PARALLEL AND DISTRIBUTIVE COMPUTING (UCS645)

Lab – 1

By

Mannya Soni

102316067

3P13

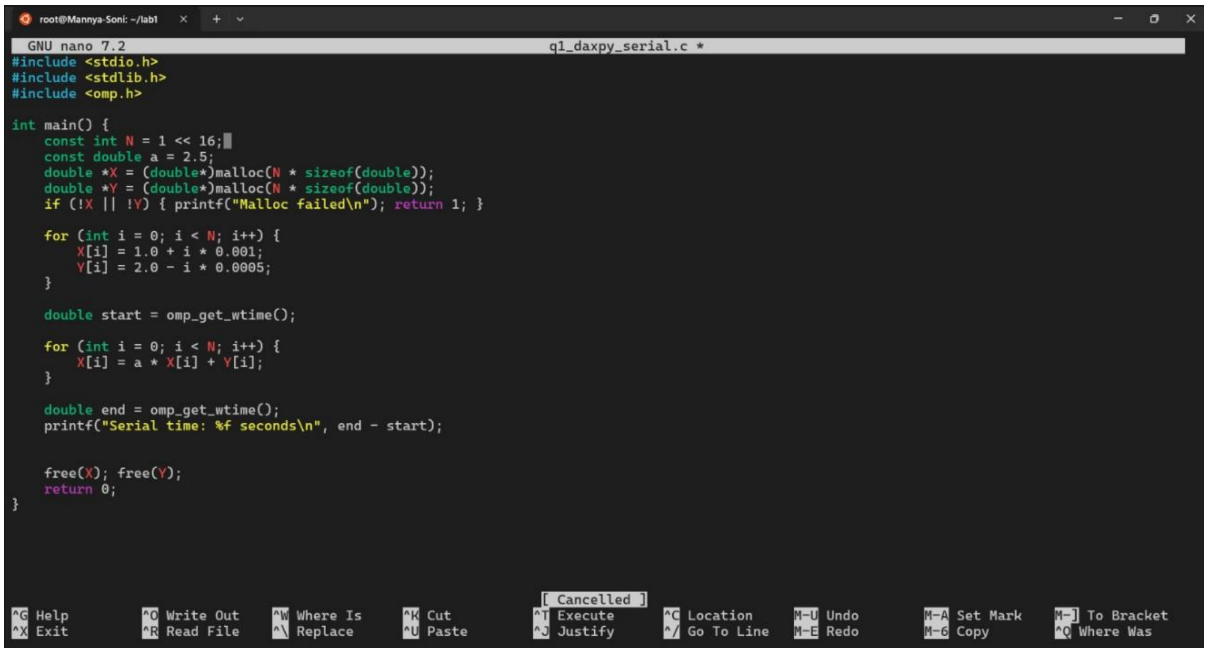
Instructor: Dr. Saif Nalband

