CSC 280
Homework 7
Spring 2020
Due Monday May 11 by 11:55 PM

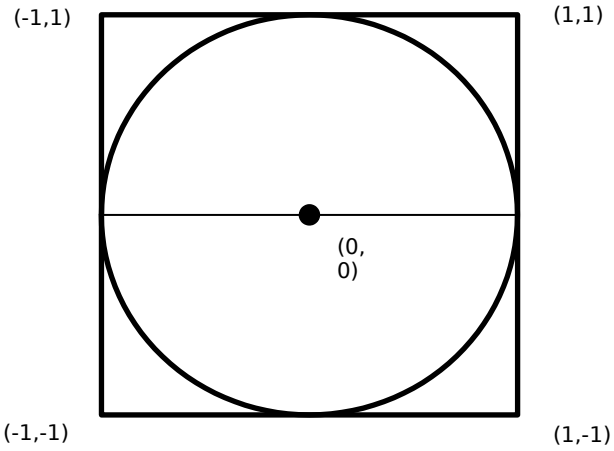
We're going to estimate the value of pi with our computer. We'll start with a sequential version of the program, then make a couple of multithreaded versions, and finally (Dave will) do some performance testing (and share the results with you).

Task 1: Sequential Version

We're going to use what's known as a Monte Carlo method to estimate pi.

"**Monte Carlo methods** (or **Monte Carlo experiments**) are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. Their essential idea is using randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three problem classes:[1] optimization, numerical integration, and generating draws from a probability distribution." (https://en.wikipedia.org/wiki/Monte_Carlo_method)

We're going to "throw darts" at a square with a center of (0,0) and a width (and height) of 2.



The area of the square is 2*2 = 4 units. The area of the unit circle inscribed in the square is $\pi*(1)^2$. The ratio of the area of the circle to the area of the square is $\pi / 4$. So if our darts are truly thrown randomly, then if we throw LOTS of darts, we should get a good approximation of pi by taking 4 * (the number of darts that land inside or on the circle / the number of darts thrown). If the math here confuses you, don't worry - take it on faith and then look at it tonight (or not – it's just algebra).

We'll generate random numbers between -1 and 1 for the x value and the y value of where a dart lands. If the x and y values are such that $x^2 + y^2 \leq 1$, then the dart was inside or on the circle. **MAKE SURE TO USE FLOATS FOR X AND Y!!!**

Here's some code to help:
https://stackoverflow.com/questions/33058848/generate-a-random-double-between-1-and-1

```
x = (double)rand()/RAND_MAX*2.0-1.0;//float in range -1 to 1
y = (double)rand()/RAND_MAX*2.0-1.0;//float in range -1 to 1
```

Remember that pi equals 4*(the number of darts that land inside or on the circle / the number of darts thrown).

Pi is approximately 3.14159265359. Run your program with 1, 10, 100, 1000, 10000, … darts until you get an approximation accurate to 3.141. **How many darts did it take?** _____

Now that you program is working, we can move on to the next task!

Task 2: Multithreaded Version 1

Write a multithreaded version of your program that uses multiple threads to throw the darts. Have

- a global variable **hits** to count the darts inside/on the circle and
- a global variable **misses** to count the darts outside the circle.

Whenever you throw a dart add 1 to **hits** or **misses** as appropriate. **Use a single mutex to prevent multiple threads from accessing those variables simultaneously.** Have your program take two command line arguments that indicates how many threads your program should use (first arg) and how many darts to throw (second arg). Make each thread throw its share of the darts. You may assume the user will enter a number of darts that is evenly divisible by the number of threads. You'll want to use array of threads. Make sure your program works right (don't forget to join your threads, etc.) by testing it (a good test is do the hits + misses add up to the number of darts thrown).

Save this program to hand in as pi1.c

Task 3: Second Multithreaded Version

This version will be almost the same, but have an array for hits and an array for misses and have each thread write to its own array slot for hits and misses. After all the threads have joined, do the math to calculate pi. You don't need (or want) a mutex for this version, since there are no race conditions.

Save this program to hand in as pi2.c

Hand in both your programs by Moodle. Dave will then distribute performance data for you to look at and an analysis!