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

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

**Course:** High Performance Computing Lab

**Practical No. 6**

**Name: Siddhi Hagawane**

**Exam Seat No: 2020BTECS00049**

**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:**

Implementation of Prefix sum.

**Screenshots:**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

void prefix\_sum\_serial(int \*array, int n) {

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

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

    int i, sum = 0;

    for (i = 0; i < n; i++) {

        sum += array[i];

        temp[i] = sum;

    }

    for (i = 0; i < n; i++) {

        array[i] = temp[i];

    }

    free(temp);

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

    printf("\nArray size %d, Serial Time: %f seconds\n", n, end\_time\_serail - start\_time\_serial);

}

void prefix\_sum\_parallel(int \*array, int n, int num\_thread) {

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

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

    int i, sum = 0;

    #pragma omp parallel for num\_threads(num\_thread)

    for (i = 0; i < n; i++) {

        #pragma omp atomic

        sum += array[i];

        temp[i] = sum;

    }

    #pragma omp parallel for num\_threads(num\_thread)

    for (i = 0; i < n; i++) {

        array[i] = temp[i];

    }

    free(temp);

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

    printf("\nArray size %d, Number of threads %d, Parallel Time: %f seconds\n", n, num\_thread, end\_time\_parallel - start\_time\_parallel);

}

int main() {

    int n = 10000;

    int arr[n];

    for(int i=0; i<n; i++){

        arr[i] = i+1;

    }

    printf("\n");

    prefix\_sum\_serial(arr, n);

    for(int i=0; i<10; i++){

        arr[i] = i+1;

    }

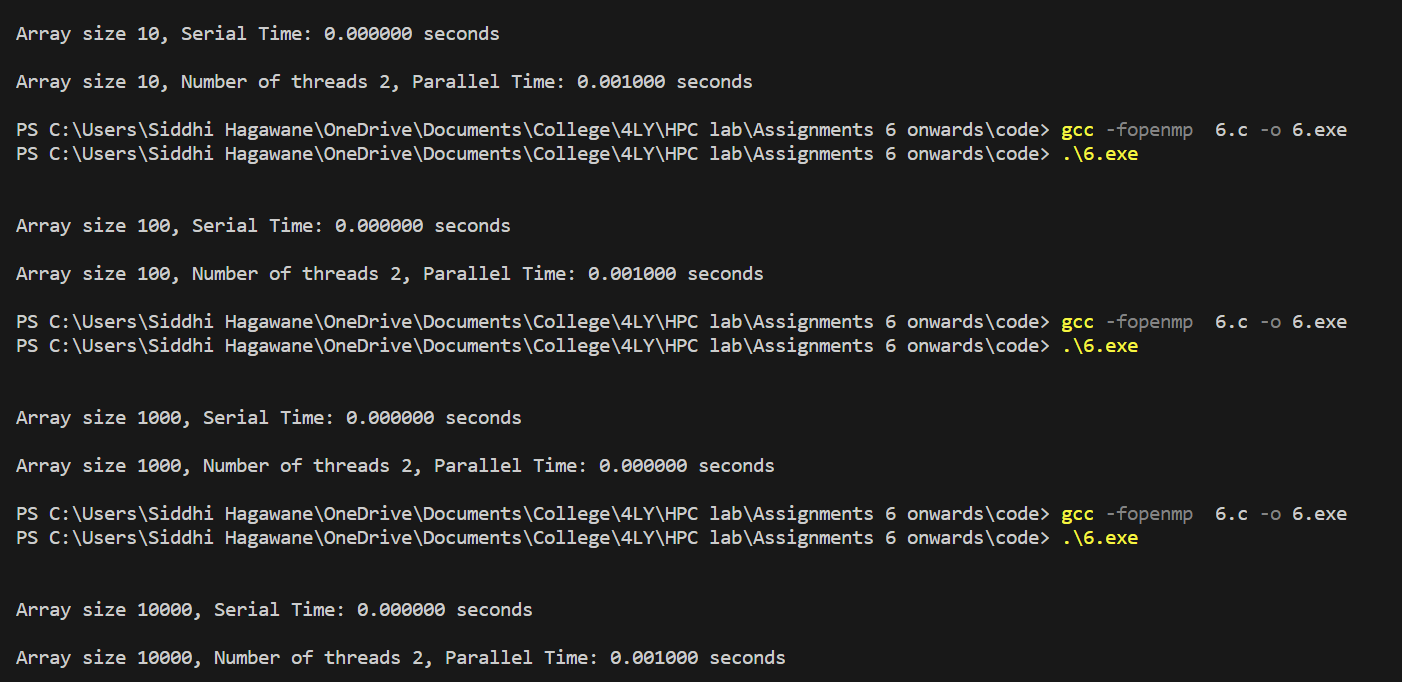
    prefix\_sum\_parallel(arr, n,4);