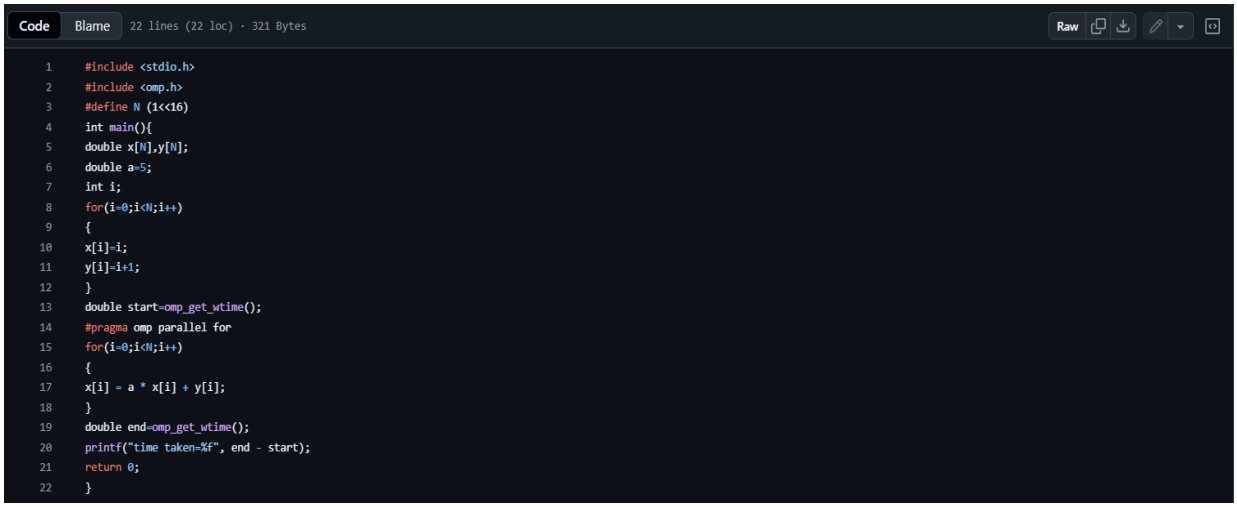
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB

INDIA

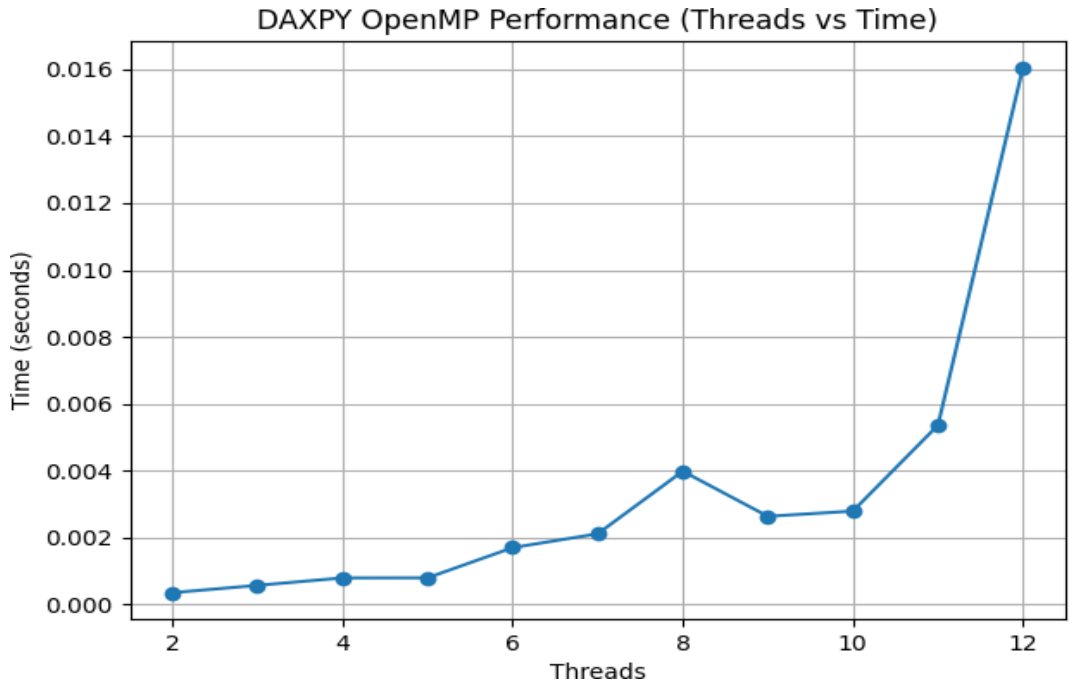
Session-Year (Jan-May, 2026)

