HPCL ASSIGNMENT\_2

Name:Shashank Shenoy

PRN:22510013

Batch:B5

**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.

PB\_1:

Code :

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

int main() {

    int n = 10000000;

    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);

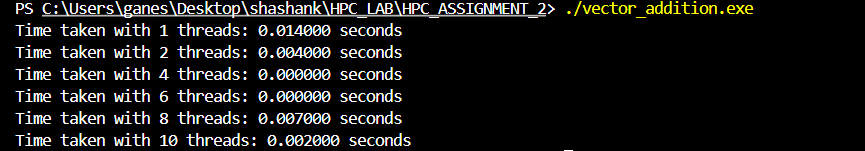
    free(B);

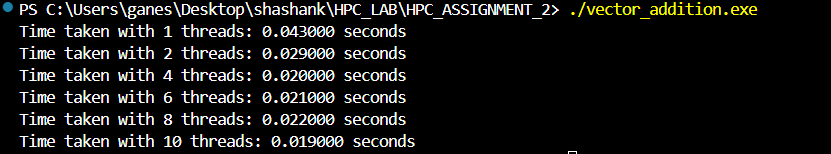
    free(C);

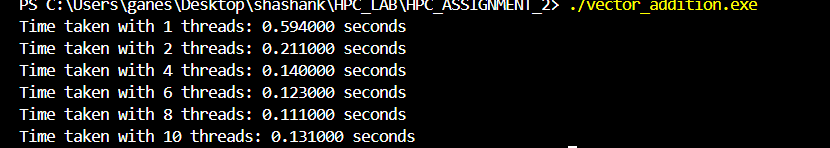
    return 0;

}

Output:







Analysis

|  |  |  |
| --- | --- | --- |
| **Data Size** | **Threads** | **Time in seconds** |
| 1000000 | 1  2  4  6  8  10 | 0.014  0.004  0.000  0.000  0.007  0.002 |
| 10000000 | 1  2  4  6  8  10 | 0.043  0.029  0.020  0.021  0.022  0.019 |
| 100000000 | 1  2  4  6  8  10 | 0.594  0.211  0.140  0.123  0.111  0.131 |

As data size increases time taken also increases and as number of threads increases the time taken decreases but some fluctuation can be seen after 6 threads

Pb\_2 :

Code:

#include <stdio.h>

#include <omp.h>

int main() {

    int num\_steps=1000000;

    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();

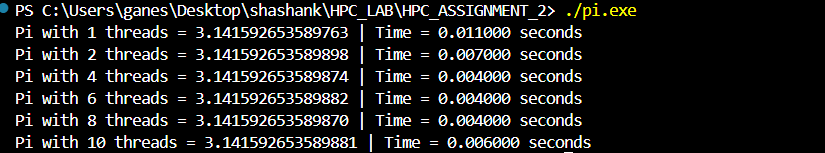
        printf("Pi with %d threads = %.15f | Time = %f seconds\n", threads[t], pi, end\_time - start\_time);

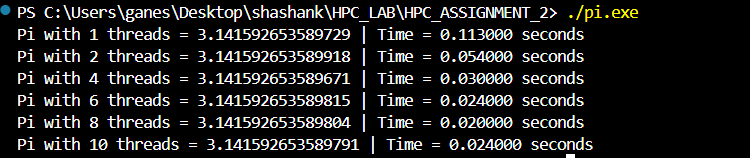
    }

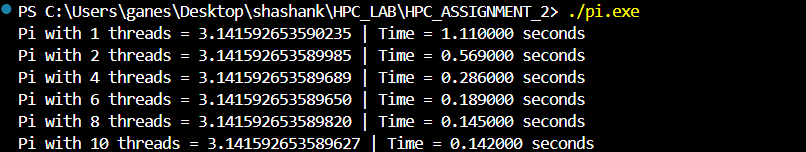
    return 0;

}

Output :







Analysis:

|  |  |  |
| --- | --- | --- |
| **Data Size** | **Threads** | **Time in seconds** |
| 1000000 | 1  2  4  6  8  10 | 0.011  0.007  0.004  0.004  0.004  0.006 |
| 10000000 | 1  2  4  6  8  10 | 0.113  0.054  0.030  0.024  0.020  0.024 |
| 100000000 | 1  2  4  6  8  10 | 1.11  0.56  0.28  0.18  0.145  0.142 |

As data size increases time taken also increases and as number of threads increases the time taken decreases but some fluctuation can be seen after 6 threads