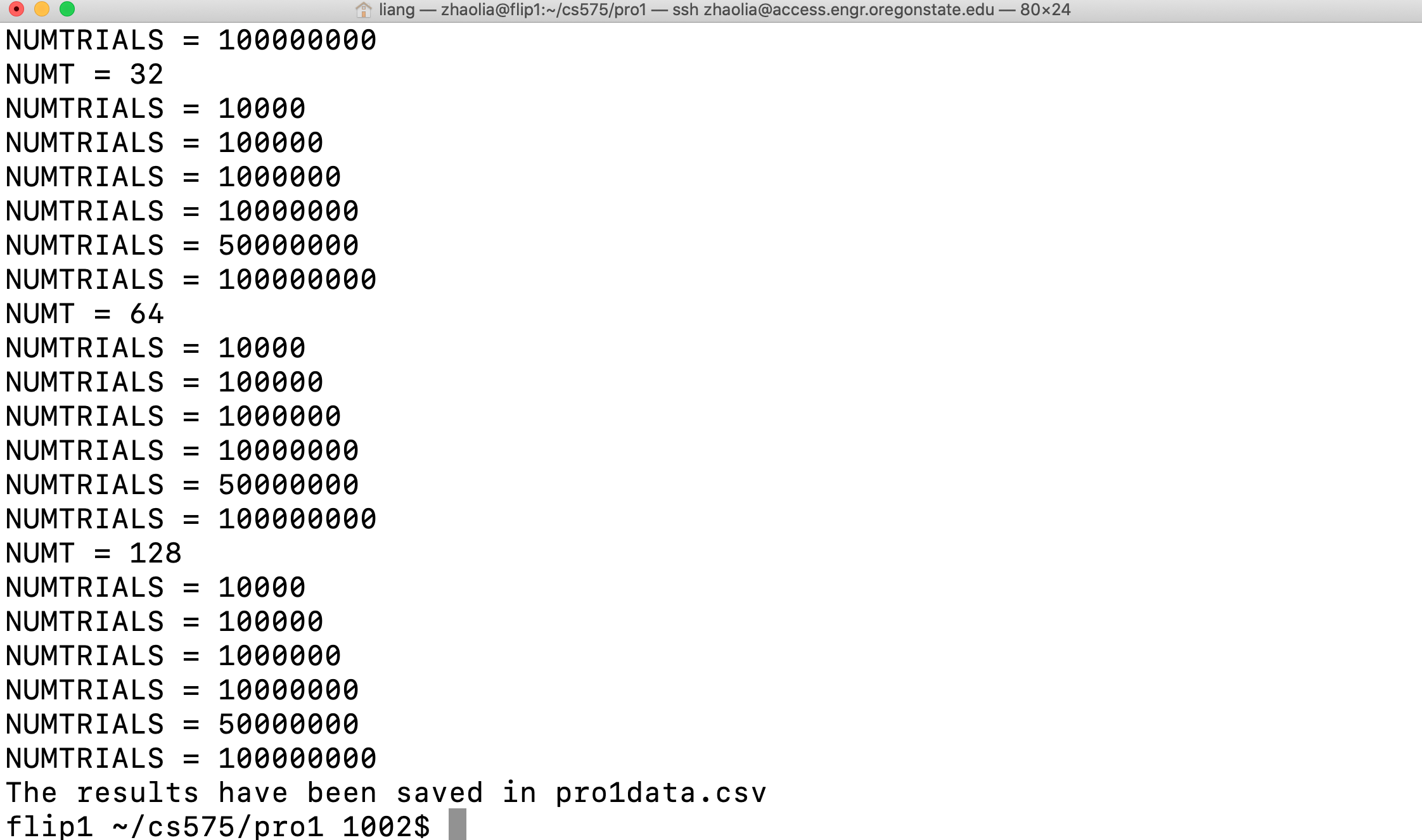
**CS 475/575 -- Spring Quarter 2020**

**Project #1**

**OpenMP: Monte Carlo Simulation**

**100 Points**

**Due: April 15**

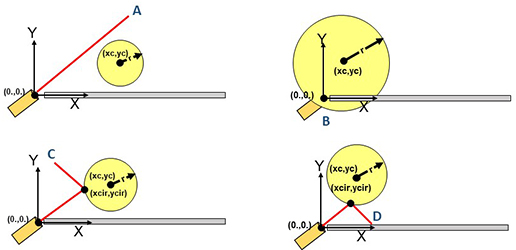


**Liang Zhao**

**933-667-879**

**zhaolia@oregonstate.edu**

Hi, this project is a job for multicore Monte Carlo simulation to determine whether the laser reaches the board below. The four situations that will appear in this experiment are shown in the figure below.



My program run on OSU Linux server: access.engr.oregonstate.edu. I used a bash script to automatically adjust the number of threads and trials, and save the results in a csv file. Let us first look at the results of the experiment.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Threads** | **Number of Trials** | **Probability of Hitting the Plate** | **Megatrials per Second** |
| 1 | 1000 | 0.142 | 9.681 |
| 1 | 10000 | 0.1348 | 18.871 |
| 1 | 50000 | 0.13152 | 18.691 |
| 1 | 100000 | 0.13104 | 18.871 |
| 1 | 200000 | 0.130965 | 18.751 |
| 1 | 300000 | 0.13041 | 18.741 |
| 1 | 500000 | 0.13078 | 18.63 |
| 2 | 1000 | 0.123 | 30.552 |
| 2 | 10000 | 0.1312 | 31.042 |
| 2 | 50000 | 0.1288 | 31.612 |
| 2 | 100000 | 0.12973 | 36.362 |
| 2 | 200000 | 0.130925 | 19.652 |
| 2 | 300000 | 0.13147 | 36.332 |
| 2 | 500000 | 0.131156 | 36.51 |
| 4 | 1000 | 0.118 | 61.184 |
