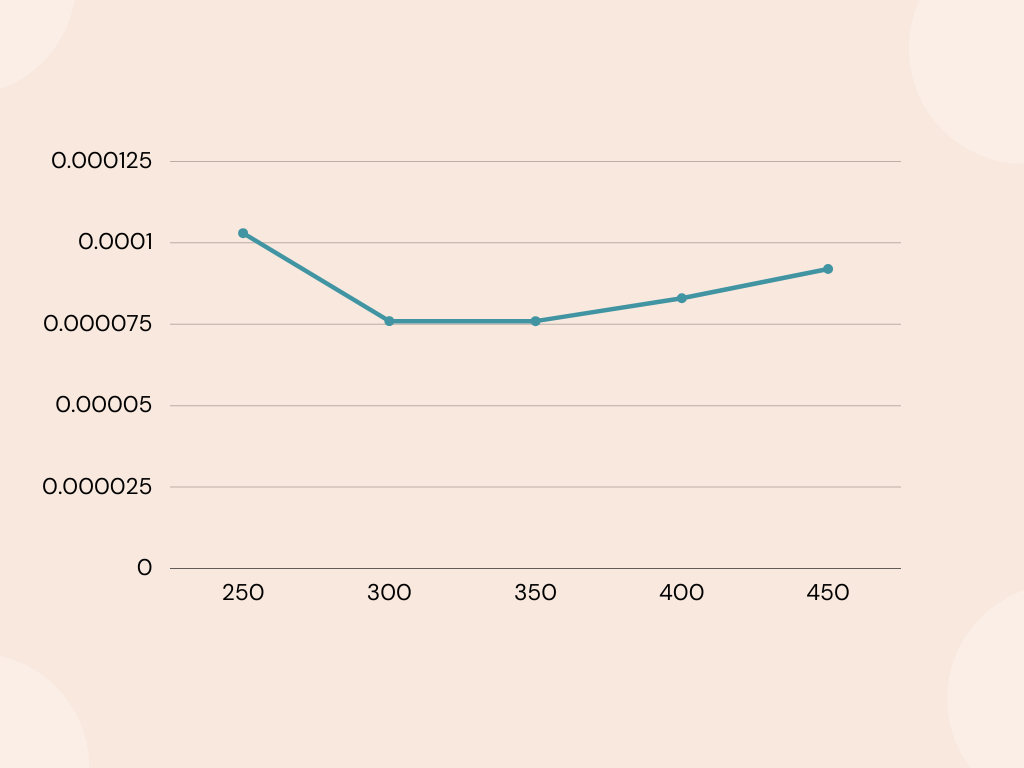
return 0;

}

**Output:**

****

****

****

**Parallel Code:**

#include <stdio.h>

#include <omp.h>

#include <time.h>

#include <stdlib.h>

void parallelPrefixSum(int arr[], int n)

{

int prefixSum[n];

prefixSum[0] = arr[0];

#pragma omp parallel for schedule(dynamic)

for (int i = 1; i < n; i++)

{

prefixSum[i] = prefixSum[i - 1] + arr[i];

}

// printf("Parallel Prefix Sum Array:\n");

// for (int i = 0; i < n; i++)

// {

// printf("%d ", prefixSum[i]);

// }

// printf("\n");

}

int main(int argc, char \*argv[])

{

if (argc != 3)

{

printf("Usage: %s [num\_threads] [array\_size]\n", argv[0]);

return 1;

}

int num\_threads = atoi(argv[1]);

int n = atoi(argv[2]);

omp\_set\_num\_threads(num\_threads);

clock\_t start = clock();

printf("The number of elements in the array: %d\n", n);

// printf("The elements of the array:\n");

int arr[n];

// for (int i = 0; i < n; i++)

// {

// arr[i] = 10;

// printf("%d ", arr[i]);

// }

parallelPrefixSum(arr, n);

clock\_t end = clock();

double t = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Size of Vector: %d\n", n);

printf("Number of threads: %d\n", num\_threads);

printf("Time taken = %f seconds.\n", t);

