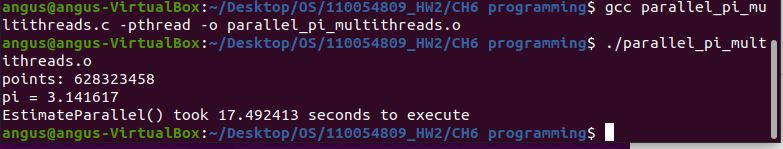
Steps：

1. gcc –pthread parallel\_pi\_multithreads.c –o parallel\_pi\_multithreads.o
2. ./ parallel\_pi\_multithreads.o

Source code：

#include <stdio.h>

#include <pthread.h>

#include <math.h>

#include <stdlib.h>

#include <limits.h>

#include <time.h>

#define SEED 7

unsigned int seed = 7;

pthread\_mutex\_t mutex;

// defining the total number of points from which random points are selected

int N = 800000000;

//global variable for number of points inside unit circle

int totalPts = 0;

//Random number generator with linear congruential generator

double RandUint (long i) {

seed = seed \* 1103515245 + 123456;

return seed / (double)UINT\_MAX;

}

//function to estimate pi using pthreads

void \*EstimateParallel (void \*param) {

int len = \*(int \*) param;

for (int i = 0; i < len; i++) {

double x = (double)RandUint (i);

double y = (double)RandUint (i);

pthread\_mutex\_lock(&mutex);

if (((x \* x) + (y \* y)) <= 1) {

totalPts++;

}

pthread\_mutex\_unlock(&mutex);

}

pthread\_exit(0);

}

int main () {

int threads = 10;

int pointsPerThread = N/threads;

double pi = 0;

// calculating the time taken by the function estimate()

clock\_t t;

t = clock ();

pthread\_t runners[threads];

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

for (size\_t i = 0; i < threads; i++) {

pthread\_create(&runners[i], 0, &EstimateParallel, &pointsPerThread);

}

for (size\_t i = 0; i < threads; i++) {

pthread\_join(runners[i], NULL);

}

t = clock () - t;

// printf("%f")

pi = (4.0 \* totalPts) / N;

printf("points: %d\n", totalPts);

double timeTaken = ((double)t) / CLOCKS\_PER\_SEC; // in seconds

printf("pi = %lf\n", pi);

printf("EstimateParallel() took %f seconds to execute \n", timeTaken);

return 0;

}