Indian Institute of Information Technology Surat

Lab Report on
High Performance Computing (CS 602) Practical

Submitted by

[RAHUL KUMAR SINGH] (UI21CS44)

Course Faculty

Dr. Sachin D. Patil

Department of Computer Science and Engineering
Indian Institute of Information Technology Surat
Gujarat-394190, India

Jan-2024

Lab No: 3

Aim: Write a program that generates np random points using any random function, computes the ratio q that approximates the probability P, and uses it to estimate the value of π.

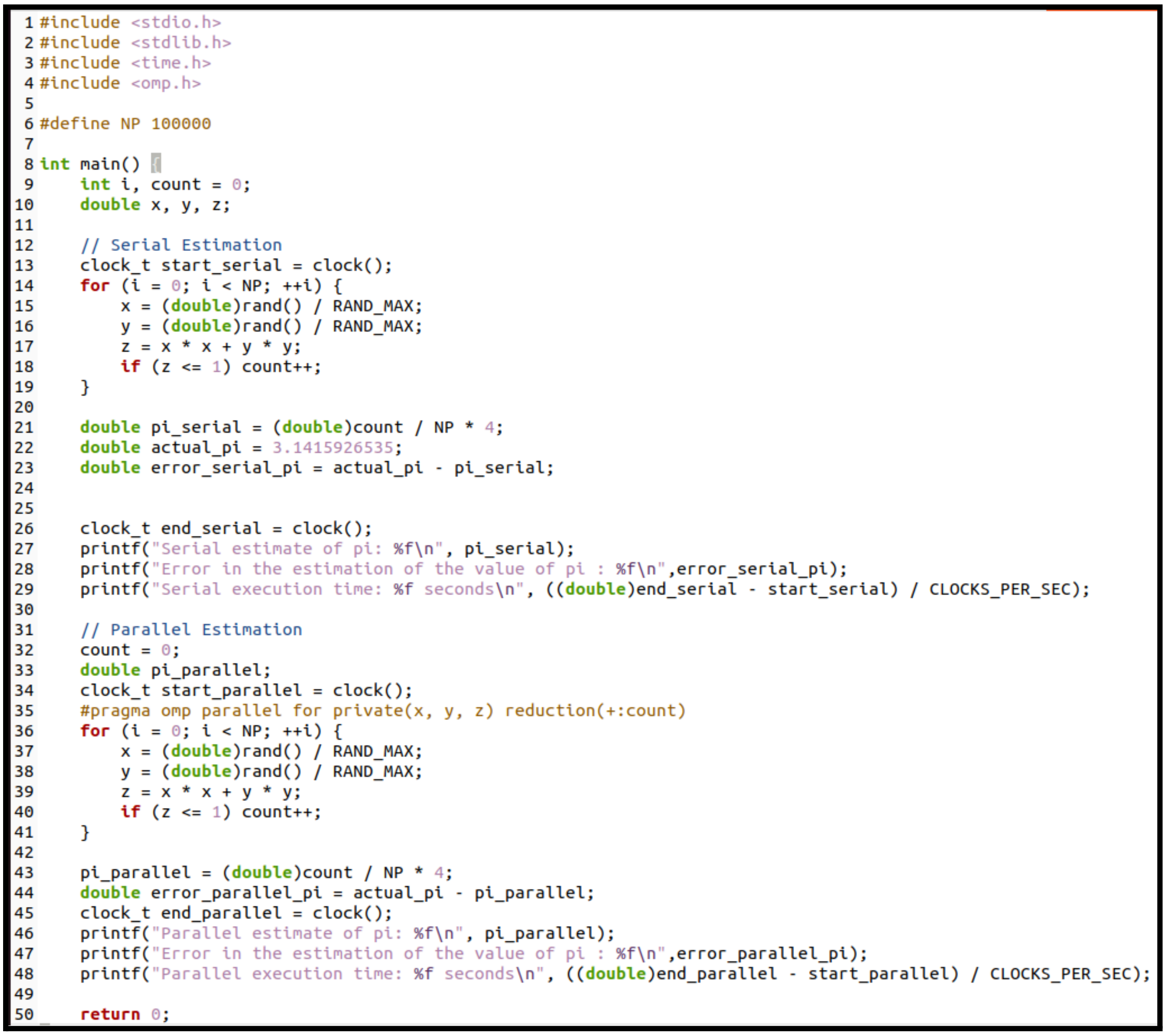
Description: In this task, we will estimate the value of π using the Monte Carlo method. Given a probability P, the Monte Carlo method relies on the generation of random samples (events) to compute a numerical approximation of P.

