

Concurrency and Parallelism 2020-21

Calculation of π Using the Monte Carlo Method

João M. Lourenço

March 17, 2021

Abstract

In this class, you will learn/remember some basic concepts of concurrency and how to process data in parallel.

1 Introduction

The “Monte Carlo Method” is a method of solving problems using statistics. Given the probability, p , that an event will occur in certain conditions, a computer can be used to generate those conditions repeatedly. The number of times the event occurs divided by the number of times the conditions are generated should be approximately equal to p .

Figure 1 shows a circle with radius $r = 1$ inscribed within a square. The area of the circle is $\pi r^2 = \pi 1^2 = \pi$, and the area of the square is $(2r)^2 = 2^2 = 4$. The ratio of the area of the circle to the area of the square is $\rho = \frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi}{4} = 0.7853981634$. Given ρ then $\pi = \rho \times 4$.

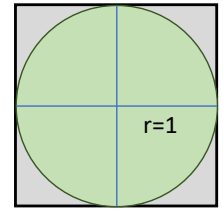


Figure 1: A circle within a square.

If we could compute ratio ρ , then we could multiple it by four to obtain the value π . One particularly simple way to compute ρ is to pick random points in the square and count how many of them lie inside the circle (see Figure2).

Suppose for example that the points $\left\{ \frac{2i-1}{32}, \frac{2j-1}{32} \right\}_{i=1, j=1}^{32, 32}$ are selected, then there are 812 points inside the circle and 212 points outside the circle. The percentage of points inside the circle is $\rho = \frac{812}{812+212} = \frac{812}{1024} = 0.792195122$. Then the area of the circle is approximated with the following calculation: $\pi = \text{Area of circle} = \rho \times \text{Area of square} = \rho \times 4 = 0.792195122 \times 4 = 3,168780488$.

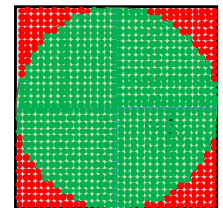


Figure 2: Ratio between areas of circle and square.

2 Monte Carlo Method for π

Monte Carlo methods can be thought of as statistical simulation methods that utilize a sequence of random numbers to perform the simulation. The name “Monte Carlo” was coined by Nicholas Constantine Metropolis (1915-1999) and inspired by Stanislaw Ulam (1909-1986), because of the similarity of statistical simulation to games of chance, and because Monte Carlo is a center for gambling and games of chance. In a typical process, one computes the number of points in a set A that lies inside box R . The ratio of the number of points that fall inside A to the total number of points tried is equal to the ratio of the two areas (or volume in 3 dimensions). The accuracy of the ratio p depends on the number of points used, with more points leading to a more accurate value.

A simple Monte Carlo simulation to approximate the value of π could involve randomly selecting points $(x_i, y_i)_{i=1}^n$ in the unit square and determining the ratio $\rho = \frac{m}{n}$ where m is number of points that satisfy $x_i^2 + y_i^2 \leq 1$. In a typical simulation of sample size $n = 1000$ there are 787 points satisfying that equation. Using this data we obtain $\rho = \frac{m}{n} = \frac{787}{1000} = 0.787$ and $\pi = p \times 4 = 0.787 \times 4 = 3.148$.

Every time a Monte Carlo simulation is made using the same sample size n , it will come up with a slightly different value. The values converge very slowly of the order $O(n^{-1/2})$. This property is a consequence of the Central Limit Theorem.

You may find a web simulation of this method at <https://academo.org/demos/estimating-pi-monte-carlo/>.

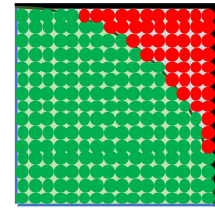


Figure 3: Monte Carlo method.

3 Lab Work

3.1 Sequential Version

Design and implement a Java program named `approxPi` that approximates the value of π by using the Monte Carlo method. The program must receive a command-line argument that specifies the number of simulations to be executed (i.e., the number of points to be generated) and provide an output as given in the example below. Try multiple values for the number of simulations, e.g.,

```
$ approxPi 1000
Total Number of points: 1000
Points within circle: 779
Pi estimation: 3.11600

$ approxPi 100000
Total Number of points: 100000
Points within the circle: 78656
Pi estimation: 3.14624
```

You may have an estimation of the time your program takes to run the full simulation by using the command `time`, e.g.,

```
$ time approxPi 1000
Total Number of points: 1000
Points within the circle: 779
Pi estimation: 3.11600
```

```
real 0m0.007s
user 0m0.001s
sys 0m0.003s
```

Use this simple lab exercise to learn how to use GIT. Create a repository in a free public server (e.g., github, gitlab, bitbucket, ...) and use it to manage the versioning of your code.

3.2 Parallel Version

Now, create a branch named “*paralell*” and develop a parallel version using Java-threads. This parallel version must accept a second argument indicating how many parallel threads will be executing. If the second argument is omitted, it defaults to one.

Remember to keep on using GIT. :)

3.3 To Think About...

The sequential version is faster, slower, or identical to the parallel version with just one thread?

The parallel version with two threads is faster than with one? When you double the number of threads does it take half the time? More? Less?

Final Note

If you search the web, it will be trivial to find an implementation of the Monte Carlo simulation to approximate the value of π , which you can easily adapt to your needs. But note that if you do that, you are cheating and you will not learn what you are supposed to learn in this lab class. Just be honest with yourself and make your own program. ;)

Acknowledgments

The text from the first two sections is an adaptation from the text in <http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html>.