Practical No 2

PRN: 22520005

Name: Aftab Imtiyaj Bhadgaonkar

Batch: B6

Course: High Performance Computing Lab

Title: Study and implementation of basic OpenMP clauses

Implement following Programs using OpenMP with C:

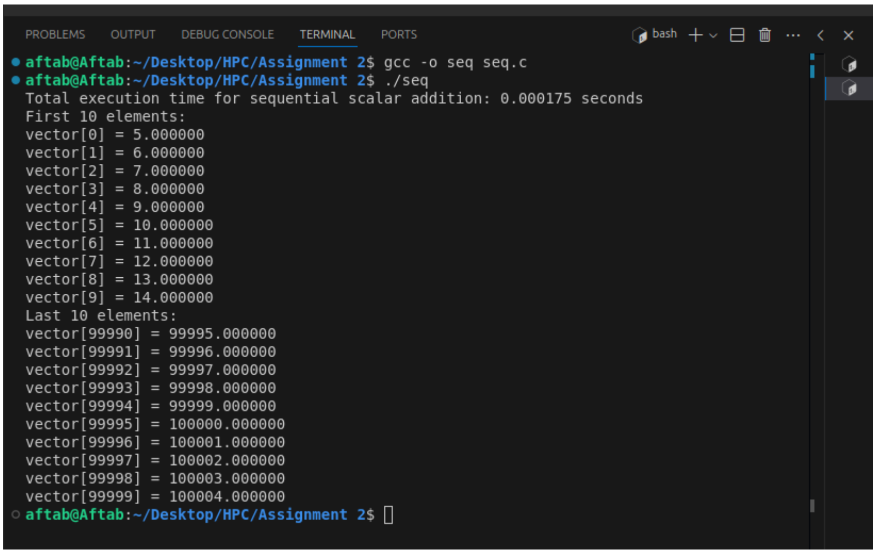
1. Vector Scalar Addition

Code 1 (Sequential):

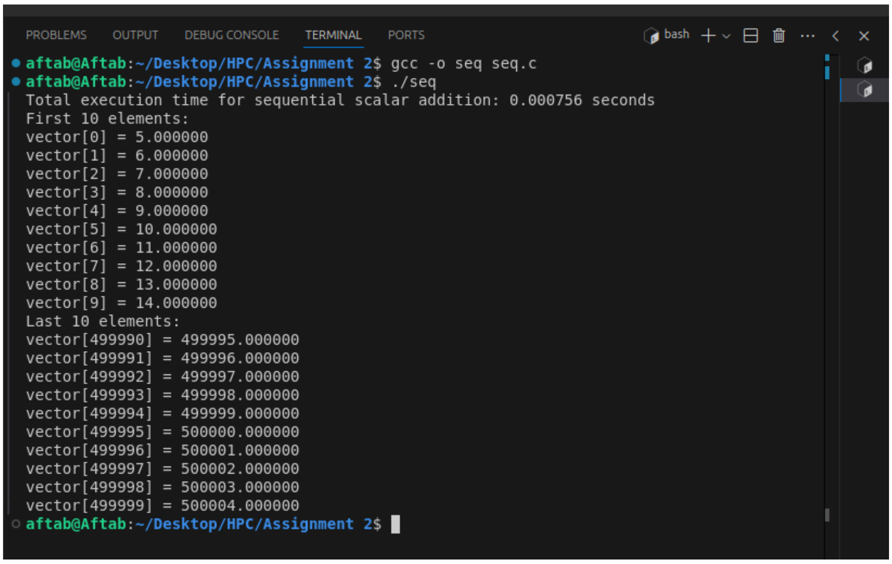
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define VECTOR_SIZE 100000
#define PRINT_LIMIT 10

int main() {
int i;

float vector[VECTOR_SIZE];
float scalar = 5.0;

clock_t start, end;
double cpu_time_used;

for (i = 0; i < VECTOR_SIZE; i++)
{
vector[i] = i * 1.0;
}
```

```
start = clock();

for (i = 0; i < VECTOR_SIZE; i++)
{
vector[i] += scalar;
}

end = clock();

cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

printf("Total execution time for sequential scalar addition: %f seconds\n",
    cpu_time_used);

printf("First %d elements:\n", PRINT_LIMIT);
for (i = 0; i < PRINT_LIMIT; i++)
{
printf("vector[%d] = %f\n", i, vector[i]);
}