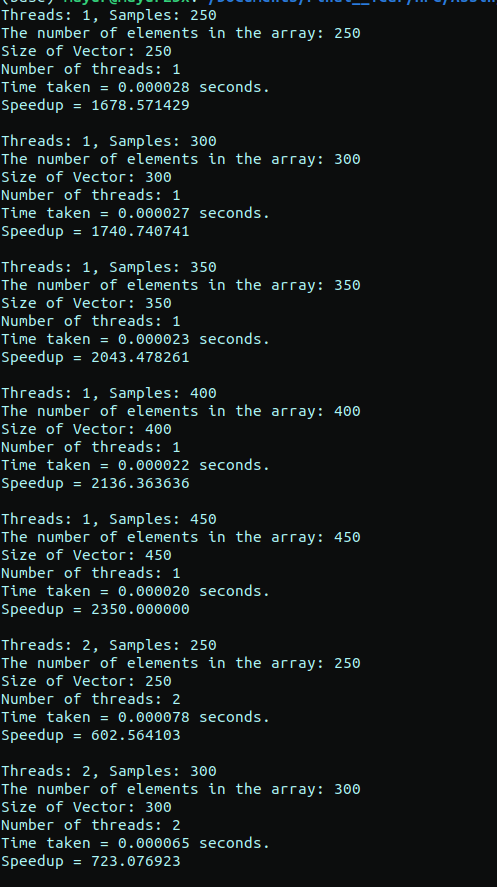
printf("Speedup = %f\n", 0.047 / t);

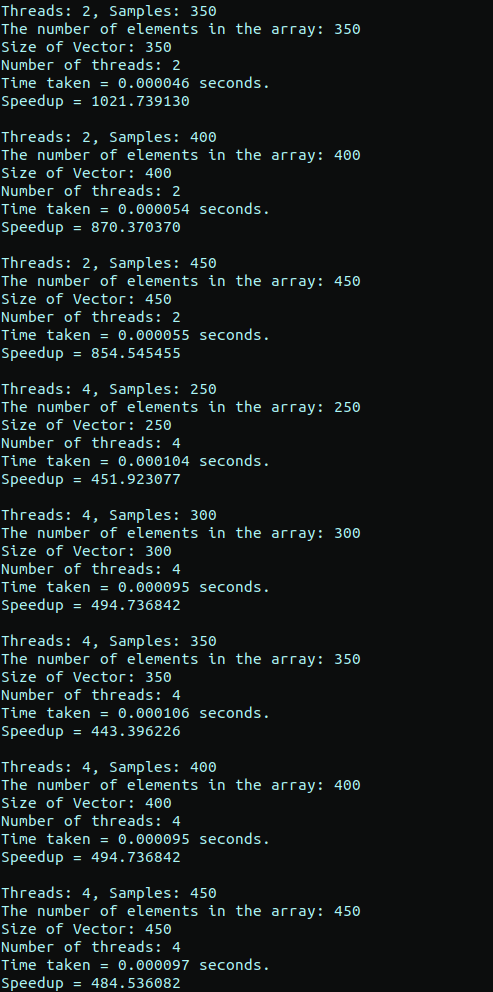
printf("\n");

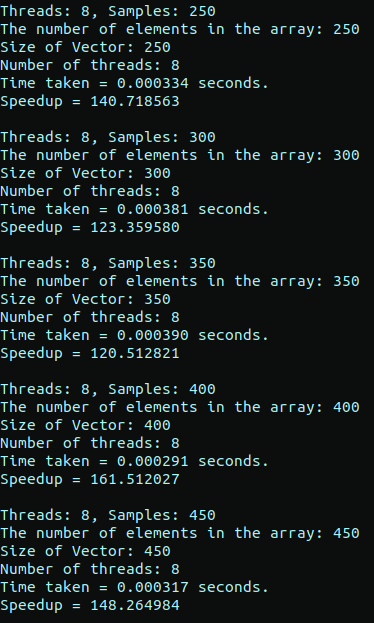
return 0;

