Random Number Generation in Parallel

David Spade

Department of Statistics
The Ohio State University

Friday, April 6, 2012

Methods of Random Number Generation

I cover three methods of Random Number Generation here.

- Multiple Recursive Generator (MRG32k3a)
- Sobol Quasi-Random Number Generator
- Mersenne Twister (MT19937)

General Random Number Generation Algorithm

In each of the three methods:

- Each generator has a state Y_n, which has one or more variables, that can be advanced by some algorithm $Y_{n+1} = f(Y_n)$ from some initial value Y_0.
- There is an output process $x_n = g(Y_n)$ that generates an approximate Uniform[0,1) random number x_n.
- Parallelization is facilitated by a "skip ahead" strategy, where each RNG can skip ahead a given number p of steps, so that $Y_{n+p} = f_p(Y_n)$.

Three Skip Ahead Strategies

Simple Skip Ahead

- Performed at the thread level
- Each thead in each block skips ahead to a specified point in RNG sequence, and then a contiguous segment of points are generated on each thread.
- Skips are chosen so that segments are adjacent and do not overlap.

Three Skip Ahead Strategies(cont.)

Strided (leap-frog)

- Performed at thread level
- n^{th} thread out of N generates points $n, n+N, n+2N$, etc.

Three Skip Ahead Strategies(cont.)

Hybrid

- Performed at the block level
- In each block, each thread does strided generation.

L'Ecuyer's Multiple Recursive Generator (MRG32k3a)

MRG32k3a is defined by the following set of equations:

- $y_{1,n} = (a_{12}y_{1,n-2} + a_{13}y_{1,n-3}) \mod m_1$
- $y_{2,n} = (a_{21}y_{2,n-1} + a_{23}y_{2,n-3}) \mod m_2$
- $x_n = (y_{1,n} + y_{2,n}) \mod m_1$,

where $a_{12} = 1403580$, $a_{21} = 527612$, $a_{13} =$ -810728, $a_{23} =$ -1370589, $m_1 = 2^{32} - 209$, $m_2 = 2^{32} - 22853$, $n \geq 3$, and $x_3, x_4, x_5, \ldots$ are outputs of the generator. Dividing the outputs by m_1 gives pseudo-random Uniform[0,1) outputs.

L'Ecuyer's Multiple Recursive Generator (MRG32k3a)

Representation of States:

- At any point in the sequence, a state can be represented by the pair of vectors $Y_{i,n} = (y_{i,n}, y_{i,n-1}, y_{i,n-2})^T$, i=1,2.
- The recursion above yields that $Y_{i,n+1} = A_i Y_{i,n} \mod m_i$, $i = 1, 2$, where A_i is a 3×3 matrix.
- To skip ahead p steps, $Y_{i,n+p} = A_i^p Y_{i,n} \mod m_i$, $i = 1, 2$.

Since the state is small, the simple skip ahead strategy works well for CUDA implementation, provided one has an efficient method of calculating A_i^p.

Advantages/Disadvantages of MRG32k3a

Advantages:

- Very straightforward and easy to parallelize
- To generate random numbers, a kernel can be launched with any configuration of threads and blocks since parallelization is done at the thread level.

Disadvantage: If it is desired to improve the speed, then there is a substantial amount of memory sacrifice in doing so.

Sobol Generation

Begin with a sequence of k-bit integers, where 2^k represents the length of the sequence to be generated.

Starting with $y_0 = 0$, $y_1, y_2, \ldots$ are computed by the formula

$$\begin{aligned} y_n &= g_1 m_1 \oplus g_2 m_2 \oplus \ldots \oplus g_k m_k \\ &= y_{n-1} \oplus m_{f(n-1)}, \end{aligned}$$

where:

Notation for Sobol Generation

- g_i are bits in binary representation of $n \oplus \left[\frac{n}{2}\right]$, so that $n \oplus \left[\frac{n}{2}\right] = g_k \cdots g_2 g_1$.
- $f(n)$ returns index of rightmost zero bit in binary representation of n.
- To get the Sobol sequence $x_1, x_2, \ldots$, set $x_n = 2^{-k} y_n$.
- $\oplus$ is the binary exclusive or operator.
- Simple skip ahead is quite fast for running this in parallel. A strided algorithm is a bit faster.

Advantages and Disadvantages of Sobol Generation

Advantages:

- In the parallelization, the skip ahead algorithm is very fast, since it only requires a loop with k iterations.
- Also lends itself to a very fast algorithm for strided generation
- Has good multidimensional uniformity and statistical properties.

Disadvantages:

- If it is desired to generate multidimensional Sobol sequences, then one must be very careful in his or her choice of the m_i numbers, since poor choices can destroy the multidimensional uniformity properties that are at the heart of the method.

Mersenne Twister

The state of the generator is a vector, but the algorithm converts this vector to a pseudo-random number.

- Algorithm is based on the following linear recurrence:

$$X_{k+N} = X_{k+M} \oplus (X_k^u | X_{k+1}^l) D \forall k \geq 0,$$

  where M and N are fixed natural numbers, X_i is represented by a w bit string of 0s and 1s, $(X_k^u | X_{k+1}^l)$ is the concatenation of the $w - r$ most significant bits of X_k and the r least significant bits of X_{k+1} for some natural number $r \leq w$.
- D is a $w \times w$ matrix of 0s and 1s.
- The matrix multiplication in the recurrence relation is performed modulo 2.

Mersenne Twister(cont.)

The popular Mersenne Twister algorithm:

- $w = 32$, $N = 624$, $M = 397$, $r = 31$, $d_{31}d_{30}\ldots d_1$, which is the last row of the D matrix, is equal to 2567483615.
- The period of the generator is $2^{19937} - 1$.
- The state vector Y_k has 19937 individual bits, which are stored as 632 32 bit words with one bit left over. Y_k is represented as

$$Y_k = (X_{N-1+k}, X_{N-2+k}, \ldots, X_k)$$

Mersenne Twister(cont.)

- Y_0 is the initial seed for the generator.
- If Y_k is regarded as a list of individual bits and is read from top to bottom, the first 19937 are used and the last 31 are ignored.
- The ignored bits are the 31 least significant bits of X_k.
- Let A be a 1×19937 matrix. When the generator is considered as a list of individual bits, $Y_{k+1} = AY_k$, and $Y_k = A^k Y_0$, where matrix multiplication is performed mod 2.
- Parallelization is somewhat difficult here, and requires a hybrid method.

Advantages and Disadvantages of the Mersenne Twister

Advantages:

- Good multidimensional uniformity and statistical properties
- Very fast compared to other algorithms, so it is useful in simulations where huge quantities of random numbers are required.

Disadvantages:

- The size of the states rules out thread-level parallelization.

Summary

- Discussed three methods of implementing RNGs in CUDA.
- Discussed three random number generators.
- I mentioned, in a broad sense, the method used to implement the RNGs in CUDA.
- For further details on the parallelization of these three RNGs, see the Bradley et. al paper posted on Radu's GPU computing website.