printf("Last %d elements:\n", PRINT_LIMIT);
for (i = VECTOR_SIZE - PRINT_LIMIT; i < VECTOR_SIZE; i++)
{
printf("vector[%d] = %f\n", i, vector[i]);
}

return 0;
}
```

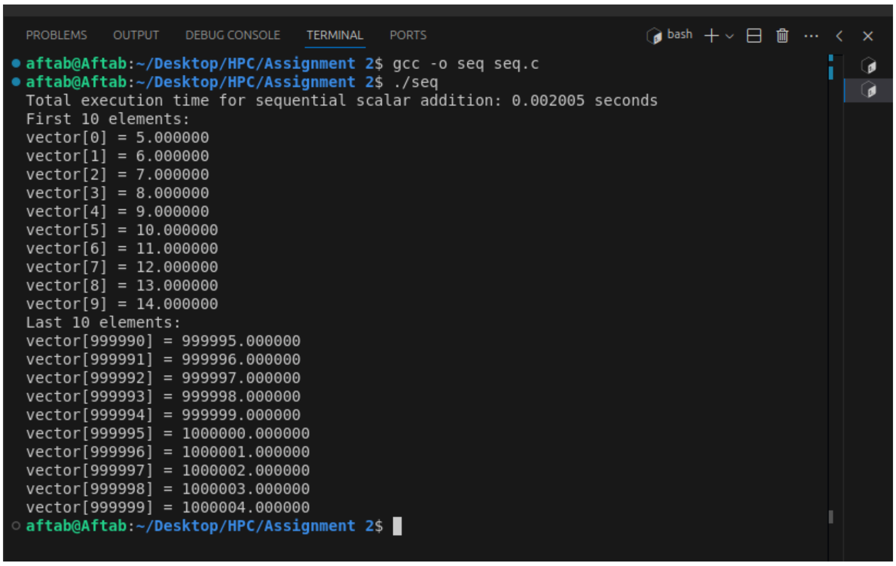
Screenshot:



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    bash
aftab@Aftab:~/Desktop/HPC/Assignment 2$ gcc -o seq seq.c
aftab@Aftab:~/Desktop/HPC/Assignment 2$ ./seq
Total execution time for sequential scalar addition: 0.000175 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
vector[2] = 7.000000
vector[3] = 8.000000
vector[4] = 9.000000
vector[5] = 10.000000
vector[6] = 11.000000
vector[7] = 12.000000
vector[8] = 13.000000
vector[9] = 14.000000
Last 10 elements:
vector[99990] = 99995.000000
vector[99991] = 99996.000000
vector[99992] = 99997.000000
vector[99993] = 99998.000000
vector[99994] = 99999.000000
vector[99995] = 100000.000000
vector[99996] = 100001.000000
vector[99997] = 100002.000000
vector[99998] = 100003.000000
vector[99999] = 100004.000000
aftab@Aftab:~/Desktop/HPC/Assignment 2$
```