}

**Output:**

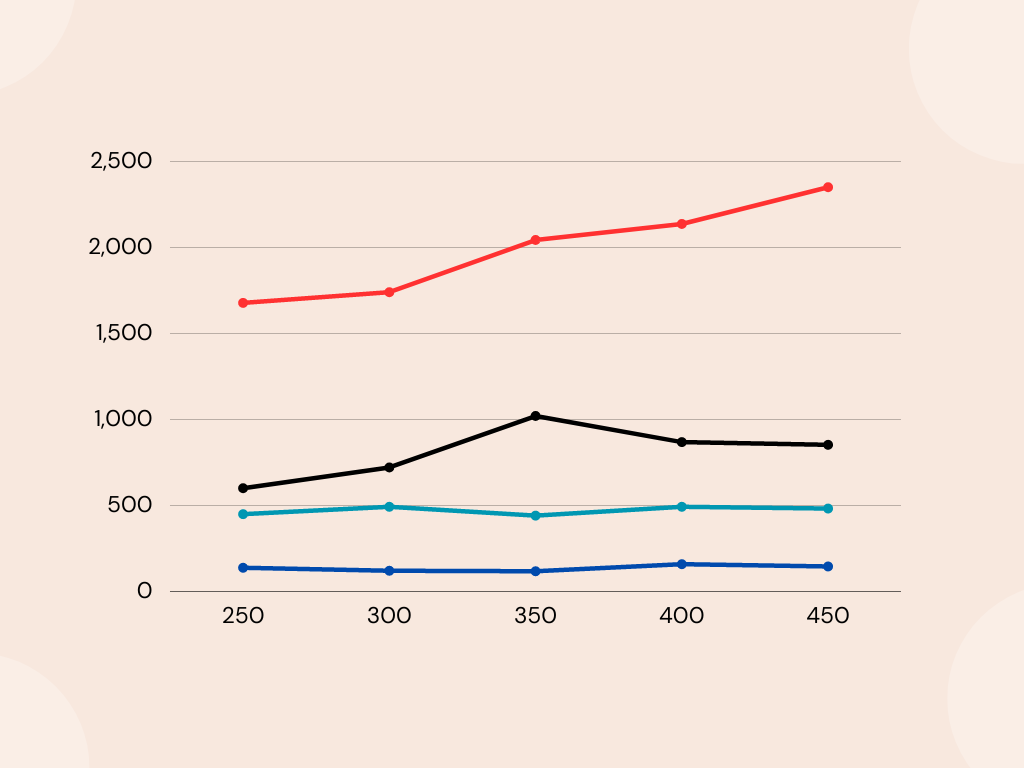
****

****

****

**Analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Threads** |  |  |  |
| **Inputs** | **1(red)** | **2(black)** | **4(pink)** | **8(blue)** |
| **250** | **1678.571429** | **602.564103** | **451.923077** | **140.718563** |
| **300** | **1740.740741** | **723.076923** | **494.736842** | **123.35958** |
| **350** | **2043.478261** | **1021.73913** | **443.396226** | **120.512821** |
| **400** | **2136.363636** | **870.37037** | **494.736842** | **161.512027** |
| **450** | **2350** | **854.545455** | **484.536082** | **148.264984** |

****

**Problem Statement 2:**

**Serial Code:**

#include<stdio.h>

#include<time.h>

#define n 200

intmain() {

    clock\_tstart=clock();

    printf("The size of matrix: %d\n",n);

    intmatrix[n][n];

    intvector[n];

    intresult[n];

    printf("The elements of the matrix:\n");

    for (inti = 0; i<n; i++) {

        for (intj = 0; j<n; j++) {

            matrix[i][j]=2;

        }

    }

    printf("The elements of the vector:\n");

    for (inti = 0; i<n; i++) {

        vector[i]=1;

    }

    for (inti = 0; i<n; i++) {

        result[i] = 0;

        for (intj = 0; j<n; j++) {

            result[i] += matrix[i][j] \* vector[j];

        }

    }

    printf("Result vector:\n");

    for (inti = 0; i<n; i++) {

        printf("%d ", result[i]);

    }

    printf("\n");

    clock\_tend=clock();

    doublet=(double)(end-start)/CLOCKS\_PER\_SEC;

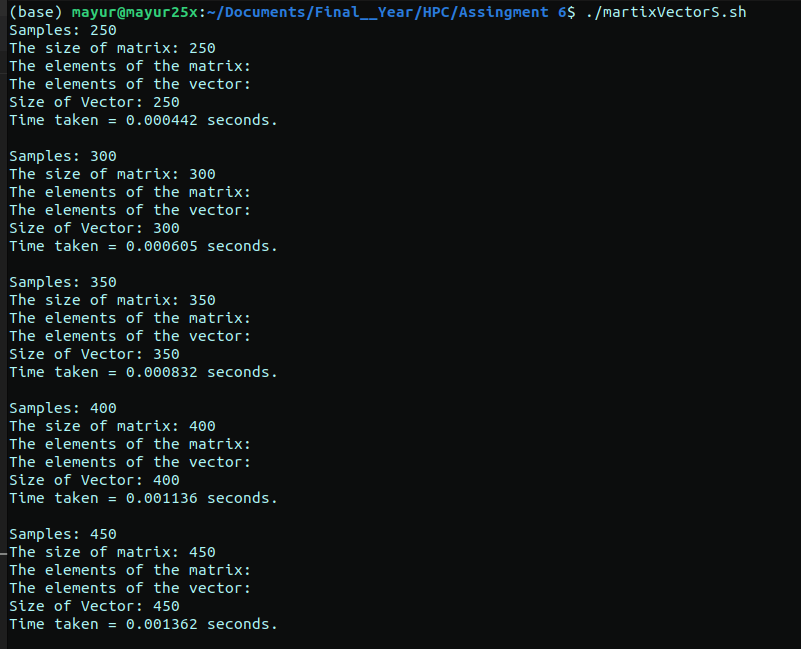
    printf("Size of Vector: %d\n",n);

    printf("Time taken = %f seconds.\n", t);

    return0;

