Bull. Nov. Comp. Center, Comp. Science, 27 (2008), 51–53

Acceleration of linear congruential generators

K.V. Kalgin

Abstract. For implementation of pseudo-random number generators (PRNG), a linear congruential generator (LCG) is the most frequently used algorithm. The techniques to increase the efficiency of the LCG based on 64-bit integers and bitwise operations are studied. The efficiency of the LCG implementation with the techniques proposed is experimentally tested.

1. Introduction

The LCG is defined by the recurrence relation:

$$X_{n+1} = (A \cdot X_n + B) \bmod M, \tag{1}$$

where X_n is the n-th number of a generated sequence, A, B, and M are integer constants that specify the generator.

The form of the LCG with $M = 2^r$ is the most frequently used for generating pseudo-random numbers, where r typically denotes the number of bits used to represent generated numbers, and A is a sufficiently large number relatively prime with respect to 2^r.

To create a LCG with a large period, we need to increase M, that is, the number of bits r. For the majority of Monte Carlo problems, $r \leq 64$ is quite sufficient, and $r \leq 128$ is more than that, however the PRNG acceleration is welcome. Hence, an increase in the efficiency of the LCG is needed. In this paper, the LCG accelerating techniques based on 64-bit integers and bitwise operations are studied.

Next we consider 40-bit LCG [1] in detail with $A = 5^{17}$, $B = 0$, $M = 2^{40}$, with the corresponding period $L = 2^{38}$. It is extensively used and carefully tested in [1]. A detailed consideration and the source code for the 128-bit LCG, used for the parallel PRNG[2] and tested in [3], with the corresponding period 2^{126}, can be found in [4].

2. The LCG implementation

The double precision floating point numbers or several integer numbers for representing large numbers (exceeding 32-bit) are used in the majority of implementations to prevent the overflow.

In the standard of C language C99 [5], the following can be found:

1. Definition of new integer types, for example `int64_t` or `uint64_t` for 64-bit *signed* and *unsigned* integers, respectively.

2. Statement "A computation involving unsigned operands can never overflow, because a result that cannot be represented by the resulting unsigned integer is reduced modulo the number that is greater than the largest value that can be represented by the resulting type" [5].

So, for the LCG implementation with $r \leq 64$ according to (1), we can operate (multiply and sum) with 64-bit integers without being afraid of overflow, because we need only low-order bits. The program code of the LCG is as follows.

```
#include <inttypes.h>
const uint64_t A, B, r;
const uint64_t M = 2^r;
uint64_t lcg_r()
{
    static uint64_t prev = 1;
    prev = (prev * A + B) & (M - 1);
    return prev;
}
```

Particularly, the above 40-bit LCG further referred to as `prng40` is implemented in the following way:

```
#include <inttypes.h>
const uint64_t A = 5^17;
const uint64_t M = 2^40;
uint64_t lcg_40b()
{
    static uint64_t prev = 1;
    prev = (prev * A) & (M - 1);
    return prev;
}
```

3. Comparison of efficiency

Let us consider the following implementations of different PRNGs with linear congruential algorithm:

`prng40` original implementation based on the double precision floating point number to represent a large integer number can be found in [6, 7].

`drand48` GNU C library [8] implementation, C and UNIX standard generator.

Comparison of efficiency of implementations of the PRNG based on the LCG

PRNG	LCG parameters (A, B, M)	Time for generation 10^9 numbers, s	
		original	accelerated
`prng40`	$(5^{17}, 0, 2^{40})$	146.5	6.5
`drand48`	$(7 \cdot 433 \cdot 739 \cdot 11003, 11, 2^{48})$	171.5	8.9

In the table, the results of testing the above two PRNGs in their original form and accelerated by means of techniques proposed are given. Computations are performed on Pentium4 2.4 GHz, Linux OS, and compiled with GNU C compiler with `-O3` optimization flag.

4. Conclusion

The techniques to increase the efficiency of the LCG based on 64-bit integers and bitwise operations are studied and tested. Comparison of efficiency of various implementations of built-in `drand48` and 40-bit `prng40` PRNGs based on the LCG are performed. Experimental results show that the acceleration of the LCG is about 20 times.

References

[1] Ermakov S.M., Mikhailov G.A. Stochastic Simulation. — Moscow: Nauka, 1982.

[2] Marchenko M. Parallel pseudorandom number generator for large-scale Monte Carlo simulations // Proc. PaCT-2007. — Springer, 2007. — (LNCS; 4671).

[3] Dyadkin I.G., Hamilton K.G. A study of 128-bit multipliers for congruential pseudorandom number generators // Comput. Phys. Comm. — 2000. — Vol. 125. — P. 239–258.

[4] Acceleration of some linear congruential generators. — http://ssdonline.sscc.ru/kalgin/prng.html

[5] C99, ISO/IEC 9899/1999.

[6] Prigarin S.M. — http://osmf.sscc.ru/~smp

[7] Prigarin S.M. Spectral Models of Random Fields in Monte Carlo Methods. — Utrecht: VSP, 2001.

[8] The GNU C Library. — http://www.gnu.org/software/libtool/manual/libc.