**Project 9**

**Due 4/29/2016 by 11:59 PM**

**No delay is accepted**

**After this due date I won’t accept any project (late or current).**

**Post your answer *on Project 9 on blackboard.***

**If you have any question please email me at mbadii@pace.edu**

Your project for this week is the programming assignment 4.2 on pages 206-207of your textbook. Copy/paste your parallel program under the word: **Answer** in this file.

Suppose we toss darts randomly at a square dartboard, whose bullseye is at the origin, and whose sides are 2 feet in length. Suppose also that there is a circle inscribed in the square dartboard. The radius of the circle is 1 foot, and its area is π square feet. If the points that are hit by the darts are uniformly distributed (and we always hit the square), then the number of darts that hit inside the circle should approximately satisfy the equation

number in circle/total number of tosses = π/4 ,

since the ratio of the area of the circle to the area of the square is π/4.

We can use this formula to estimate the value of π with a random number generator:

number in circle = 0;

for (toss = 0; toss < number of tosses; toss++) {

x = random double between −1 and 1;

y = random double between −1 and 1;

distance squared = x∗x + y∗y;

if (distance squared <= 1) number in circle++;

}

pi estimate = 4∗number in circle/((double) number of tosses);

This is called a “Monte Carlo” method, since it uses randomness (the dart tosses).

Write a Pthreads program that uses a Monte Carlo method to estimate π. The main thread should read in the total number of tosses and print the estimate. You may want to use long long ints for the number of hits in the circle and the number of tosses, since both may have to be very large to get a reasonable estimate of π .

**Answer**:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

long long int tosses =100000;

long long int number\_in\_circle=0 ;

pthread\_mutex\_t mutex;

void \*MonteClare\_Approach(void\* \_);

double grabRand() ;

int main(int argc,char \*agrv[]){

pthread\_t\* thread\_handles;

double number\_of\_tosses;

srand(time(NULL));

double pi\_estimate;

int thread,thread\_count;

printf("How many threads? ");

scanf("%d", &thread\_count);

if(thread\_count>tosses || thread\_count<1)

{

printf("Supply an integer thread count between 1 and %llu tosses \n",tosses);

return 1;

}

thread\_handles = malloc(thread\_count\*sizeof(pthread\_t));

tosses = tosses/thread\_count;

pthread\_mutex\_init(&mutex, NULL);

for (thread = 0; thread < thread\_count; thread++)

{

pthread\_create(&thread\_handles[thread], NULL, MonteClare\_Approach,NULL);

}

for (thread = 0; thread < thread\_count; thread++)

{

pthread\_join(thread\_handles[thread], NULL);

}

pthread\_mutex\_destroy(&mutex);

free(thread\_handles);

number\_of\_tosses = thread\_count\*tosses;

printf("Total tosses : %g\n",number\_of\_tosses);

printf("Number in circle %llu \n",number\_in\_circle);

pi\_estimate = 4\*number\_in\_circle/number\_of\_tosses;

printf("Estimation of PI : \t%.4f \n",pi\_estimate);

return 0;

}

double grabRand()

{

double div = RAND\_MAX / 2;

return -1 + (rand() / div);

}

void \*MonteClare\_Approach(void\* \_){

long long int mycircleCount =0;

int toss;

for (toss=0;toss<tosses; toss++) {

double x = grabRand();

double y = grabRand();

double distance\_squared = (x\*x) + (y\*y);

if (distance\_squared <= 1)

{

mycircleCount++;

}

}

pthread\_mutex\_lock(&mutex);

number\_in\_circle +=mycircleCount;

pthread\_mutex\_unlock(&mutex);

return NULL;

}