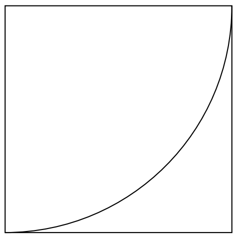
ECE 459: Programming for Performance
Assignment 1

Jon Eyolfson

January 13, 2012 (Due: January 30, 2012)

Background

For this assignment, we'll be parallelizing the **Monte Carlo Estimation for Pi**. It's a crude estimation that works by generating a bunch of random numbers between [0, 1]. Consider a 1×1 square with a quarter circle enclosed within.



We can estimate π by using a bunch of random points and seeing how many are contained within the quarter circle. We pick two random numbers x and y, check if $x^2 + y^2 \leq 1$, and if it does we record it. If we pick i random points, and c falls within the quarter circle. We can estimate π by simply calculating $4 \cdot \frac{c}{i}$ (since the area ratio of the quarter circle to the sqaure is $\frac{\pi}{4}$). The program will generate i random points, determine c by calculating how many are enclosed in the quarter circle and finally output π.

Setup

Download the provided assignment code from `http://ece459.eyolfson.com/media/assignments/provided-assignment-01.tar.gz` and untar it using the `tar xzvf assignment-01.tar.gz` command.

You should do this assignment in Linux, as the provided `Makefile` was only tested on Linux and isn't very robust. Since we're testing parallel execution, you may not want to use a virtual machine and instead run it natively (most free ones don't support multiple CPUs well). You of course can use the course machine, `ece459-1.uwaterloo.ca`, just e-mail me with your `uwaterloo.ca` address for `ssh` access. I tested everything on this system, so everything should be working. Just in case you want to make sure you setup is the same, I used `gcc 4.6.2` and `glibc 2.15`. Make sure to use the optimization flag `-O2`, or you may get some weird results.

Part 1 - Programming (60 marks)

Use the `pthread` library create a threaded version of the provided program. All of your parallel code should be within the `#ifdef PARALLEL` sections. Your program should create as many threads as the `num_threads` variable (which reads the value from the `-t` command line option). As a reminder, you should make sure all of your external calls are *thread-safe* (for example, `man 3 rand` to look at the documentation). **Edit: turns out `rand` is a lot slower than `rand_r`, so the sequential version actually uses `rand_r` so your comparsions make sense. You should still get use to looking up *thread-safe* calls.** Also, do not change the fundamental algorithm (as bad as it may be), since we want the same amount of done in the sequential and parallel versions.

Part 2 - Benchmarking (40 marks)

Run the sequential version with the number of iterations (set with the `-i` option) is sufficient for the program to run for at least 10 seconds. You may use the `time` command to benchmark your program. Run it at least 6 times and record the average. Assuming the code is 100% parallel, predict its runtime running on the number of physical cores in your system (4 for `ece459-1`, and should be at least 2). Explicitly state the number of physical cores in the machine. Next, run your parallel version with the same number of iterations and the amount of threads set to the number of physical cores in your system. Run this at least 6 times and record the average. Is your parallel runtime approximately equal to your predicted runtime? Write a few sentences describing why or why not your results agree with what you predicted.

Finally, if your CPU has hyperthreading, set the number of threads equal to the number of **virtual CPUs**, run your benchmark again 6 times and record the average. Calculate the speedup and verify it is less than the number of virtual CPUs. Now, set the number of threads equal to one more than the number of virtual CPUs (this will be the same as the number of physical cores if you don't have hyperthreading). Calculate the speedup and compare it to using an equal number of threads as virtual CPUs. Which performs better? Write a few sentences explaining any similarities or differences.

Submitting

Keep the same folder structure you were given. Archive in `tar.gz` format (or `zip` if you must) the `assignment-01` folder. It should contain the following in that folder: `Makefile`, `src/montecarlo.c` and `report.pdf` (you can modify the provided `report/report.tex` and create it with `make report`). After running `make` in the `assignment-01` folder it should produce two files, `bin/montecarlo` and `bin/montecarlo_parallel` in that folder. When completed, submit your archive on the course website, `http://ece459.eyolfson.com/` after logging in.