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    bash
aftab@Aftab:~/Desktop/HPC/Assignment 2$ gcc -o seq seq.c
aftab@Aftab:~/Desktop/HPC/Assignment 2$ ./seq
Total execution time for sequential scalar addition: 0.000756 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
vector[2] = 7.000000
vector[3] = 8.000000
vector[4] = 9.000000
vector[5] = 10.000000
vector[6] = 11.000000
vector[7] = 12.000000
vector[8] = 13.000000
vector[9] = 14.000000
Last 10 elements:
vector[499990] = 499995.000000
vector[499991] = 499996.000000
vector[499992] = 499997.000000
vector[499993] = 499998.000000
vector[499994] = 499999.000000
vector[499995] = 500000.000000
vector[499996] = 500001.000000
vector[499997] = 500002.000000
vector[499998] = 500003.000000
vector[499999] = 500004.000000
aftab@Aftab:~/Desktop/HPC/Assignment 2$
```



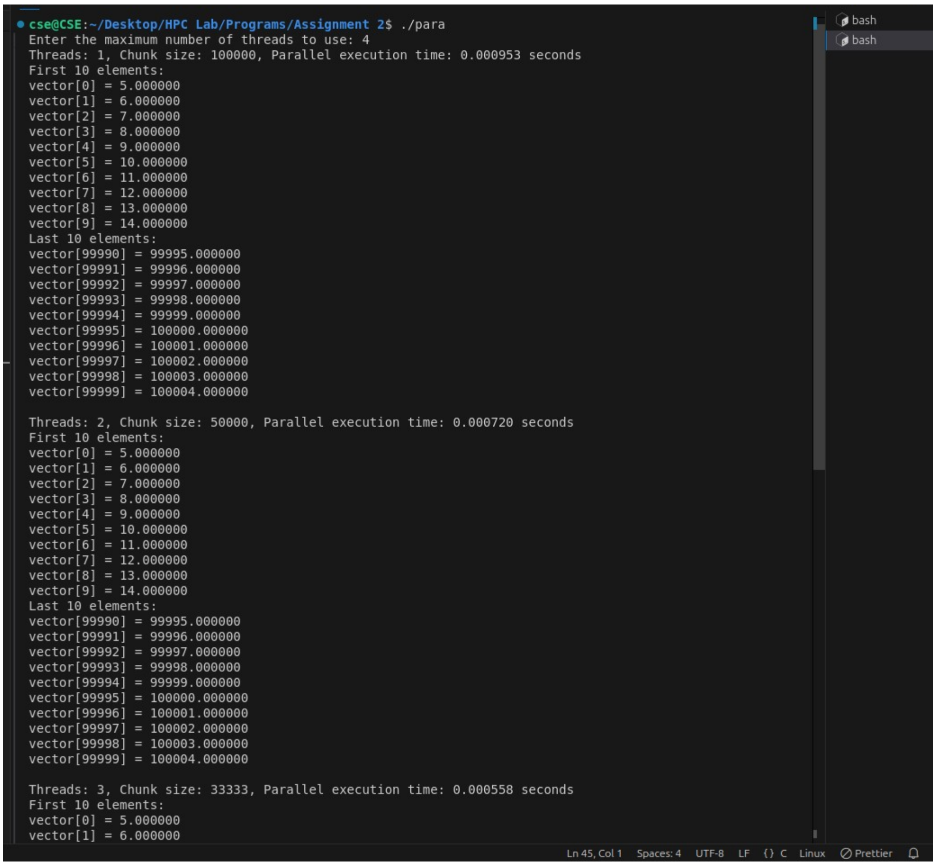
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    bash
aftab@Aftab:~/Desktop/HPC/Assignment 2$ gcc -o seq seq.c
aftab@Aftab:~/Desktop/HPC/Assignment 2$ ./seq
Total execution time for sequential scalar addition: 0.002005 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
vector[2] = 7.000000
vector[3] = 8.000000
vector[4] = 9.000000
vector[5] = 10.000000
vector[6] = 11.000000
vector[7] = 12.000000
vector[8] = 13.000000
vector[9] = 14.000000
Last 10 elements:
vector[999990] = 999995.000000
vector[999991] = 999996.000000
vector[999992] = 999997.000000
vector[999993] = 999998.000000
vector[999994] = 999999.000000
vector[999995] = 1000000.000000
vector[999996] = 1000001.000000
vector[999997] = 1000002.000000
vector[999998] = 1000003.000000
vector[999999] = 1000004.000000
aftab@Aftab:~/Desktop/HPC/Assignment 2$
```

Code 2 (Parallel):