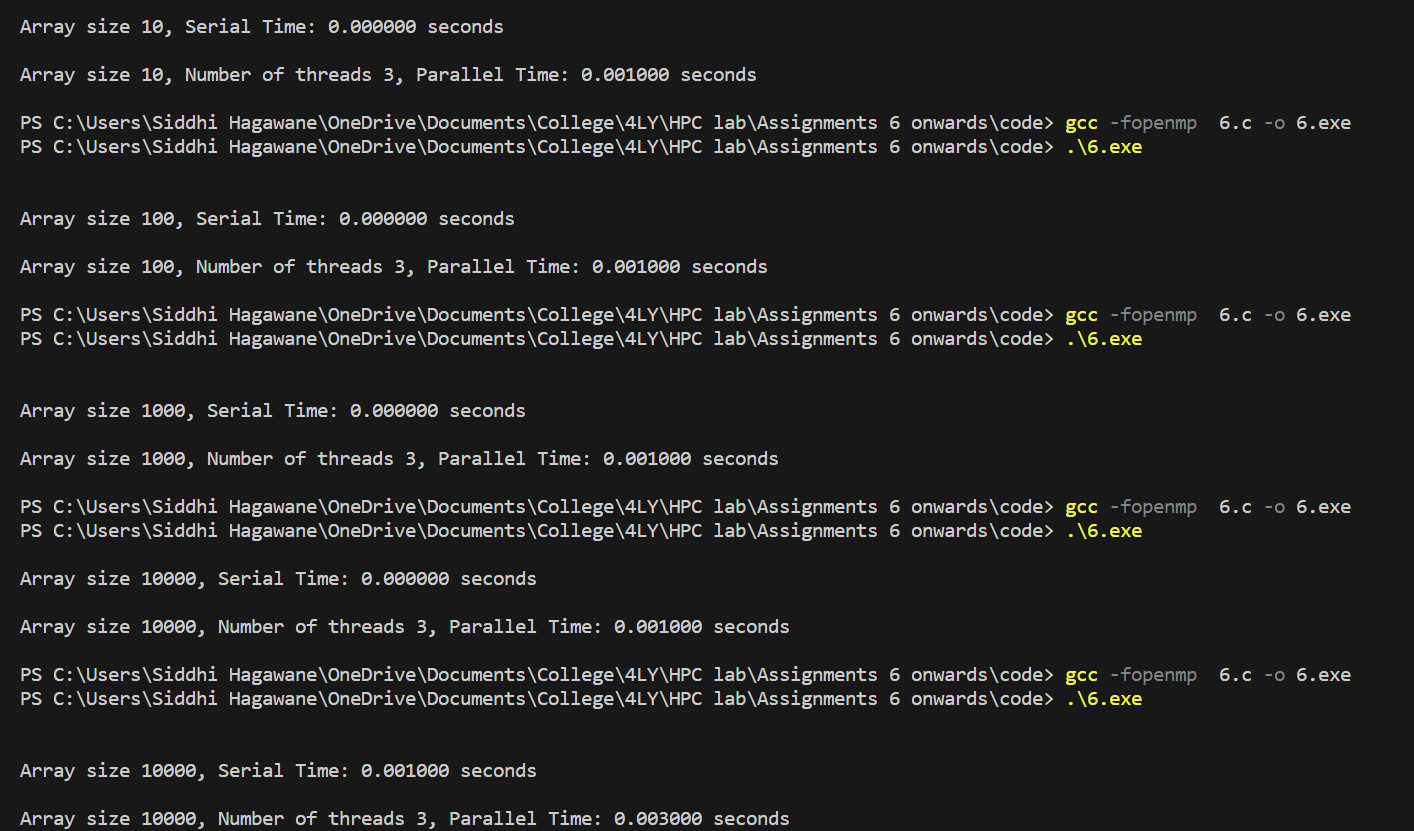
    printf("\n");

