HPC_Spring 2017

OpenMP Mini Project

Kaiqin Huang

The algorithm I chose to work on is Monte Carlo. The algorithm is wildly applied in fields such as mathematics, physical sciences, engineering, climate change, computational biology, computer graphics, artificial intelligence for games, finance and business, law, and so on.

Monte Carlo methods are a broad class of computational algorithms. It relies on repeated random sampling to obtain numerical results. The essential idea is using randomness to solve problems that might be deterministic in principle. Monte Carlo methods are mainly used in three distinct problem classes: optimization, numerical integration, and generating draws from a probability distribution. In principle, Monte Carlo methods can be used to solve any problem having a probabilistic interpretation.

Monte Carlo methods vary, but tend to follow a particular pattern:

1. Define a domain of possible inputs.
2. Generate inputs randomly from a probability distribution over the domain.
3. Perform a deterministic computation on the inputs.
4. Aggregate the results.

A classic example is to estimate π. Consider a circle inscribed in a unit square. Given that the circle and the square have a ratio of areas that is π/4, the value of π can be approximated using a Monte Carlo method:

1. Draw a square, and then inscribe a circle within it.
2. Uniformly scatter objects of uniform size over the square.
3. Count the number of objects inside the circle and the total number of objects.
4. The ratio of the two counts is an estimate of the ratio of the two areas, which is π/4. Multiply the result by 4 to estimate π.

In this procedure the domain of inputs is the square that circumscribes our circle. We generate random inputs by scattering grains over the square then perform a computation on each input (test whether it falls within the circle). Finally, we aggregate the results to obtain our final result, the approximation of π.

Code is easily found online. Originally, the number of iterations is a user input. I changed it to a default number 500000 in order to calculate the run time more conveniently. Also edited it into a more readable format and added comments. Ran with and without printing out the thread status.

Surprisingly, what I find is that the sequential code is always the fastest, no matter printing out the thread work or not.

```
#include <omp.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define SEED 42
#define nthreads 4

int main(int argc, char* argv)
{
	double start_time = omp_get_wtime();

	int niter = 500000;	/* # of iterations */
	double x, y;	/* x and y coordinates */
	int i, tid, count = 0;	/* # of points in the 1st quadrant of unit circle */
	double z;	/* Distance from original */
	double pi;	/* Estimated pi value */
	srand(SEED);	/* Initialize random number generator; srand is used to seed the random number generated by rand() */

	#pragma omp parallel for private(x,y,z,tid) reduction(+:count) num_threads(nthreads)
	/* Initialize OpenMP parallel for with reduction(∑) */

	for (i = 0; i < niter; i++) {
		/* Randomly generate a number and divide it by the max possible, so we get a ratio between 0 and 1 */
		x = (double)rand() / RAND_MAX;
		y = (double)rand() / RAND_MAX;
		z = (x * x + y * y);	/* Calculate the diagonal */

		if (z <= 1) count++;	/* Assume the square's length of side (and also the circle's radius) is 1 */
		/* Check if it lies within the circle; if yes then increment count */

		/* Each of the following if statements is printing out the result of a specific thread */
		if (i==(niter/8)-1) {
			tid = omp_get_thread_num();
			printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
		}
		if (i==(2*niter/8)-1) {
			tid = omp_get_thread_num();
			printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
		}
		if (i==(3*niter/8)-1) {
```

```
                tid = omp_get_thread_num();
                printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
            }
            if (i==(4*niter/8)-1) {
                tid = omp_get_thread_num();
                printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
            }
            if (i==(5*niter/8)-1) {
                tid = omp_get_thread_num();
                printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
            }
            if (i==(6*niter/8)-1) {
                tid = omp_get_thread_num();
                printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
            }
            if (i==(7*niter/8)-1) {
                tid = omp_get_thread_num();
                printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
            }
            if (i==(8*niter/8)-1) {
                tid = omp_get_thread_num();
                printf(" thread %i just did iteration %i the count is %i\n",tid,i,count);
            }
        }

        printf( "The total count is: %i \n", count);
        pi = (double)count / niter * 4;   /* Calculate PI based on the aggregate count of the
points that lie within the circle */
        printf("# of trials = %d, estimate of pi is %g \n", niter, pi);
        double time = omp_get_wtime() - start_time;    /* To get the run time */
        printf("# of threads = %d, run time is %f \n", nthreads, time);
        return 0;
}
```

