departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Concurrency and Parallelism 2019-20
Calculation of π in C + OpenMP using the Monte Carlo Method

João Lourenço

March 17, 2020

Abstract

In this class you will learn/remember some basic concepts of concurrency and how to process data in parallel using the C programming language.

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.

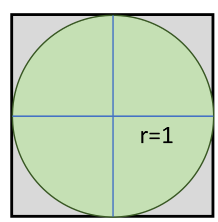

Figure 1: A circle within a square.

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 $p = \frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi}{4} = 0.7853981634$

If we could compute ratio p, then we could multiple it by four to obtain the value $\pi = p \times 4$. One particularly simple way to do this is to pick lattice 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 and the percentage of points inside the circle is $p = \frac{812}{812+212} = \frac{812}{1024} = 0.792195122$. Then the area of the circle is approximated with the following calculation: Area of circle $= p \times$ Area of square $= p \times 4 = 0.792195122 \times 4 = 3,168780488$.

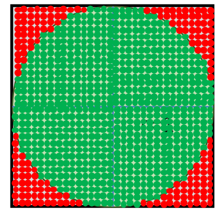

Figure 2: Ratio between areas of circle and square.

2 Monte Carlo Method for π

Monte Carlo methods can be thought of as statistical simulation methods that utilize a sequences of random numbers to perform the simulation. The name "Monte Carlo" was coined by Nicholas Constantine Metropolis (1915-1999) and inspired by Stanslaw 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 compute 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 $p = \frac{m}{n}$ 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 $p = \frac{m}{n} = \frac{787}{1000} = 0.787$ and $\pi = p \times 4 = 0.787 \times 4 = 3.148$.

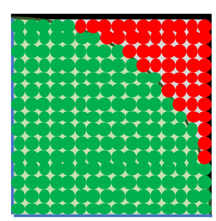

Figure 3: Monte Carlo method.

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/`.

3 Lab Work

3.1 Study OpenMP

Start by studying OpenMP (`https://www.openmp.org/resources/tutorials-articles/`). I recommend you to study these slides

`https://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf`

up to page 20 and and try the proposed exercises (up to page 20).

You may also follow this tutorial in youtube

`https://www.youtube.com/playlist?list=PLLX-Q6B8xqZ8n8bwjGdzBJ25X2utwnoEG`

and/or study this book

`https://libgen.lc/ads.php?md5=D27D7EE55FA748118FEA185BC27D6FB5`

Then step into the next section of this lab protocol.

3.2 OpenMP Parallel Version

Depart from the sequential version that you developed for the Lab of last week (P2). Now, use OpenMP to parallelize your program developing two versions:

1. An OpenMP parallel version that has as few changes to the sequential version as possible; and
2. An OpenMP parallel version that is optimized.

The parallel program(s) must receive as command line arguments the number of simulations to be executed (i.e., the number of points to be generated) and how many parallel threads shall be executing. The second command line argument shall be optional and default to one.

Remember to keep on using GIT. :)

3.3 Experimental evaluation

Add a flag to your command line arguments that will make your program output these plain values separated by tabs: `n_iterations`, `n_threads`, `run_time`. Now, run your program with different values for iterations and processors. Also, remember to run each configuration at least three times. You may use the script below to help you run the tests:

```
ITEREATIONS = 1000 10000 100000 1000000 10000000 100000000
THREADS = 1 2 4 8 16
NRUNS = 5
for IT in $ITEREATIONS; do
  for TH in $THREADS; do
     for R in `seq 1 $NRUNS`; do
        approxPi $ITEREATIONS $THREADS
     done
   done
done | tee output_file.csv
```

name it

```
file_name.sh
```

and run it with

```
bash file_name.sh
```

3.4 To Think About...

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

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

Which of your implementations is faster? Java? C + pThreads? C + OpenMP? Why?

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, that 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 to 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`.