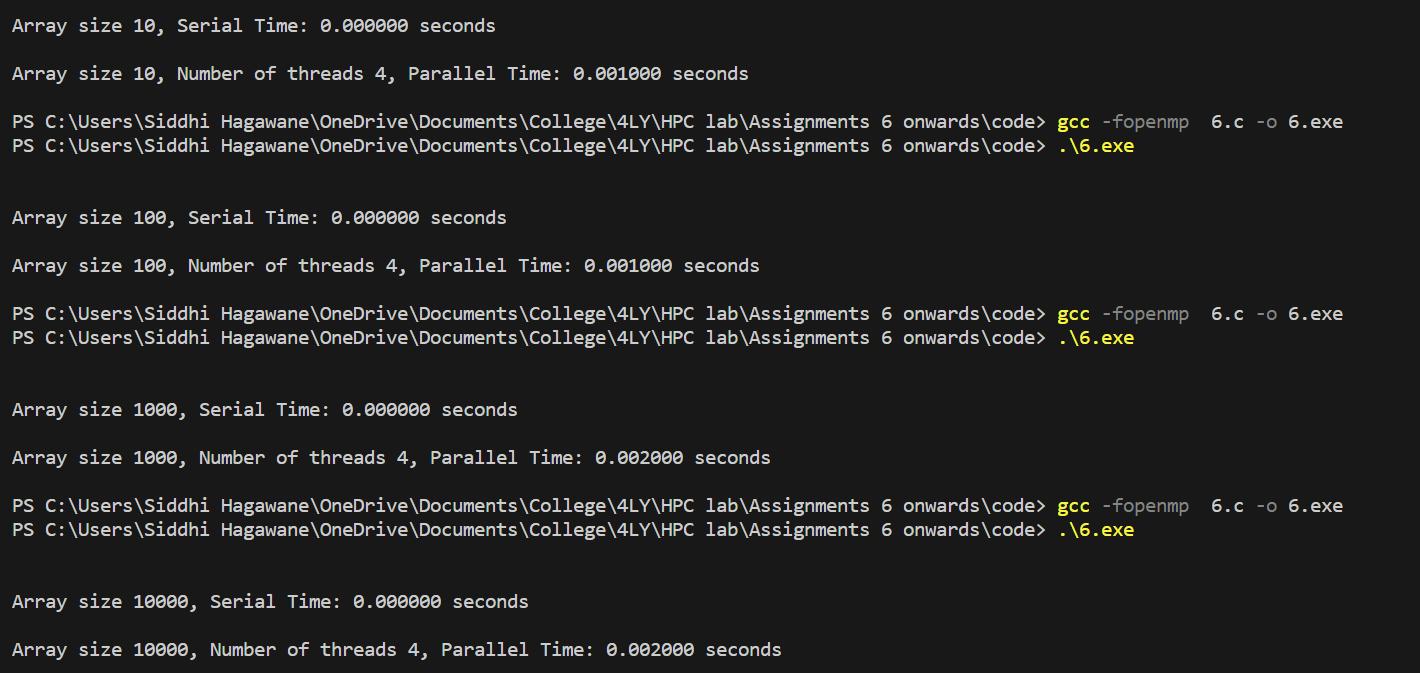
    return 0;

}

**OutPut:**

****

****

****

**Information:**

In prefix sum problem time, need to calculate sum of all numbers present in before blocks of array.