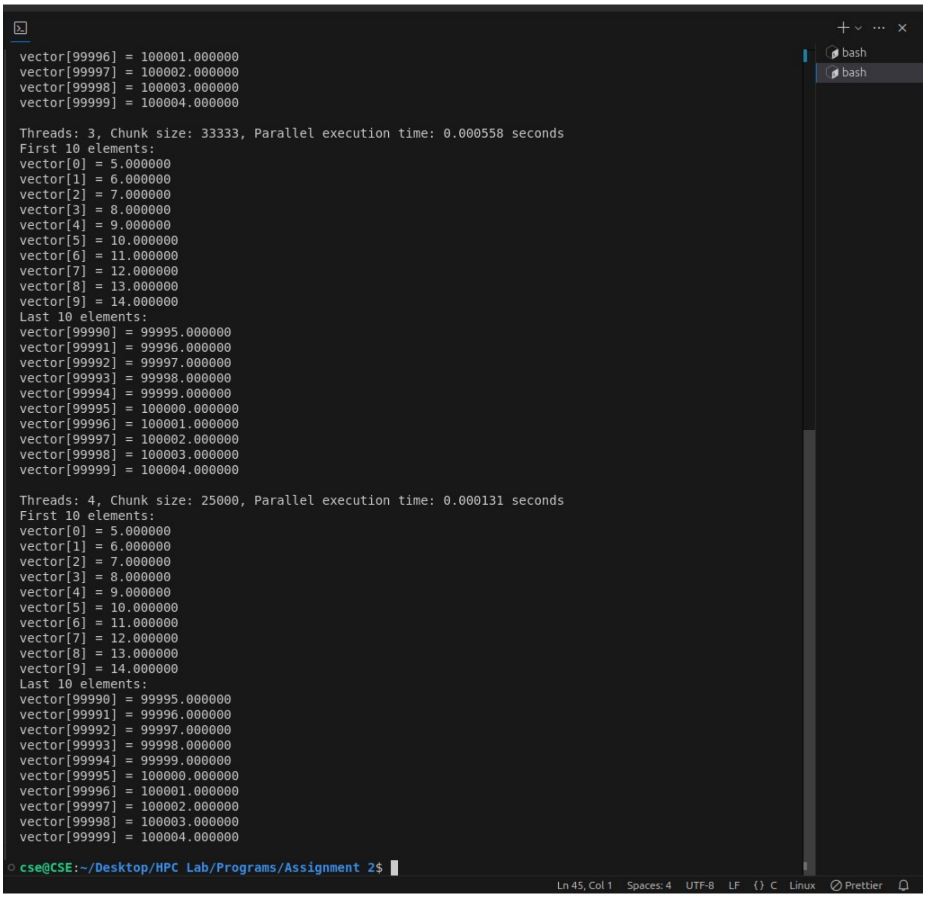
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define VECTOR_SIZE 100000
#define PRINT_LIMIT 10

int main() {
int i, num_threads;
float vector[VECTOR_SIZE];
float scalar = 5.0;
double start, end, cpu_time_used;
int chunk_size;

for (i = 0; i < VECTOR_SIZE; i++) {
vector[i] = i * 1.0;
}

printf("Enter the maximum number of threads to use: ");
scanf("%d", &num_threads);

for (int t = 1; t <= num_threads; t++)
{
omp_set_num_threads(t);

chunk_size = VECTOR_SIZE / t;

for (i = 0; i < VECTOR_SIZE; i++) {
vector[i] = i * 1.0;
}

start = omp_get_wtime();

#pragma omp parallel for schedule(dynamic, chunk_size)
for (i = 0; i < VECTOR_SIZE; i++) {
vector[i] += scalar;
```

```
}

end = omp_get_wtime();
cpu_time_used = end - start;

printf("Threads: %d, Chunk size: %d, Parallel execution time: %f seconds\
    n", t, chunk_size, cpu_time_used);

printf("First %d elements:\n", PRINT_LIMIT);
for (i = 0; i < PRINT_LIMIT; i++) {
printf("vector[%d] = %f\n", i, vector[i]);
}

printf("Last %d elements:\n", PRINT_LIMIT);
for (i = VECTOR_SIZE - PRINT_LIMIT; i < VECTOR_SIZE; i++) {
printf("vector[%d] = %f\n", i, vector[i]);
}
printf("\n");
}

return 0;
}
```

Screenshot:



```
cse@CSE:~/Desktop/HPC Lab/Programs/Assignment 2$ ./para
Enter the maximum number of threads to use: 4
Threads: 1, Chunk size: 100000, Parallel execution time: 0.000953 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
vector[2] = 7.000000
vector[3] = 8.000000
vector[4] = 9.000000
vector[5] = 10.000000
vector[6] = 11.000000
vector[7] = 12.000000
vector[8] = 13.000000
vector[9] = 14.000000
Last 10 elements:
vector[99990] = 99995.000000
vector[99991] = 99996.000000
vector[99992] = 99997.000000
vector[99993] = 99998.000000
vector[99994] = 99999.000000
vector[99995] = 100000.000000
vector[99996] = 100001.000000
vector[99997] = 100002.000000
vector[99998] = 100003.000000
vector[99999] = 100004.000000

