NOVA SCHOOL OF SCIENCE & TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

Concurrency and Parallelism 2022-23
Lab 1: Calculation of π Using the Monte Carlo Method

Hervé Paulino & Alex Davidson

March 2023

Abstract

In this week's lab, you will learn/recall some basic concepts about git and concurrency. The main goal will be to develop an example application that processes data in parallel using the Java programming language and Java threads.

1 Somewhat optional: Git & GitHub Fundamentals

The `git` command line tool allows you to build repositories of files, with clear versioning procedures and documented sequencing of changes. GitHub is a website that allows you to upload git repositories, and display in a way that is helpful for others to read, and give feedback on.

Using git and GitHub during this project is not mandatory, but is **highly** recommended. Using such tools on a daily basis is a basic requirement for most engineering roles, and even in academic contexts. Using these tools will help you to structure your assignment solutions (and eventually your project) in a way that others can understand the way that you came up with your eventual solution.

1.1 Learning to use git and GitHub

Open the following link, and log on (or register) to GitHub as requested

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

and then follow the tutorial on git and GitHub.

Not all of the steps are required: just make sure that you understand how the basic commands operate (i.e. `add`, `commit`, `push`, and `pull`). These are the commands that you will need to use most frequently.

2 Monte Carlo Methods

Monte Carlo methods [1] can be thought of as statistical simulation methods that utilize a sequence of random numbers to perform some kind of 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 centre for gambling and games of chance. In a typical two-dimensional process, one computes the number of points within a shape A that lies inside a larger shape 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 a three-dimensional process). The accuracy of the ratio ρ depends on the number of points used, with more points leading to a more accurate value.

2.1 Estimating the value of π

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

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

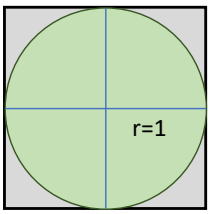

Figure 1: A circle within a square.

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

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

A simple Monte Carlo simulation to approximate the value of π can first estimate ρ and then π. Given ρ 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 Figure 2).

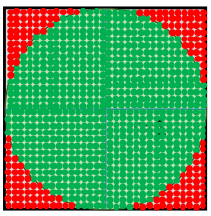

Figure 2: Ratio between areas of circle and square.

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 random, every time a simulation is made using the same sample size n, it will arrive at a slightly different value. The values converge very slowly with order $O(n^{-1/2})$. This property is a consequence of the Central Limit Theorem [2].

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

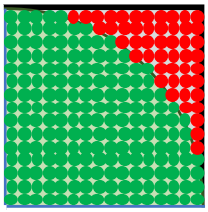

Figure 3: Monte Carlo method.

3 Lab Work

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

`https://classroom.github.com/a/e0il6wlj`

This should create a repository containing a Java codebase that uses the gradle framework for building and testing the code. To run/debug the code, you should be able to load it into your favourite editor (i.e. VS Code/Eclipse/etc...).

The `src/main/java/seq` folder contains a file that constructs a basic Java main class that you will use to build a simulation of the Monte Carlo method using standard sequential programming techniques.

The `src/main/java/par` folder contains a file that constructs a basic Java main class that you will eventually use to build a simulation of the Monte Carlo method using parallel programming technologies.

In both versions (*sequential* and *parallel*, see below), you are expected to implement the method `pi4()`.

3.1 Sequential Version

Using the code found in the `src/main/java/seq` folder of the git repository, complete the implementationof the `ApproxPiSeq` class 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.,

```
# n = 1000
totalSteps      time (tsec)     Pi (estimation)
      1000      0.002389        3.072000000

# n = 10000
totalSteps      time (tsec)     Pi (estimation)
     10000      0.003012        3.149200000

# n = 100000
totalSteps      time (tsec)     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 given starting code) to manage the versioning of your code. Remember to do **frequent** commits with **clear** commit messages.

Once your sequential version is running correctly, 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 `src/main/java/par` folder and, based on the code you already have for the sequential 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** using git: frequently, and with informative commit messages.

3.3 Some questions to think about

1. Is the sequential version faster, slower, or identical to the parallel version, when only using one thread?
2. Is the parallel version with two threads faster than with one? When you double the number of threads does it take half the time? More? Less?

Final Note

There is a large bibliography of work on this problem, the corresponding Monte Carlo simulation, and its implementation in a variety of programming languages. Please understand that the objective of this homework is for you to learn about concurrency (and concurrency in the Java programming language). In particular, this lab assignment is **not** graded. So, there is no value in cheating by searching 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 of the text found at `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.