It takes O(n) time in serial computation.

**Analysis:**

1. Serial time takes small time as compared to parallel time. Due to as parallel programming required dividing code into different tasks which take more time, so that parallel time get increase.
2. As number of threads increase parallel time also increasing correspondingly.
3. Serial computation is time conserving instead of parallel program for this computation.

**Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads** | **Array Size** | **Serial Time** | **Parallel Time** | **Speedup** |
| **2** | **10** | **0** | **0.001** | **0** |
| **2** | **100** | **0** | **0.001** | **0** |
| **2** | **1000** | **0** | **0.000** | **0** |
| **2** | **10000** | **0** | **0.001** | **0** |
| **3** | **10** | **0** | **0.001** | **0** |
| **3** | **100** | **0** | **0.001** | **0** |
| **3** | **1000** | **0** | **0.001** | **0** |
| **3** | **10000** | **0** | **0.001** | **0** |
| **4** | **10** | **0** | **0.001** | **0** |
| **4** | **100** | **0** | **0.001** | **0** |
| **4** | **1000** | **0** | **0.002** | **0** |
| **4** | **10000** | **0** | **0.002** | **0** |

**Graph:**

**Problem Statement 2:**

Implementation of Matrix-Vector Multiplication.

**Screenshots:**

**Code for Serial:**

#include <stdio.h>

#include <omp.h>

#define N 400 // Size of the matrix and vector

void matrixVectorMult(double matrix[N][N], double vector[N], double result[N]) {

    for (int i = 0; i < N; i++) {

        result[i] = 0.0;

        for (int j = 0; j < N; j++) {

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

        }

    }

}

int main() {

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

    double matrix[N][N];

    double vector[N];

    double result[N];

    for (int i = 0; i < N; i++) {

        vector[i] = i;

        for (int j = 0; j < N; j++) {

            matrix[i][j] = i + j;

        }

    }

    matrixVectorMult(matrix, vector, result);

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

    printf("\nValue of N %d, Serial Time: %f seconds\n", N, end\_time\_serail - start\_time\_serial);

    return 0;

}

**Code for Parallel:**

#include <stdio.h>

#include <omp.h>

#define N 400 // Size of the matrix and vector

#define num\_thread 4

void matrixVectorMult(double matrix[N][N], double vector[N], double result[N]) {

    #pragma omp parallel for num\_threads(num\_thread)

    for (int i = 0; i < N; i++) {

        result[i] = 0.0;

        for (int j = 0; j < N; j++) {

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

        }

    }

}

int main() {

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

    double matrix[N][N];

    double vector[N];

    double result[N];

    for (int i = 0; i < N; i++) {

        vector[i] = i;

        for (int j = 0; j < N; j++) {

            matrix[i][j] = i + j;

        }

    }

    matrixVectorMult(matrix, vector, result);

