FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Concurrency and Parallelism 2024-25

Calculing “π” Using the Monte Carlo Method

João M. Lourenço

March 5, 2025

Abstract

In this week, you will learn/remember some basic concepts about git and about concurrency, and how to process data in parallel using the Java programming language and Java threads.

1 Git & GitHub Fundamentals

If you are not familiar woth Git, please spend some time on the Git documentation, where you can find the online reference manual, the Pro Git book, and also some video tutorials.

2 Monte Carlo Method for π

Monte Carlo methods [1] 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 ρ depends on the number of points used, with more points leading to a more accurate value.

Figure 1 shows a circle with radius $r = 1$ inscribed within a square. The area of the circle is

$$\mathrm{A}_{\bigcirc} = \pi \cdot r^2 = \pi \cdot 1^2 = \pi$$

and the area of the square is

$$\mathrm{A}_{\square} = (2 \cdot r)^2 = 2^2 = 4$$

The ratio ρ of the area of the circle to the area of the square is

$$\rho = \frac{\mathrm{A}_{\bigcirc}}{\mathrm{A}_{\square}} = \frac{\pi}{4} = 0.7853981634$$

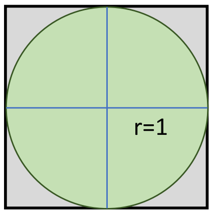

Figure 1: A circle within a square.

A simple Monte Carlo simulation to approximate the value of π must first estimate ρ. Once ρ is estimated, then

$$\pi = \rho \times 4 = 0.7853981634 \times 4 = 3.1415926536$$

.

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 Figure 2).

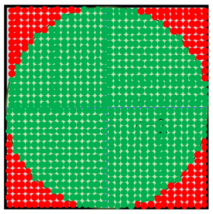
Figure 2: Ratio between areas of circle and square.

A Monte Carlo simulation to approximate the value of π will then involve randomly selecting points $(x_i, y_i)_{i=1}^{n}$ in the unit square and determining the ratio $\rho = \frac{m}{n}$, where n is the total number of points and m is number of points that satisfy $x_i^2 + y_i^2 \leq 1$ (see Figure 3).

In a simulation of sample size $n = 1000$, there were 787 points satisfying that equation. Using this data we obtain

$$\rho = \frac{m}{n} = \frac{787}{1000} = 0.787$$

and

$$\pi = \mathrm{A}_{\bigcirc} = \rho \times \mathrm{A}_{\square} = \rho \times 4 = 0.787 \times 4 = 3.148$$

.

If the numbers used in the Monte Carlo simulation are radom, every time a 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 [2].

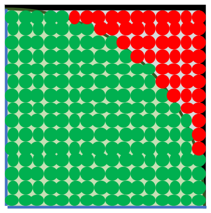
Figure 3: Monte Carlo method.

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

3 Lab Work

Open the following link, and log on to GitHub if requested...

`https://classroom.github.com/a/gcUXSI9t`

and then follow the instructions below.

In both the *sequential* and the *parallel* versions (see below), you are expected to implement the class `Simulate`.

3.1 Sequential Version

Design and implement a Java (sequential) program named `approxPiSeq` 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.,

```
$ approxPiSeq 1000
totalSteps      time (µsec)     Pi (estimation)
      1000      0.002389        3.072000000

$ approxPiSeq 10000
totalSteps      time (µsec)     Pi (estimation)
     10000      0.003012        3.149200000

$ java ApproxPiSeq 10000000
totalSteps      time (µsec)     Pi (estimation)
  10000000      0.559076        3.141480400
```

Remember to use this simple lab exercise to learn how to use GIT. Use the given repository (and the give starting code) to manage the versioning of your code. Remember to do frequent commits with clear commit messages.

Once your sequential version is running all right, try your program with some values (e.g., 100, 1000, 10000, etc) and build a plot with the execution time (t) as a function of n.

3.2 Parallel Version

Switch to the "Par" folder and, based on the code you already have for the squential version, develop a parallel version using Java-threads. This parallel version accepts a second (optional) argument indicating how many parallel threads will be executing. If the second argument is omitted, it defaults to one (see the given starting code).

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

There is plenty of bibliography about this problem, the corresponding Monte Carlo simulation, and its implementation in many different programming languages. Please understand that the objetive of this homework is for you to learn about concurrency (and concurrency in the Java programmin language). So, it makes no sense to cheat and search the web for working solutions of this problem! *Just be honest with yourself and develop your own solution, or just ignore it!*

Acknowledgments

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

References

[1] A. M. Johansen. “Monte Carlo Methods”. In: *International Encyclopedia of Education (Third Edition)*. Ed. by Penelope Peterson, Eva Baker, and Barry McGaw. Third Edition. Oxford: Elsevier, 2010, pp. 296–303. ISBN: 978-0-08-044894-7. DOI: `10.1016/B978-0-08-044894-7.01543-8`.

[2] S. L. Zabell. “Alan Turing and the Central Limit Theorem”. In: *The American Mathematical Monthly* 102.6 (1995), pp. 483–494. ISSN: 00029890, 19300972. URL: `http://www.jstor.org/stable/2974762`.