| 4 | 10000 | 0.1261 | 72.594 |
| 4 | 50000 | 0.13238 | 39.064 |
| 4 | 100000 | 0.13164 | 39.554 |
| 4 | 200000 | 0.13071 | 39.424 |
| 4 | 300000 | 0.13048 | 39.314 |
| 4 | 500000 | 0.130912 | 72.45 |
| 8 | 1000 | 0.123 | 93.308 |
| 8 | 10000 | 0.1248 | 138.778 |
| 8 | 50000 | 0.1323 | 141.908 |
| 8 | 100000 | 0.13229 | 143.608 |
| 8 | 200000 | 0.13176 | 78.688 |
| 8 | 300000 | 0.1321167 | 76.948 |
| 8 | 500000 | 0.131694 | 77.62 |
| 16 | 1000 | 0.154 | 61.2216 |
| 16 | 10000 | 0.1377 | 96.7616 |
| 16 | 50000 | 0.13256 | 104.8016 |
| 16 | 100000 | 0.13204 | 105.0916 |
| 16 | 200000 | 0.1323 | 105.2316 |
| 16 | 300000 | 0.1325933 | 103.7816 |
| 16 | 500000 | 0.131952 | 104.27 |
| 32 | 1000 | 0.129 | 18.6732 |
| 32 | 10000 | 0.1288 | 68.8532 |
| 32 | 50000 | 0.13098 | 102.3732 |
| 32 | 100000 | 0.13069 | 114.5032 |
| 32 | 200000 | 0.130165 | 119.9432 |
| 32 | 300000 | 0.1304367 | 177.4732 |
| 32 | 500000 | 0.13107 | 184.21 |
| 64 | 1000 | 0.119 | 10.6864 |
| 64 | 10000 | 0.1347 | 43.9564 |
| 64 | 50000 | 0.13152 | 80.0564 |
| 64 | 100000 | 0.13043 | 116.2864 |
| 64 | 200000 | 0.130875 | 140.9064 |
| 64 | 300000 | 0.1312867 | 148.0264 |
| 64 | 500000 | 0.131308 | 180.05 |

(1): According to the actual number of random tests, the more the closer to the theoretical probability, so I think the actual probability is 13%.