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

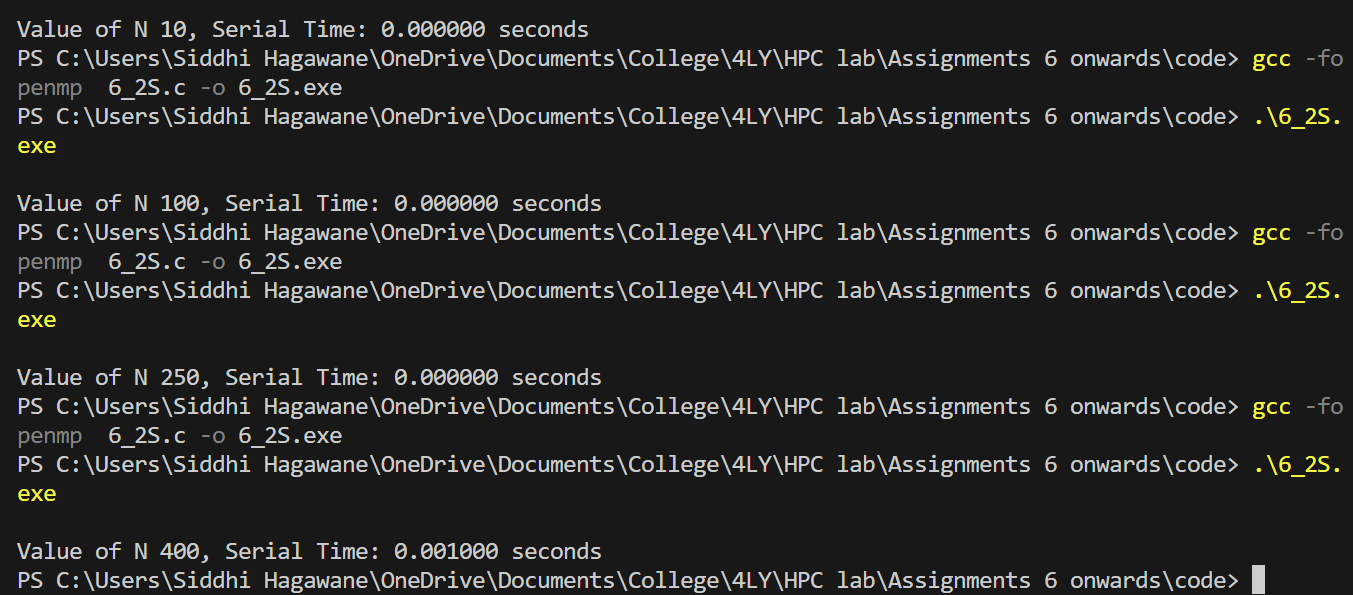
    printf("\nValue of N %d, Number of threads %d, Parallel Time: %f seconds\n", N, num\_thread, end\_time\_parallel - start\_time\_parallel);

    return 0;