Q1. DAXPY Loop Analysis:





```
root@Mannya-Soni: ~/lab1
root@Mannya-Soni:~/lab1# nano q1_daxpy_omp.c
root@Mannya-Soni:~/lab1# gcc q1_daxpy_omp.c    -fopenmp -O2 -o q1p
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=2 ./q1p
time taken=0.000341root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=3 ./q1p
time taken=0.000560root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=4 ./q1p
time taken=0.000783root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=5 ./q1p
time taken=0.000785root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=6 ./q1p
time taken=0.001689root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=7 ./q1p
time taken=0.002103root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=8 ./q1p
time taken=0.003977root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=9 ./q1p
time taken=0.002626root@Mannya-Soni:~/lab1#
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=10 ./q1p
time taken=0.002783root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=11 ./q1p
time taken=0.005353root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=12 ./q1p
time taken=0.016066root@Mannya-Soni:~/lab1#
```



Observation :

From the execution time graph, it is observed that increasing the number of threads does not significantly improve performance. In fact, after a small number of threads, the execution time increases. This happens because DAXPY is a memory-bound operation where threads compete for memory bandwidth and cache resources. Therefore, DAXPY shows poor scalability with OpenMP and performs best with fewer threads.

Q2. Matrix Multiplication Analysis:

root@Mannya-Soni: ~/lab1

GNU nano 7.2 q2_mm_serial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
    const int N = 1000;
    double *A = (double*)malloc(N*N*sizeof(double));
    double *B = (double*)malloc(N*N*sizeof(double));
    double *C = (double*)malloc(N*N*sizeof(double));
    if (!A || !B || !C) { printf("Malloc failed\n"); return 1; }

    for (int i = 0; i < N*N; i++) {
        A[i] = 1.0;
        B[i] = 1.0;
        C[i] = 0.0;
    }

    double start = omp_get_wtime();

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double sum = 0.0;
            for (int k = 0; k < N; k++) {
                sum += A[i*N + k] * B[k*N + j];
            }
            C[i*N + j] = sum;
        }
    }

    double end = omp_get_wtime();
    printf("Serial MM time: %f sec\n", end - start);

    free(A); free(B); free(C);
    return 0;
```

[Read 35 lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark M-] To Bracket
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy ^Q Where Was

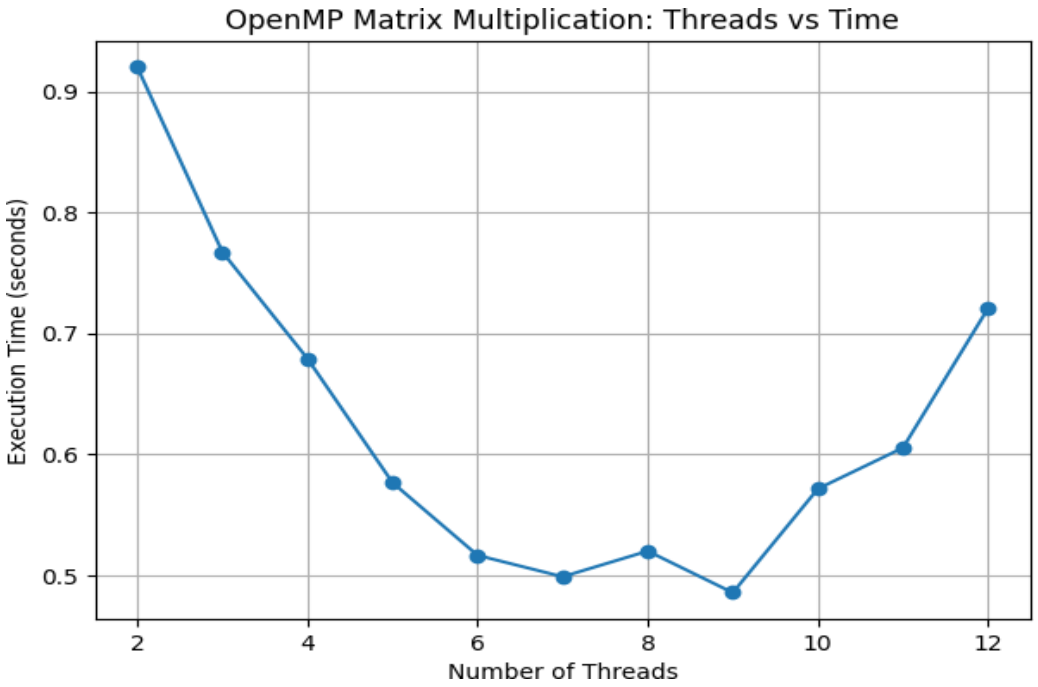
root@Mannya-Soni: ~/lab1