}

**Output:**

****

**Parallel Code:**

#include <stdio.h>

#include <omp.h>

#include <time.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

if (argc != 3)

{

printf("Usage: %s [num\_threads] [matrix\_size]\n", argv[0]);

return 1;

}

int num\_threads = atoi(argv[1]);

int n = atoi(argv[2]);

omp\_set\_num\_threads(num\_threads);

clock\_t start = clock();

printf("The size of matrix: %d\n", n);

int matrix[n][n];

int vector[n];

int result[n];

printf("The elements of the matrix:\n");

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

{

for (int j = 0; j < n; j++)

{

matrix[i][j] = 2;

}

}

printf("The elements of the vector:\n");

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

{

vector[i] = 1;

}

#pragma omp parallel for

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

{

result[i] = 0;

for (int j = 0; j < n; j++)

{

result[i] += matrix[i][j] \* vector[j];

}

}

// printf("Result vector:\n");

// for (int i = 0; i < n; i++)

// {

// printf("%d ", result[i]);

// }

clock\_t end = clock();

double t = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Size of Vector: %d\n", n);

printf("Number of threads: %d\n", num\_threads);

printf("Time taken = %f seconds.\n", t);

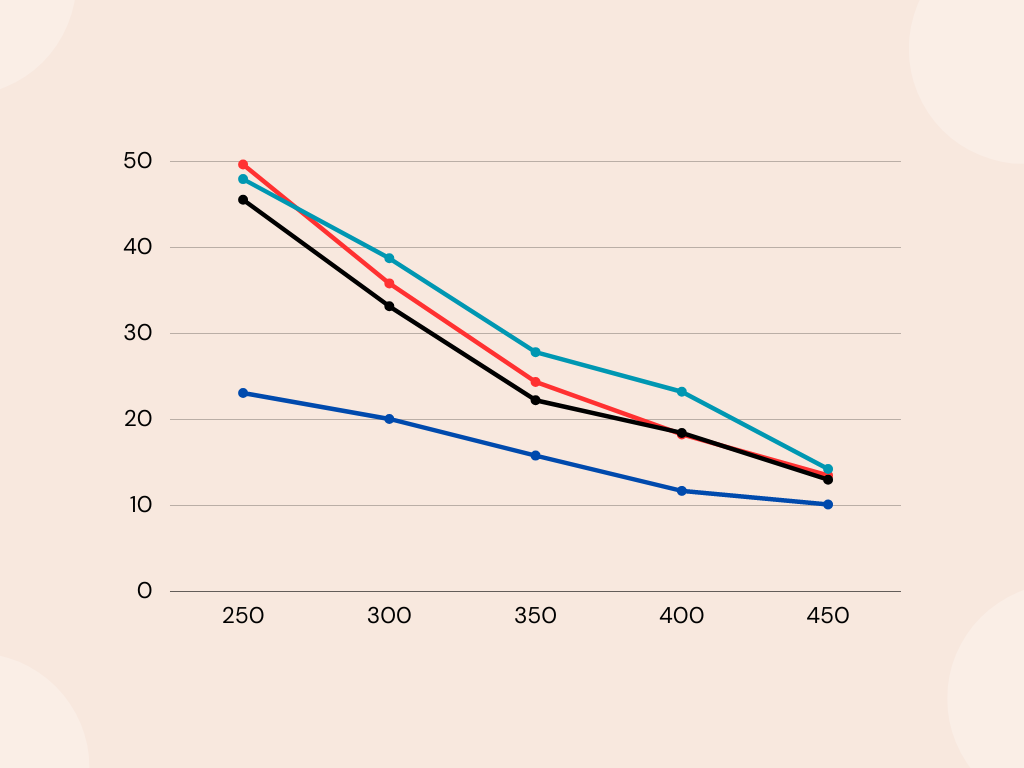
printf("Speedup = %f\n", 0.021 / t);

printf("\n");

return 0;

}

**Output:**

****

**Analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Threads** |  |  |
| **Input** | **1(red)** | **2(black)** | **4(pink)** | **8(blue)** |
| **250** | **49.64539** | **45.553145** | **47.945205** | **23.10231** |
| **300** | **35.836177** | **33.175355** | **38.745387** | **20.095694** |
| **350** | **24.390244** | **22.269353** | **27.851459** | **15.837104** |
| **400** | **18.292683** | **18.469657** | **23.255814** | **11.738401** |
| **450** | **13.522215** | **13.035382** | **14.276003** | **10.159652** |