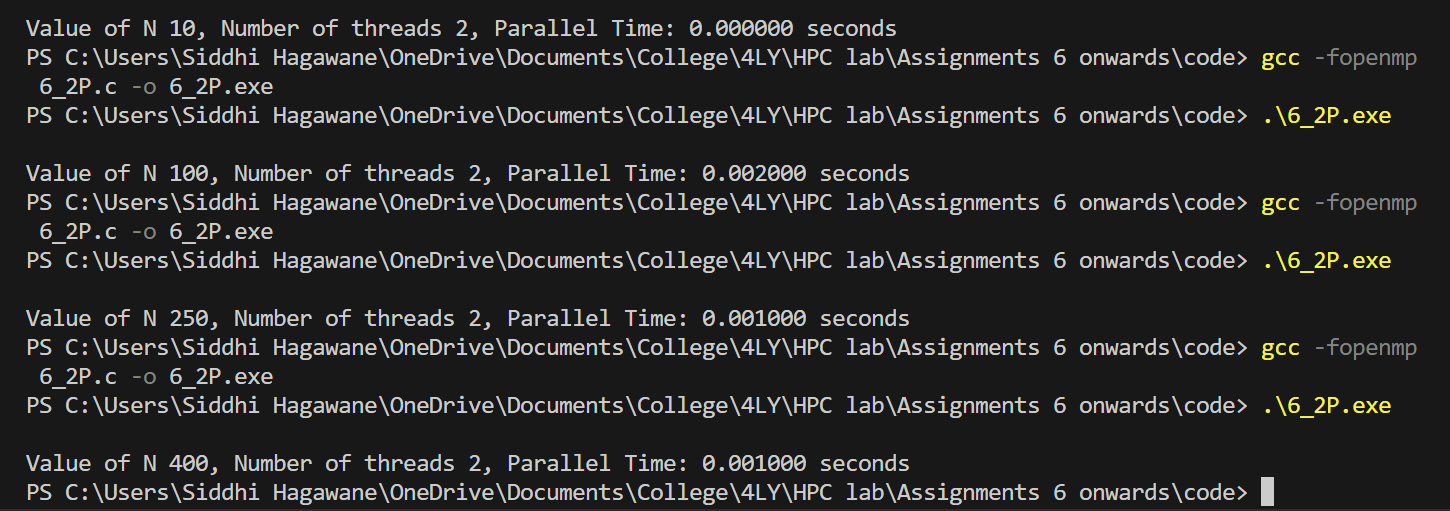
}

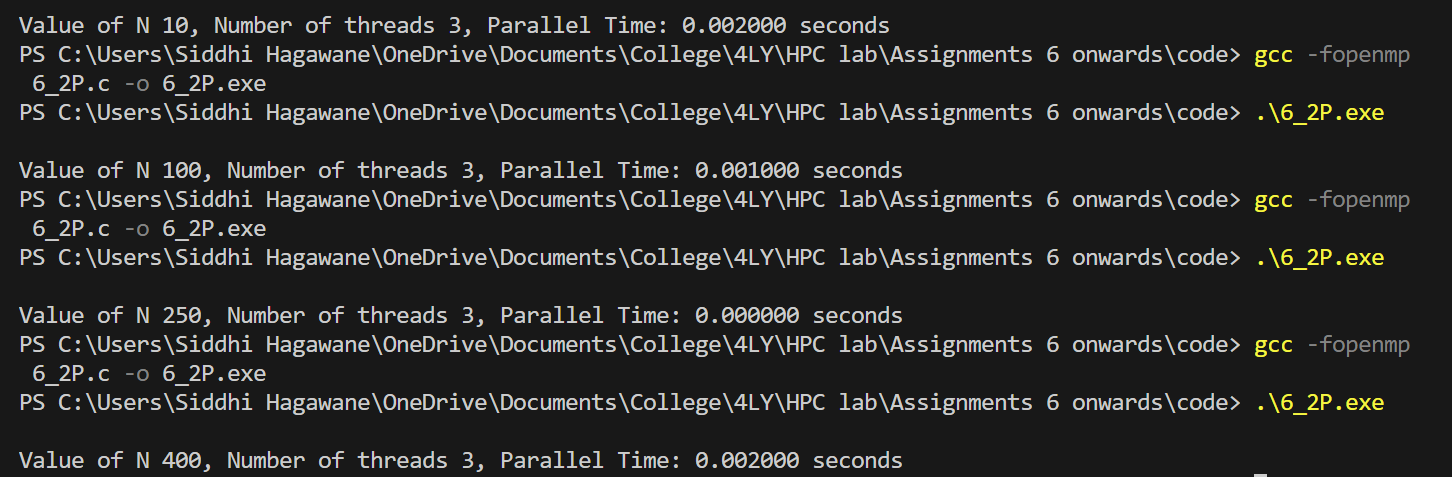
**OutPut:**

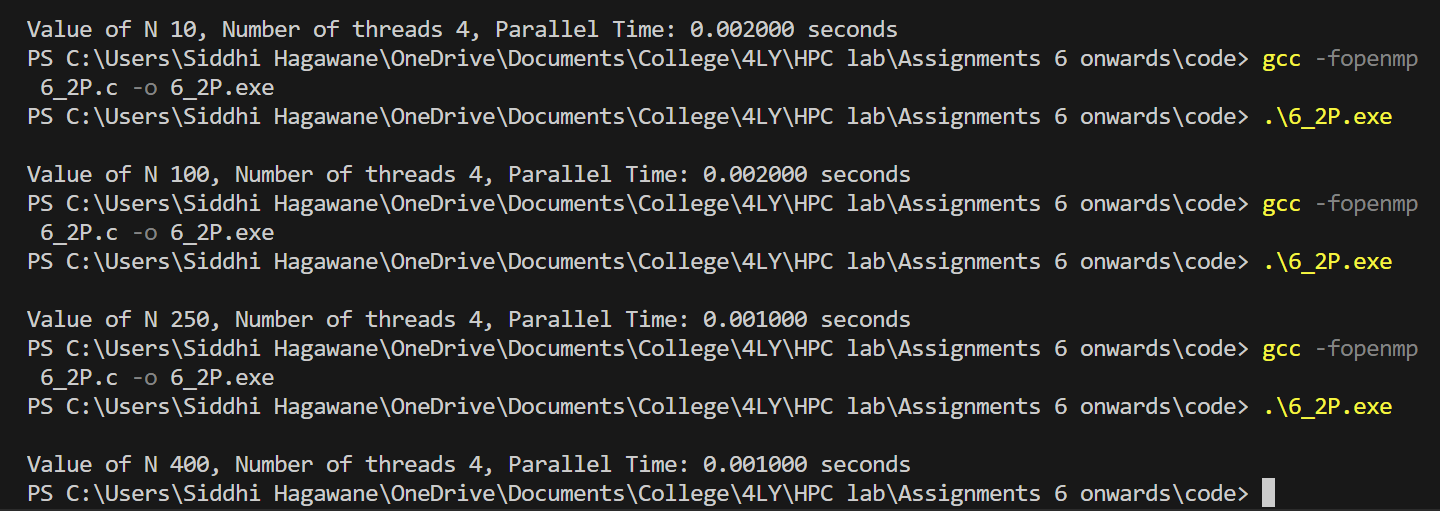
**Serial:**

****

**Parallel:**

****

****

****

**Information:**

Matrix vector multiplication take O(n2) time. Only nearly N size can be valid till 450, above it will support by system of 4 core system.

Serial time required take less as compared to parallel time due to size of matrix.

In above example size ranges from 10, 100, 250 to 400 with corresponding changing

Number of threads 2, 3 and 4 , because this system is 4-core system.

Serial and parallel is calculated for each possible cases.

By observing time calculated from above program serial time take less time as compared to parallel due to overhead problem.

**Analysis:**

**Performance:** The performance gain from parallelization will depend on the number of CPU cores available and the size of the matrices. For small matrices, the overhead of thread creation and synchronization may outweigh the benefits of parallelism. For large matrices, parallel execution can lead to significant speedup

Bs observing below table and by comparing serial time, parallel with number of threads and array size,

