Lab 1 Pseudorandom Number Generator

Ziyang Wang
ShanghaiTech University
wangzy4@shanghaitech.edu.cn

1. Sequential LCG

First, I started to write a sequential LCG using with m=2^{31}, a=1103515245 and c=12345(Seq_original.c).

```
int a = 1103515245;
int m = 0x80000000;
int c = 12345;
int x0 = 1;
long k = 10;
double time;

int main()
{
    int count=1;
    for (k = 10; k < 10000000000; k*=10)
    {
    long* lcg=malloc((k+1)*sizeof(long));
    lcg[0] = x0;
    time = omp_get_wtime();
    for (int i = 1; i < k+1; ++i)
    {
        lcg[i] = (a*lcg[i-1]+c)%m;
    }
    time = omp_get_wtime() - time;
    printf("k=10^%d,omp time %fs\n",count, time);
    count++;
    free(lcg);
    }
    return 0;
}
```

k	time/s
10	0.000000
10^2	0.000006
10^3	0.000022
10^4	0.000220
10^5	0.002206
10^6	0.020567
10^7	0.190699
10^8	1.930415
10^9	22.518042

After changing the code to parallel, the result of sequence became different. Because $x_{i+1} = (ax_i + c) mod\ m$, which cause data dependence between threads. If just add *#pragma omp parallel for* before the loop, LCG will run with error.

2. Parallel LCG with Leap Frog method

In A brief Introduction to parallel programming, it mentioned a threadsafe version of LCG. To make each threads run it's own sequence, it make sure that each thread runs the same sequence each time.

Here's the reference code:

```
static long a,m,c,x0;
long random_last = 0;
#pragma omp threadprivate(random_last)
double random(){
    long random_next;
    random_next = (a*random_last+c)%m;
    random_last = ramdom_next;

    return random_last;
}

#pragma omp single{
nthreads = omp_get_num_threads()
iseed = x0;
pseed[0] = iseed;
mult_n = a;
for(int i=1;i<nthreads;i++){
    iseed = (a*iseed)%m;
    pseed[i]=iseed;
    mult_n=(mult_n*a)%m;
}
}
random_last = pseed[omp_get_thread_num()];
#pragma omp for schedule(static)
for(int i=0;i<k+1;i++){
    seq[i]=random();
}
```

After modification, the program could generate same LCG each time. While due to the scheduler of openmp, the order of the sequence is different with the original sequence, but it could be corrected in $O(1)$ time (Corrected in Seq.c).

3. Result

Here's the final result with 8 threads:

k	Sequential time/s	Parallel time/s	Speedup
10	0.000001	0.000447	0.002
10^2	0.000002	0.000131	0.015
10^3	0.000025	0.000157	0.159
10^4	0.000240	0.000263	0.913
10^5	0.002421	0.001414	1.172
10^6	0.024508	0.009165	2.674
10^7	0.206334	0.064390	3.204
10^8	2.019135	0.607273	3.325
10^9	21.393735	8.579351	2.494

Parallel LCG is much slower than sequential when k is small, but it performs better with k increasing.

4. Reference

1. A brief Introduction to parallel programming
2. Linear congruential generator - Wikipedia