Without printing:

```
The total count is: 392634
# of trials = 500000, estimate of pi is 3.14107
# of threads = 1, run time is 0.013475

The total count is: 392528
# of trials = 500000, estimate of pi is 3.14022
# of threads = 2, run time is 0.123137

The total count is: 392555
# of trials = 500000, estimate of pi is 3.14044
# of threads = 4, run time is 0.152256

The total count is: 392294
# of trials = 500000, estimate of pi is 3.13835
# of threads = 8, run time is 0.312842

The total count is: 392457
# of trials = 500000, estimate of pi is 3.13966
# of threads = 16, run time is 0.636141
```

With printing:

```
thread 0 just did iteration 62499 the count is 48918
 thread 0 just did iteration 124999 the count is 98150
 thread 0 just did iteration 187499 the count is 147246
 thread 0 just did iteration 249999 the count is 196201
 thread 0 just did iteration 312499 the count is 245261
 thread 0 just did iteration 374999 the count is 294477
 thread 0 just did iteration 437499 the count is 343635
 thread 0 just did iteration 499999 the count is 392634
The total count is: 392634
# of trials = 500000, estimate of pi is 3.14107
# of threads = 1, run time is 0.032059


thread 0 just did iteration 62499 the count is 48958
 thread 1 just did iteration 312499 the count is 49188
 thread 1 just did iteration 374999 the count is 98104
 thread 0 just did iteration 124999 the count is 98044
 thread 0 just did iteration 187499 the count is 147170
 thread 1 just did iteration 437499 the count is 147131
```

```
 thread 0 just did iteration 249999 the count is 196291
 thread 1 just did iteration 499999 the count is 196181
The total count is: 392472
# of trials = 500000, estimate of pi is 3.13978
# of threads = 2, run time is 0.249733


 thread 1 just did iteration 187499 the count is 48889
 thread 3 just did iteration 437499 the count is 48830
 thread 2 just did iteration 312499 the count is 49114
 thread 0 just did iteration 62499 the count is 49133
 thread 1 just did iteration 249999 the count is 98004
 thread 3 just did iteration 499999 the count is 97999
 thread 2 just did iteration 374999 the count is 98245
 thread 0 just did iteration 124999 the count is 98197
The total count is: 392445
# of trials = 500000, estimate of pi is 3.13956
# of threads = 4, run time is 0.277669


 thread 1 just did iteration 124999 the count is 48941
 thread 3 just did iteration 249999 the count is 49072
 thread 7 just did iteration 499999 the count is 49016
 thread 6 just did iteration 437499 the count is 49147
 thread 5 just did iteration 374999 the count is 49350
 thread 4 just did iteration 312499 the count is 48968
 thread 2 just did iteration 187499 the count is 49062
 thread 0 just did iteration 62499 the count is 49040
The total count is: 392596
# of trials = 500000, estimate of pi is 3.14077
# of threads = 8, run time is 0.487853


 thread 1 just did iteration 62499 the count is 24427
 thread 7 just did iteration 249999 the count is 24516
 thread 3 just did iteration 124999 the count is 24575
 thread 13 just did iteration 437499 the count is 24531
 thread 11 just did iteration 374999 the count is 24500
 thread 5 just did iteration 187499 the count is 24517
 thread 15 just did iteration 499999 the count is 24506
 thread 9 just did iteration 312499 the count is 24580
The total count is: 392630
# of trials = 500000, estimate of pi is 3.14104
# of threads = 16, run time is 0.827409
```

References

https://en.wikipedia.org/wiki/Monte_Carlo_method

http://www.umsl.edu/~siegelj/cs4790/openmp/pimonti_omp.c.HTML

http://www.openmp.org

http://computing.llnl.gov/tutorials/openMP