Parallel Scientific Computing, ME 471, Boise State University
March 2022

Parallelization of the Monte Carlo Method to Find Pi

Gavin Wale
Boise State University
BS Mechanical Engineering
https://github.com/gavinwale

ABSTRACT

The Monte Carlo method is a statistical approach to approximating integrals through repeated random sampling. Pi, π, is the ratio of the circumference of any circle to the diameter of that circle regardless of size. The irrational number is well defined and well known, which makes it the ideal value to test the power of parallel computation. In this academic project, a simple algorithm was programmed and scaled to account for up to 2^{28} run tests and up to 64 processors on Boise State University's R2 compute cluster. Strong and weak experiments were executed, testing the scalability of the program. It was found that the written algorithm scaled correctly. Attempts to optimize the algorithm are also documented.

NOMENCLATURE

π	Pi
S_p	Speedup on p processes
T_1	Serial execution time
T_p	Parallel execution time
r	Permanently serial code
T_c	Communication execution time
E_p	Efficiency
T_c	Communication execution time

Equations

Amdahl's Law: $S_p = \frac{T_1}{(1-r)\frac{T_1}{p} + rT_1 + T_c}$

Efficiency (strong): $E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}$

Efficiency (weak): $E_p = \frac{T_1}{T_p}$

INTRODUCTION

Background

The Monte Carlo method uses randomly generated numbers as input to generate solutions and gain understanding of complex systems (Ref 1). The statistical method is used across information technology in applications from artificial intelligence to the stock market. With enough trials, strong correlations can be made to complex problems that are otherwise thought to be entirely random.

Project Objectives

The goal of this project is to apply the Monte Carlo method in a C algorithm to accurately predict the value of Pi. The goal is to analyze how the number of trials affects accuracy of the prediction and to test the parallel efficiency of the program itself. As test cases reach 9, 10 and even 11 digits, parallelization and optimization become important as to execute the program in the shortest amount of time possible.

Theory

Ideally, as the number of trials increases, the percent error from the actual value of Pi should decrease. This is a core concept of the monte carlo method. This will be tested serially and will be seen as parallel tests are run. Optimal performance of a parallel program is *linear speedup.* This can be shown by the following equation.

$$T_p = \frac{T_1}{p} \quad \text{thus,} \quad S_p = p$$

In order for scalability to be close to linear, it is necessary that the time spent on processor communication is considerably smaller than the total parallel execution time (shown below).

$$T_c \ll \frac{T_1}{p}$$

EXPERIMENTAL METHODS

Environment

Using a secure shell connection to Boise State's R2 compute cluster, I compiled both my serial and parallel programs in C and wrote bash scripts to run the program with a desired amount of processors, nodes, and trials on the supercomputer. The VSCode IDE was used in my local environment for most development while the VIM text editor was used to touch up on final details while within the virtual machine. Although redundant, as data was collected into text files, I saved the data I needed into convenient CSV files in order to be easily interpreted by MATLAB and plotted to show results.

Procedure

Each time a program is submitted to R2's queue, you can make sure your code was executed using the *squeue* command. After this, you will have an output file with the name you designated followed by the R2 job number. To see how to do this, visit the github repository linked in references. These files can be conveniently downloaded to a local workstation using the *scp* command or a linux tool like MobaXTerm. From there, the data can be analyzed in a spreadsheet, MATLAB, or any other convenient and effective way.

Data Reduction

Due to the nature of the Monte Carlo method and the random seeds of my program's generated numbers, there was no need to run multiple trials for each evaluation. Every set of data was representative of how efficient the program was and how accurately the algorithm could calculate Pi. Data was collected in the form of execution time, accuracy, number of trials, and number of processors and reduced to demonstrate efficiency and speedup.

RESULTS

Experiment Results

All plots and data analyzation will be shown in the following pages for convenience and clarity. The plots and discussions surrounding them were generated with MATLAB mlx files.

REFERENCES

[1] Britannica, The Editors of Encyclopaedia. "Monte Carlo method". *Encyclopedia Britannica*, 10 Jan. 2022, https://www.britannica.com/science/Monte-Carlo-method. Accessed 1 March 2022.

[2] "C Programming Language." 1 March 2022. https://devdocs.io/c/.

[3] "What is Monte Carlo Simulation?" IBM. 1 March 2022. https://www.ibm.com/cloud/learn/monte-carlo-simulation.

[4] "MPICH." 1 March 2022. https://www.mpich.org/documentation/.