```
root@Mannya-Soni:~/lab1# nano q2_mm_serial.c
root@Mannya-Soni:~/lab1# ./mm_s
Serial MM time: 0.906190 sec
root@Mannya-Soni:~/lab1#
```

Code Blame 33 lines (31 loc) · 756 Bytes Raw

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <omp.h>
4   #define N 1000
5
6   int main() {
7       int i, j, k;
8       double *A, *B, *C;
9       A = (double*)malloc(N * N * sizeof(double));
10      B = (double*)malloc(N * N * sizeof(double));
11      C = (double*)malloc(N * N * sizeof(double));
12      for (i = 0; i < N*N; i++) {
13          A[i] = 2.0;
14          B[i] = 3.0;
15          C[i] = 0.0;
16      }
17      double start = omp_get_wtime();
18      #pragma omp parallel for
19      for (i = 0; i < N; i++) {
20          for (j = 0; j < N; j++) {
21              for (k = 0; k < N; k++) {
22                  C[i*N + j] += A[i*N + k] * B[k*N + j];
23              }
24          }
25      }
26      double end = omp_get_wtime();
27      printf("Time taken = %f", end - start);
28      free(A);
29      free(B);
30      free(C);
31      return 0;
32  }
```

```
root@Mannya-Soni:~/lab1# nano q2_mm_1d.c
root@Mannya-Soni:~/lab1# gcc q2_mm_1d.c     -fopenmp -O2 -o mm_1d
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=2 ./mm_1d
Time taken = 0.919969root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=3 ./mm_1d
Time taken = 0.767318root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=4 ./mm_1d
Time taken = 0.679285root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=5 ./mm_1d
Time taken = 0.576821root@Mannya-Soni:~/lab1# ^C
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=6 ./mm_1d
Time taken = 0.516749root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=7 ./mm_1d
Time taken = 0.499091root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=8 ./mm_1d
Time taken = 0.520290root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=9 ./mm_1d
Time taken = 0.485900root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=10 ./mm_1d
Time taken = 0.571730root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=11 ./mm_1d
Time taken = 0.605598root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=12 ./mm_1d
Time taken = 0.720786root@Mannya-Soni:~/lab1#
```