1. Time required by serial program is less as compared to parallel program.
2. As vector size(N) increase corresponding serial time as well as parallel time increasing, due to increasing input size.
3. Some where like vector size 10 and number of thread is 2 , due increasing in overhead for dividing small amount of input that is 10, so that time take more.
4. As number of threads increasing, time required less but due to overhead for small amount of input data it takes quite time.

**Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads** | **Vector Size(N)** | **Serial Time** | **Parallel Time** | **Speedup** |
| **2** | **10** | **0** | **0.000** | **0** |
| **2** | **100** | **0** | **0.002** | **0** |
| **2** | **250** | **0** | **0.001** | **0** |
| **2** | **400** | **0.001** | **0.001** | **1** |
| **3** | **10** | **0** | **0.002** | **0** |
| **3** | **100** | **0** | **0.001** | **0** |
| **3** | **250** | **0** | **0.000** | **0** |
| **3** | **400** | **0.001** | **0.002** | **0.5** |
| **4** | **10** | **0** | **0.002** | **0** |
| **4** | **100** | **0** | **0.002** | **0** |
| **4** | **250** | **0** | **0.001** | **0** |
| **4** | **400** | **0.001** | **0.001** | **1** |

**Graph:**

**GitHub Link:**

<https://github.com/Siddhish16/HPC-Assignments>