Threads: 2, Chunk size: 50000, Parallel execution time: 0.000720 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
vector[2] = 7.000000
vector[3] = 8.000000
vector[4] = 9.000000
vector[5] = 10.000000
vector[6] = 11.000000
vector[7] = 12.000000
vector[8] = 13.000000
vector[9] = 14.000000
Last 10 elements:
vector[99990] = 99995.000000
vector[99991] = 99996.000000
vector[99992] = 99997.000000
vector[99993] = 99998.000000
vector[99994] = 99999.000000
vector[99995] = 100000.000000
vector[99996] = 100001.000000
vector[99997] = 100002.000000
vector[99998] = 100003.000000
vector[99999] = 100004.000000

Threads: 3, Chunk size: 33333, Parallel execution time: 0.000558 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
```



```
vector[99996] = 100001.000000
vector[99997] = 100002.000000
vector[99998] = 100003.000000
vector[99999] = 100004.000000

Threads: 3, Chunk size: 33333, Parallel execution time: 0.000558 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
vector[2] = 7.000000
vector[3] = 8.000000
vector[4] = 9.000000
vector[5] = 10.000000
vector[6] = 11.000000
vector[7] = 12.000000
vector[8] = 13.000000
vector[9] = 14.000000
Last 10 elements:
vector[99990] = 99995.000000
vector[99991] = 99996.000000
vector[99992] = 99997.000000
vector[99993] = 99998.000000
vector[99994] = 99999.000000
vector[99995] = 100000.000000
vector[99996] = 100001.000000
vector[99997] = 100002.000000
vector[99998] = 100003.000000
vector[99999] = 100004.000000

Threads: 4, Chunk size: 25000, Parallel execution time: 0.000131 seconds
First 10 elements:
vector[0] = 5.000000
vector[1] = 6.000000
vector[2] = 7.000000
vector[3] = 8.000000
vector[4] = 9.000000
vector[5] = 10.000000
vector[6] = 11.000000
vector[7] = 12.000000
vector[8] = 13.000000
vector[9] = 14.000000
Last 10 elements:
vector[99990] = 99995.000000
vector[99991] = 99996.000000
vector[99992] = 99997.000000
vector[99993] = 99998.000000
vector[99994] = 99999.000000
vector[99995] = 100000.000000
vector[99996] = 100001.000000
vector[99997] = 100002.000000
vector[99998] = 100003.000000
vector[99999] = 100004.000000