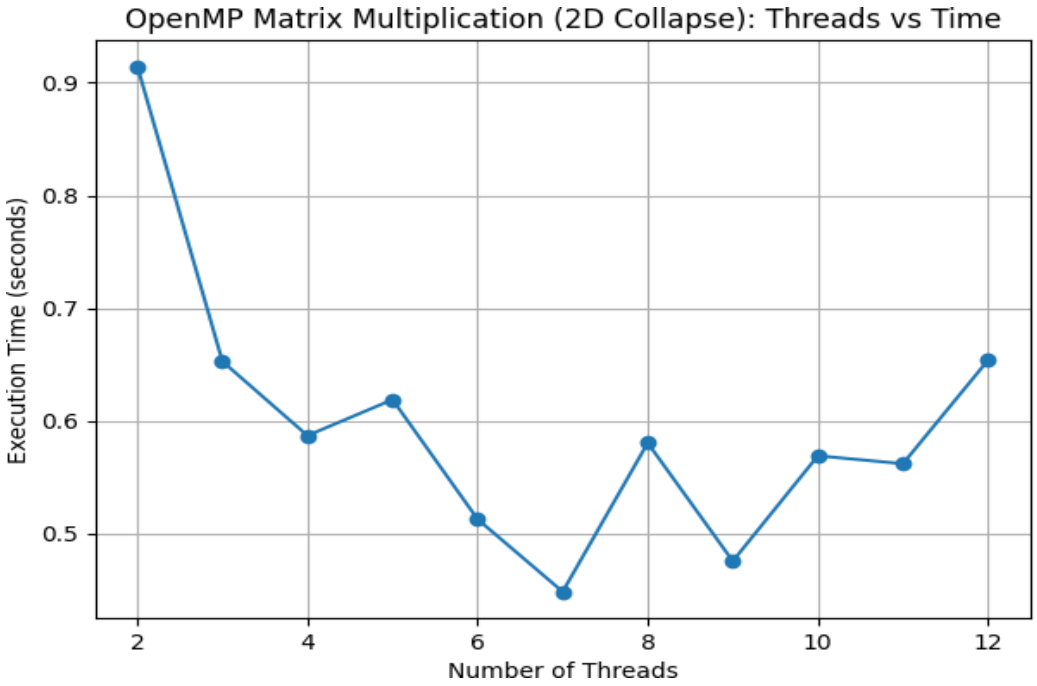


Code Blame 38 lines (37 loc) · 937 Bytes Raw

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 1000

int main() {
    int i, j, k;
    double *A, *B, *C;
    A = (double *)malloc(N * N * sizeof(double));
    B = (double *)malloc(N * N * sizeof(double));
    C = (double *)malloc(N * N * sizeof(double));
    if (A == NULL || B == NULL || C == NULL) {
        printf("allocation failed");
        return 1;
    }
    for (i = 0; i < N * N; i++) {
        A[i] = 2.0;
        B[i] = 4.0;
        C[i] = 0.0;
    }
    double start = omp_get_wtime();
    #pragma omp parallel for collapse(2)
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            double sum = 0.0;
            for (k = 0; k < N; k++) {
                sum += A[i*N + k] * B[k*N + j];
            }
            C[i*N + j] = sum;
        }
    }
    double end = omp_get_wtime();
    printf("2D Threading Time = %f", end - start);
    free(A);
    free(B);
    free(C);
    return 0;
}
```

