Assignment No. 6

PRN: 2020BTECS00057
Name: Mrunal Anil Khade
Course: High Performance Computing Lab
Title of practical: Study and implementation of Open MP program.

Implement following Programs using OpenMP with C:
1. Implementation of Prefix sum.
2. Implementation of Matrix-Vector Multiplication.

1. Implementation of Prefix sum.

```c
#include <stdio.h>
#include <omp.h>
#include <time.h>

void prefixSum(int *arr, int n) {
   int i, sum = 0;
// #pragma omp parallel for num_threads(4)
   for (i = 0; i < n; i++) {
#pragma omp critical
       {
           sum += arr[i];
           arr[i] = sum;
       }
   }
}

int main() {
   #pragma omp parallel for num_threads()
   double startTime = omp_get_wtime();
   int n = 10;
   int arr[n];
   for (int i = 1; i <= n; i++) {
       arr[i - 1] = i;
   }
   prefixSum(arr, n);
```

```
    printf("Prefix Sum:");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    printf("Threads: 8");
     printf("\n");
    double endTime = omp_get_wtime();
    printf("Execution time: %f", endTime - startTime);
     printf("\n");
    return 0;
}
```

```
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :10
Threads: 4
Execution time: 0.000038
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :100
Threads: 4
Execution time: 0.000039
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :1000
Threads: 4
Execution time: 0.000048
```

```
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :10
Threads: 8
Execution time: 0.000037
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :100
Threads: 8
Execution time: 0.000039
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :1000
Threads: 8
Execution time: 0.000052
```

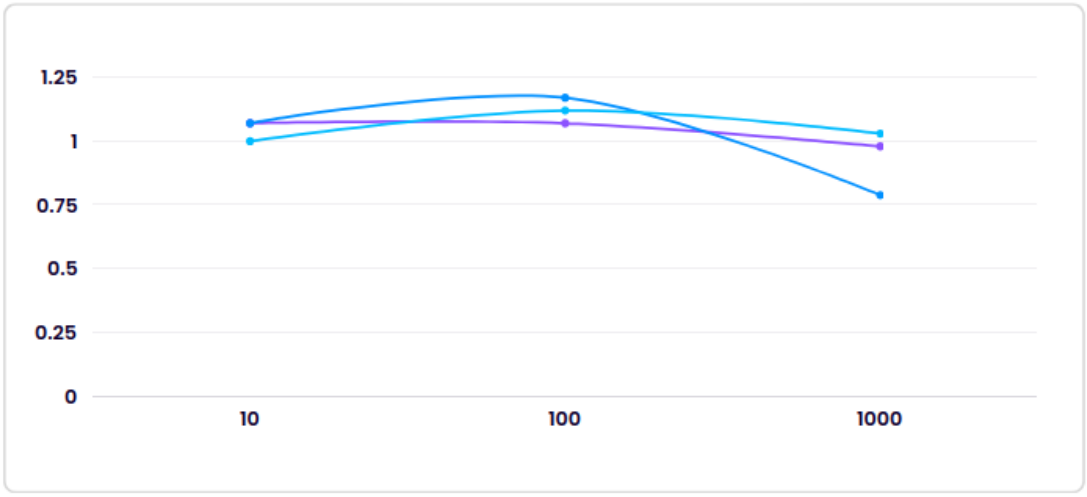
```
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :10
Threads: 2
Execution time: 0.000038
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :100
Threads: 2
Execution time: 0.000039
mrunal@mrunal:~/Desktop/HPC_$ gcc 6_1.c -lgomp -o test.exe
mrunal@mrunal:~/Desktop/HPC_$ ./test.exe
data Size :1000
Threads: 2
Execution time: 0.000053
```

Parallel

	10	100	1000
2	0.000038	0.000039	0.000053
4	0.000038	0.000039	0.000048
8	0.000037	0.000039	0.000052

Serial

	10	100	1000
2	0.000041	0.000042	0.000052
4	0.000041	0.000046	0.000038
8	0.000037	0.000044	0.000054




```c
#include <omp.h>
#include<time.h>

#define N 4 // Matrix size (N x N)

void matrixVectorMult(int matrix[N][N], int vector[N], int result[N]) {
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        result[i] = 0; // Initialize the result for this row

        for (int j = 0; j < N; j++) {
            result[i] += matrix[i][j] * vector[j];
        }
    }
}

int main() {
  double start=omp_get_wtime();
    int matrix[N][N] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
```

```
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };

    int vector[N] = {1, 2, 3, 4};
    int result[N];

    matrixVectorMult(matrix, vector, result);

    printf("Matrix-Vector Multiplication Result:\n");
    for (int i = 0; i < N; i++) {
        printf("%d ", result[i]);
    }
    printf("\n");
    double end=omp_get_wtime();

    printf("Execution time: %f", end-start);

    return 0;
}
```

Information:

Matrix-Vector Multiplication Function:
- This function takes three parameters:
  - `matrix[N][N]`: A 2D array representing the matrix.
  - `vector[N]`: An array representing the vector.
  - `result[N]`: An array to store the result of the matrix-vector multiplication.

Parallelization Using OpenMP:
- `#pragma omp parallel for`: This directive tells OpenMP to parallelize the following `for` loop. OpenMP will automatically distribute the loop iterations among multiple threads.

Matrix-Vector Multiplication:
- Inside the parallel `for` loop:
  - Each thread is responsible for computing a single row of the result vector.
  - `result[i]` is initialized to 0 for each row.
  - The inner loop computes the dot product of a row of the matrix and the vector. The result is stored in the corresponding position of the result vector.

No. of Threads	Speedup
2	0.954545
4	0.875000
8	0.913043
10	0.807692