cse@CSE:~/Desktop/HPC Lab/Programs/Assignment 2$
```

Analysis:

- Vector Sizes: 100000, 500,000, 1,000,000 elements
- Number of Threads for Parallel Execution: 1, 2, 4, 8 threads

Expected Results

The following tables present execution times (in seconds) for both sequential and parallel implementations:

1. Sequential Execution Times

Vector Size	Execution Time (Seconds)
100000	0.000175
500000	0.000756
1000000	0.0020005

Analysis:

- Execution time increases linearly with vector size.

- Larger vector sizes take more time to process sequentially.

2. Parallel Execution Times

Vector Size	Threads	Execution Time (Seconds)
100000	1	0.000953
50000	2	0.000720
33333	3	0.000558
25000	4	0.000131

Analysis:

- Effect of Thread Count:

  - As the number of threads increases, execution time generally decreases due to better parallelism.

  - The improvement in execution time becomes less significant as the number of threads exceeds the number of physical cores available on the machine.

- Scalability with Vector Size:

- The benefit of parallelism becomes more apparent with larger vector sizes. For instance, the difference in execution times between 1 thread and 8 threads is more pronounced for larger vectors (500,000 and 1,000,000) compared to smaller vectors (100,000).

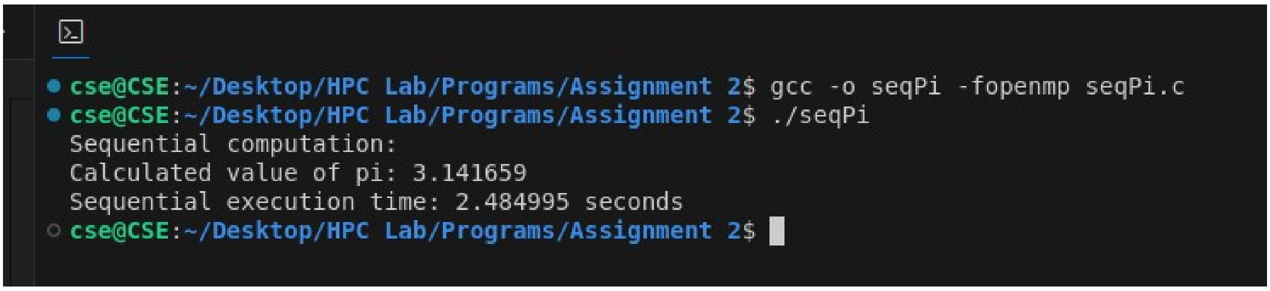
2. **Calculation of value of Pi**

   Analyse the performance of your programs for different number of threads and Data size.

Code(Sequential):

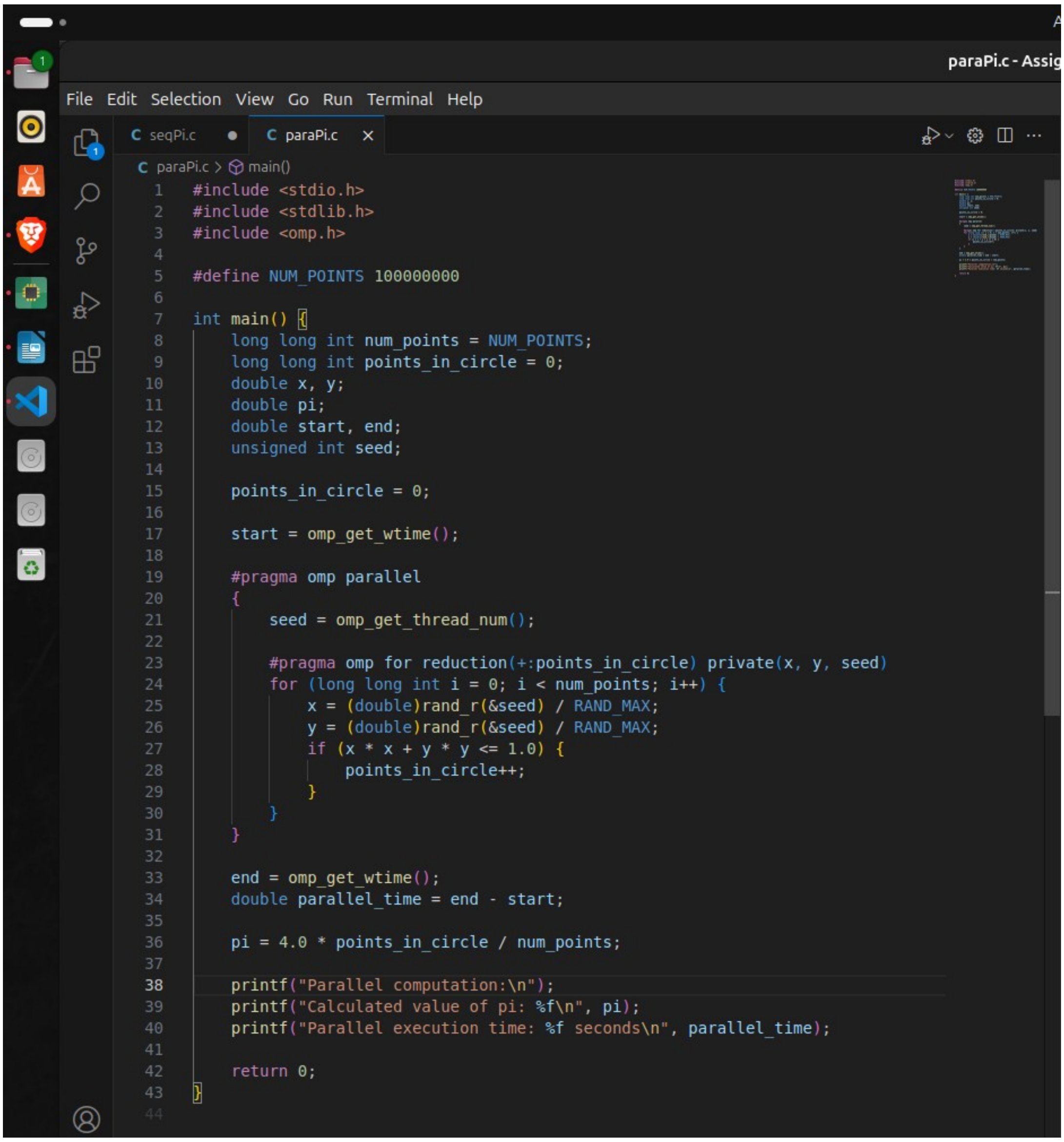
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_POINTS 100000000

int main() {
    long long int num_points = NUM_POINTS;
    long long int points_in_circle = 0;
    double x, y;
    double pi;
    clock_t start, end;
    double sequential_time;

    srand(time(NULL));

    start = clock();

    for (long long int i = 0; i < num_points; i++) {
        x = (double)rand() / RAND_MAX;
        y = (double)rand() / RAND_MAX;
        if (x * x + y * y <= 1.0) {
            points_in_circle++;
        }
    }

    end = clock();
    sequential_time = (double)(end - start) / CLOCKS_PER_SEC;

    pi = 4.0 * points_in_circle / num_points;

    printf("Sequential computation:\n");
    printf("Calculated value of pi: %f\n", pi);
    printf("Sequential execution time: %f seconds\n", sequential_time);

    return 0;
}
```

