Parallel Program 1

Jennifer Kulich

September 18, 2020

1 Sieve of Eratosthenes

1.1 Description

This program will take in a number in which they would like to see all the primes that occur before that number. I did have some issues though. I know there's a loop dependency, and I don't think I did things correct (it's hard to tell because it runs correctly). Not going to lie though, I'm kind of proud of myself for knowing what's going on (even if I can't fix it right now) because this stuff is something I'm not the best at.

1.2 Algorithms and Libraries Used

This program uses the sieve of Eratosthenes algorithm to find all of the primes. As for libraries, the omp library was needed to make it parallel.

1.3 Functions and Program Structure

There's one function outside of main- the print primes function. I had issues returning an array from a function, and I thought it was more important to get other things done first.

1.4 How to Compile and User Program

To compile the program, you simply do make all. After that, you do ./prime ¡number¿ where number is the number you want to calculate the primes to. Please make sure to enter a number, and please only enter an integer.
To clean, you simply have to do make clean

1.5 Testing and Verification

For this, manual testing sufficed. I started by doing all primes from 1-10 to make sure it was working. After that, to test the parallel part, I would test the primes from 1-100. Once I felt comfortable with that, I tried large numbers such as 1,000,000.

2 Darts

2.1 Description

This program will simulate throwing darts on a dartboard. Instead of using a full dartboard, just 1/4 of it is used. There is a 1 ft radius of dartboard in the square that is either hit or not hit when a dart is 'thrown'. To throw a dart, you get a random x and a random y which is then calculated to see if it was within the circle or not.
For this, I used a reduction statement. You asked a question whether a reduction clause or a critical statement, and the reduction statement was quicker by about 0.03 seconds.
I also took out the parallel part and timed the program. Running it in parallel was sometimes slower slower- odd. That makes me think that I did things wrong, but I did a little research and it could just be because of omp overhead (I'm not sure if that's correct). Now, if I had used a critical statement, doing things sequentially would have been faster. The time was pretty similar.
I'd also like to note that I used rand... which is not actually random. I was struggling to find something to use that I understood (again, I like making things look pretty, so this was sort of a struggle). A lot of what I found still ended up being deterministic. Some people used the number of threads to make things a little more random, but I didn't think that it would be what you were looking for. It got to the point where I realized that I had other things to do. Sorry :(

2.2 Algorithms and Libraries Used

The algorithm for this was relatively easy. You loop through the number of darts you want to throw, get a random x and y, calculate if it's in the circle, and if it's in the circle, increment the number of darts in the circle. To check if the dart is in the circle, the x$^2 + y^2 willbe <= 1$.

2.3 Functions and Program Structure

This program was relatively small, so there are no functions other than main. Sorry :(

2.4 How to Compile and User Program

Since there's a makefile, you just have to do make clean;make all. After that, to compile you just do ./darts number of throws. I will describe this below, but a good number to enter for the number of darts to throw is 10,000,000. Please enter the number of darts to throw, and please only enter an integer.
To clean, simply do a make clean.

2.5 Testing and Verification

For this, I also did manual testing. I started by throwing 100 darts. This was not nearly close enough to see that the number would be pi, however it showed me that I was on the right track (I was getting either 0 or 1 for a while which was frustrating because I wasn't sure where things were wrong). I found when testing is serially (making sure that my algorithm was correct), 10,000,000 darts got me pretty close to Pi. The minimum seems to be about 1,000 darts- you'd get 3.17. At 100 darts, you dip to 2.76.

3 Overall- What's being submitted

Each program has a Makefile- how to make was described above in their respective section. Every program has just one file- the .c file with a main in it. The programs were small enough that it did not warrant classes or external files.