(2): The graph of performance vs. number of trials

(3) The graph of performance vs. number of threads

(4)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Threads** | **1000 Trials** | **10000 Trials** | **50000 Trials** | **100000 Trials** | **200000 Trials** | **300000 Trials** | **500000 Trials** |
| 1 | 9.831 | 9.051 | 9.941 | 9.861 | 10.15 | 10.43 | 10.118 |
| 2 | 19.502 | 19.872 | 19.602 | 21.422 | 21.462 | 21.672 | 21.89 |
| 4 | 35.768 | 37.76 | 45.78 | 39.714 | 39.464 | 39.034 | 42.69 |
| 8 | 62.728 | 73.978 | 75.898 | 79.158 | 78.908 | 78.648 | 78.51 |
| 16 | 64.3316 | 94.2616 | 105.6816 | 105.9616 | 105.2816 | 107.2716 | 105.44 |
| 32 | 80.907 | 101.982 | 105.1232 | 111.9232 | 147.6332 | 177.1732 | 181.5 |

I choose the commonly used 8 threads and 1 thread for speed comparison base on 500000 Trials.

Speedup, S = (Execution time with one thread) / (Execution time with four threads) = (Performance with eight threads) / (Performance with one thread)

float Fp = (8./7.)\*( 1. - (1./S) );

So, S = (78.51/10.118) = 7.76

Fp = (8/7)\*(1-(1/7.76)) = 0.9955

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

#include <stdio.h>

// setting the number of threads:

#ifndef NUMT

#define NUMT 64

#endif

// setting the number of trials in the monte carlo simulation:

#ifndef NUMTRIALS

#define NUMTRIALS   10000

#endif

// how many tries to discover the maximum performance:

#ifndef NUMTRIES

#define NUMTRIES 10

#endif

// ranges for the random numbers:

const float XCMIN = -1.0;

const float XCMAX =  1.0;

const float YCMIN =  0.0;

const float YCMAX =  2.0;

const float RMIN  =  0.5;

const float RMAX  =  2.0;

// function prototypes:

float       Ranf( float, float );

int         Ranf( int, int );

void        TimeOfDaySeed( );

// main program:

int

main( int argc, char \*argv[ ] )

{

#ifndef \_OPENMP

    fprintf( stderr, "No OpenMP support!\n" );

    return 1;

#endif

    float tn = tan( (M\_PI/180.)\*30. );

    TimeOfDaySeed( );       // seed the random number generator

    omp\_set\_num\_threads( NUMT );    // set the number of threads to use in the for-loop:`

    // better to define these here so that the rand() calls don't get into the thread timing:

    float \*xcs = new float [NUMTRIALS];

    float \*ycs = new float [NUMTRIALS];

    float \* rs = new float [NUMTRIALS];

    // fill the random-value arrays:

    for( int n = 0; n < NUMTRIALS; n++ )

    {

        xcs[n] = Ranf( XCMIN, XCMAX );

        ycs[n] = Ranf( YCMIN, YCMAX );

        rs[n] = Ranf(  RMIN,  RMAX );

    }

    // get ready to record the maximum performance and the probability:

    float maxPerformance = 0.;      // must be declared outside the NUMTRIES loop

    float currentProb;              // must be declared outside the NUMTRIES loop

    //looking for the maximum performance:

    for (int t = 0; t < NUMTRIES; t++)

    {

        double time0 = omp\_get\_wtime();

        int numHits = 0;

        #pragma omp parallel for default(none) shared(xcs,ycs,rs,tn) reduction(+:numHits)

        for (int n = 0; n < NUMTRIALS; n++)

        {

            // randomize the location and radius of the circle:

            float xc = xcs[n];

            float yc = ycs[n];

            float  r = rs[n];

            // solve for the intersection using the quadratic formula:

            float a = 1. + tn \* tn;

            float b = -2.\*(xc + yc \* tn);

            float c = xc \* xc + yc \* yc - r \* r;

            float d = b \* b - 4.\*a\*c;

            // case A: If d is less than 0., then the circle was completely missed. (Case A) Continue on to the next trial in the for-loop.

            if (d < 0) {

                continue; //A

            }

            // hits the circle:

            // get the first intersection:

            d = sqrt(d);

            float t1 = (-b + d) / (2.\*a);   // time to intersect the circle

            float t2 = (-b - d) / (2.\*a);   // time to intersect the circle

            float tmin = t1 < t2 ? t1 : t2;     // only care about the first intersection

            // case B: the circle completely engulfs the laser pointer

            if (tmin < 0) {

                continue;  //B

            }

            // where does it intersect the circle?

            float xcir = tmin;

            float ycir = tmin \* tn;

            // get the unitized normal vector at the point of intersection:

            float nx = xcir - xc;

            float ny = ycir - yc;

            float nxy = sqrt(nx\*nx + ny \* ny);

            nx /= nxy;  // unit vector

            ny /= nxy;  // unit vector

            // get the unitized incoming vector:

            float inx = xcir - 0.;

            float iny = ycir - 0.;

            float in = sqrt(inx\*inx + iny \* iny);

            inx /= in;  // unit vector

            iny /= in;  // unit vector

            // get the outgoing (bounced) vector:

            float dot = inx \* nx + iny \* ny;

            float outx = inx - 2.\*nx\*dot;   // angle of reflection = angle of incidence`

            float outy = iny - 2.\*ny\*dot;   // angle of reflection = angle of incidence`

            // find out if it hits the infinite plate:

            float tt = (0. - ycir) / outy;

            // case C: reflected beam went up instead of down

            if (tt < 0) {

                continue;

            }

            // case D:

            numHits++;

        }

        double time1 = omp\_get\_wtime();

        double megaTrialsPerSecond = (double)NUMTRIALS / (time1 - time0) / 1000000.;

        if (megaTrialsPerSecond > maxPerformance)

            maxPerformance = megaTrialsPerSecond;

        currentProb = (float)numHits / (float)NUMTRIALS;

    }

    // printf("Number of threads: %d\n", NUMT);

    // printf("Number of trials: %d\n", NUMTRIALS);

    // printf("Probability: %1.7lf\n", currentProb);

    // printf("MegaTrialsPerSecond: %8.2lf\n", maxPerformance);

    //printf("Threads: %d\t Trials: %d\t Probability:  %8.2lf\t MegaTrialsPerSecond: %8.2lf\t\n", NUMT, NUMTRIALS, currentProb, maxPerformance);

    printf("%d,%d,%1.7lf,%8.2lf", NUMT, NUMTRIALS, currentProb, maxPerformance);

}

//Helper Functions

float

Ranf( float low, float high )

{

        float r = (float) rand();               // 0 - RAND\_MAX

        float t = r  /  (float) RAND\_MAX;       // 0. - 1.

        return   low  +  t \* ( high - low );

}

int

Ranf( int ilow, int ihigh )

{

        float low = (float)ilow;

        float high = ceil( (float)ihigh );

        return (int) Ranf(low,high);

}

void

TimeOfDaySeed( )

{

    struct tm y2k = { 0 };

    y2k.tm\_hour = 0;   y2k.tm\_min = 0; y2k.tm\_sec = 0;

    y2k.tm\_year = 100; y2k.tm\_mon = 0; y2k.tm\_mday = 1;

    time\_t  timer;

    time( &timer );

    double seconds = difftime( timer, mktime(&y2k) );

    unsigned int seed = (unsigned int)( 1000.\*seconds );    // milliseconds

    srand( seed );

}

#!/bin/bash

touch pro1data.csv

rm pro1data.csv

echo "Number of Threads, Number of Trials, Probability of Hitting the Plate, Megatrials per Second" >> pro1data.csv

# number of threads:

for t in 1 2 4 8 16 32 64 128

do

    echo NUMT = $t

    # number of trials:

    for s in 10000 100000 1000000 10000000 50000000 100000000

    do

        echo NUMTRIALS = $s

        g++ -DNUMTRIALS=$s -DNUMT=$t pro1.cpp -o pro1 -lm -fopenmp

        ./pro1 >> pro1data.csv

    done

    echo -e >> pro1data.csv

done

echo "The results have been saved in pro1data.csv"