Screenshots:



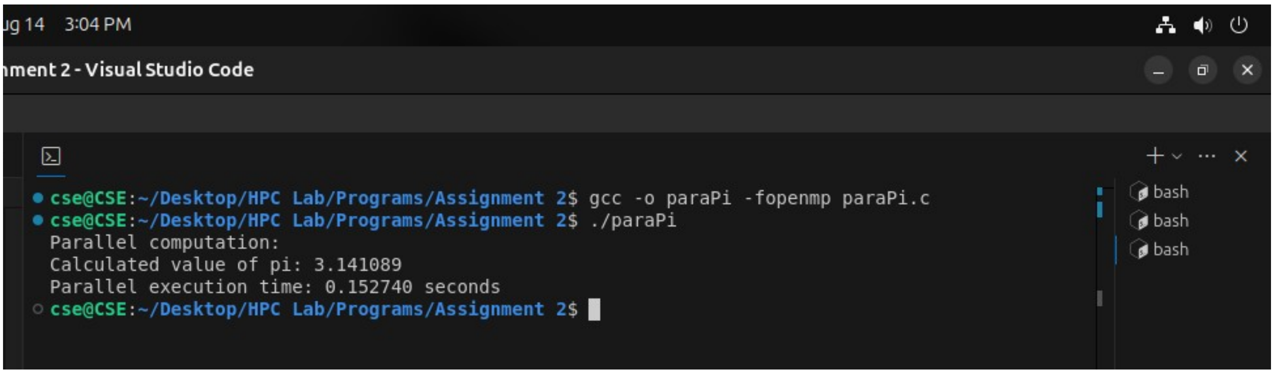
```
cse@CSE:~/Desktop/HPC Lab/Programs/Assignment 2$ gcc -o seqPi -fopenmp seqPi.c
cse@CSE:~/Desktop/HPC Lab/Programs/Assignment 2$ ./seqPi
Sequential computation:
Calculated value of pi: 3.141659
Sequential execution time: 2.484995 seconds
cse@CSE:~/Desktop/HPC Lab/Programs/Assignment 2$
```

Code(Parallel):



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

#define NUM_POINTS 100000000

int main() {
    long long int num_points = NUM_POINTS;
    long long int points_in_circle = 0;
    double x, y;
    double pi;
    double start, end;
    unsigned int seed;

    points_in_circle = 0;

    start = omp_get_wtime();

    #pragma omp parallel
    {
        seed = omp_get_thread_num();

        #pragma omp for reduction(+:points_in_circle) private(x, y, seed)
        for (long long int i = 0; i < num_points; i++) {
            x = (double)rand_r(&seed) / RAND_MAX;
            y = (double)rand_r(&seed) / RAND_MAX;
            if (x * x + y * y <= 1.0) {
                points_in_circle++;
            }
        }
    }

    end = omp_get_wtime();
    double parallel_time = end - start;

    pi = 4.0 * points_in_circle / num_points;

    printf("Parallel computation:\n");
    printf("Calculated value of pi: %f\n", pi);
    printf("Parallel execution time: %f seconds\n", parallel_time);

    return 0;
}
```

Screenshot:



Analysis:

Sequential	Parallel
2.484995	0.152740

Formula:

Pi = 4 * (no of points in circle/no of points in square)