```
root@Mannya-Soni:~/lab1# touch q2_mm_2d.c
root@Mannya-Soni:~/lab1# touch q2_mm_2d.c
root@Mannya-Soni:~/lab1# nano q2_mm_2d.c
root@Mannya-Soni:~/lab1# gcc q2_mm_2d.c     -fopenmp -O2 -o mm_2d
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=2 ./mm_2d
2D Threading Time = 0.914642root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=3 ./mm_2d
2D Threading Time = 0.652942root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=4 ./mm_2d
2D Threading Time = 0.587082root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=5 ./mm_2d
2D Threading Time = 0.618995root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=6 ./mm_2d
2D Threading Time = 0.512936root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=7 ./mm_2d
2D Threading Time = 0.448649root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=8 ./mm_2d
2D Threading Time = 0.580598root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=9 ./mm_2d
2D Threading Time = 0.475766root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=10 ./mm_2d
2D Threading Time = 0.569172root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=11 ./mm_2d
2D Threading Time = 0.562154root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=12 ./mm_2d
2D Threading Time = 0.653956root@Mannya-Soni:~/lab1#
```



Observation:

The results show a clear reduction in execution time as the number of threads increases, demonstrating good parallel scalability. Since matrix multiplication is compute-intensive, the workload is effectively distributed across threads, leading to significant speedup. The 2D collapse strategy performs better than 1D threading because it improves load balancing and reduces idle time among threads. However, after a certain thread, performance begins to degrade due to overheads.

Q3. π Calculation Analysis:

Problem Statement:

The value of π is computed using numerical integration:

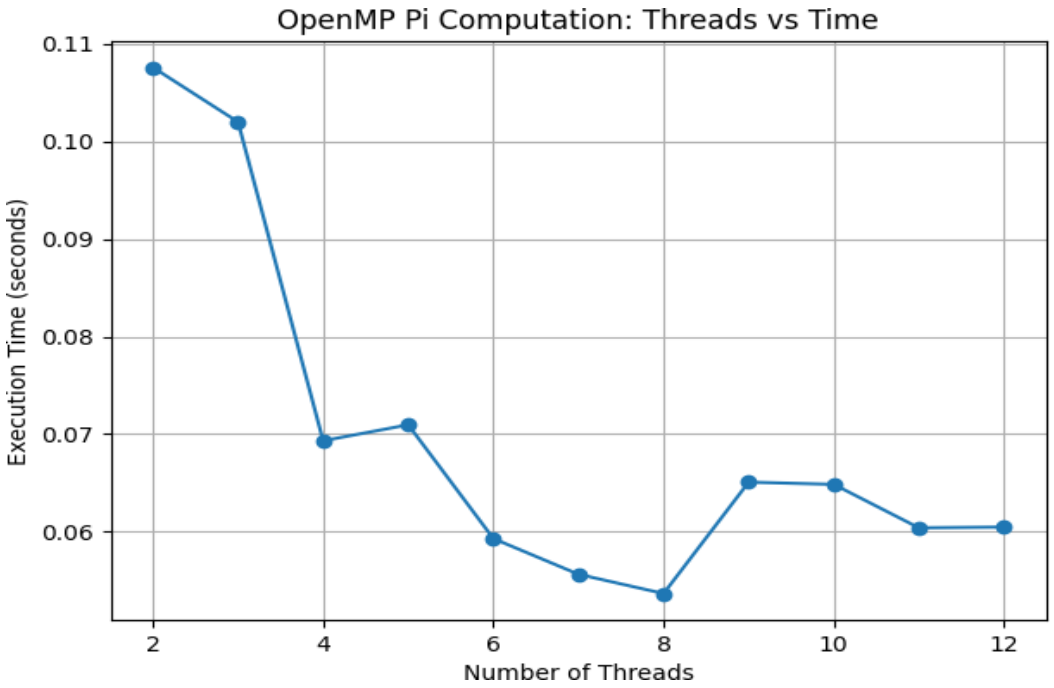
$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$

The integral is approximated using the rectangle method and parallelized using OpenMP reduction.

Code | Blame | 19 lines (19 loc) · 523 Bytes | Raw

```c
#include <stdio.h>
#include <omp.h>
#define STEPS 100000000
int main() {
    int i;
    double x, pi, sum = 0.0;
    double step = 1.0 / STEPS;
    double start = omp_get_wtime();
    #pragma omp parallel for private(x) reduction(+:sum)
    for (i = 0; i < STEPS; i++) {
        x = (i + 0.5) * step;
        sum += 4.0 / (1.0 + x * x);
    }
    pi = step * sum;
    double end = omp_get_wtime();
    printf("Calculated value of Pi = %.10f\n", pi);
    printf("Time taken = %f seconds\n", end - start);
    return 0;
}
```

```
root@Mannya-Soni:~/lab1# nano q3_pi_omp.c
root@Mannya-Soni:~/lab1# gcc q3_pi_omp.c      -fopenmp -O2 -o pi_p
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=2 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.107599 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=3 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.102003 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=4 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.069321 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=5 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.070958 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=6 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.059295 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=7 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.055633 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=8 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.053658 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=9 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.065079 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=10 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.064851 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=11 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.060395 seconds
root@Mannya-Soni:~/lab1# OMP_NUM_THREADS=12 ./pi_p
Calculated value of Pi = 3.1415926536
Time taken = 0.060477 seconds
root@Mannya-Soni:~/lab1#
```



Observation:

The parallel reduction approach significantly improves performance compared to the serial implementation. Execution time decreases steadily as the number of threads increases up to the number of available CPU cores. Beyond this point, the performance gain saturates and may slightly degrade due to synchronization overhead and thread management costs.