Consider a circle inscribed in a square. The method simply consists in generating random points (x, y) within the square range. Given the probability that the points lie within the circle, and the actual number of generated random points that do so, we can estimate π.

First, what is the probability of random points ending up within the area of the circle? The answer is to find the relationship between the geometry of the square and the circle. The area of the square is 4*R^2 and the area of the circle is πR^2. Now, the ratio of the area of the circle over the area of the square is π/4. That is, the probability that a random point within the square lies also within the circle is P =π/4, and thus π = 4*P.

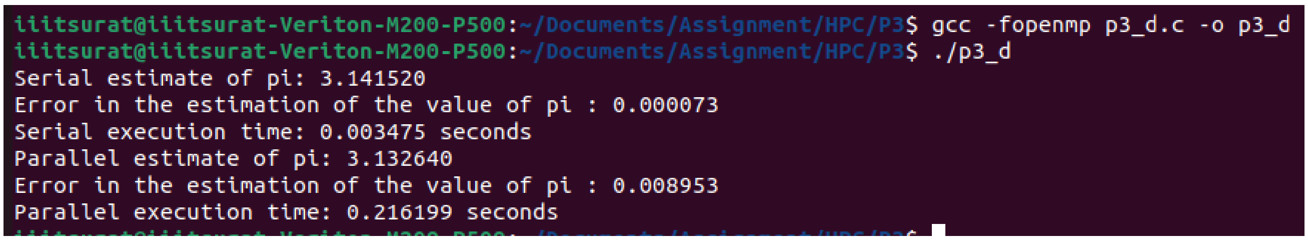
In this task, we will generate np (1,00, 000) random points using any random function, compute the ratio q that approximates the probability P, and use it to estimate the value of π while maintaining parallel OpenMP construct.

Source Code:



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

#define NP 100000

int main() {
    int i, count = 0;
    double x, y, z;

    // Serial Estimation
    clock_t start_serial = clock();
    for (i = 0; i < NP; ++i) {
        x = (double)rand() / RAND_MAX;
        y = (double)rand() / RAND_MAX;
        z = x * x + y * y;
        if (z <= 1) count++;
    }

    double pi_serial = (double)count / NP * 4;
    double actual_pi = 3.1415926535;
    double error_serial_pi = actual_pi - pi_serial;


    clock_t end_serial = clock();
    printf("Serial estimate of pi: %f\n", pi_serial);
    printf("Error in the estimation of the value of pi : %f\n",error_serial_pi);
    printf("Serial execution time: %f seconds\n", ((double)end_serial - start_serial) / CLOCKS_PER_SEC);

    // Parallel Estimation
    count = 0;
    double pi_parallel;
    clock_t start_parallel = clock();
    #pragma omp parallel for private(x, y, z) reduction(+:count)
    for (i = 0; i < NP; ++i) {
        x = (double)rand() / RAND_MAX;
        y = (double)rand() / RAND_MAX;
        z = x * x + y * y;
        if (z <= 1) count++;
    }

    pi_parallel = (double)count / NP * 4;
    double error_parallel_pi = actual_pi - pi_parallel;
    clock_t end_parallel = clock();
    printf("Parallel estimate of pi: %f\n", pi_parallel);
    printf("Error in the estimation of the value of pi : %f\n",error_parallel_pi);
    printf("Parallel execution time: %f seconds\n", ((double)end_parallel - start_parallel) / CLOCKS_PER_SEC);

    return 0;
```

Output:



```
iiitsurat@iiitsurat-Veriton-M200-P500:~/Documents/Assignment/HPC/P3$ gcc -fopenmp p3_d.c -o p3_d
iiitsurat@iiitsurat-Veriton-M200-P500:~/Documents/Assignment/HPC/P3$ ./p3_d
Serial estimate of pi: 3.141520
Error in the estimation of the value of pi : 0.000073
Serial execution time: 0.003475 seconds
Parallel estimate of pi: 3.132640
Error in the estimation of the value of pi : 0.008953
Parallel execution time: 0.216199 seconds
```

Conclusion:

- It can be observed how close the estimation is to the known value, acknowledging the stochastic nature of the method.
- Careful consideration was given to shared and private variables. Variables like count were marked for reduction to ensure thread safety, while x, y and z were designated as private to prevent data races.
- The program was parallelized using OpenMP to expedite the generation of random points.
- The Monte Carlo method is a probabilistic numerical technique that uses random sampling to estimate mathematical values.