PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 06-18

Comparación del desempeño de generadores de números aleatorios

Javier Cohenar González

2006

Comparación del desempeño de generadores de números aleatorios
Cohenar González, Javier
ISSN 0797-6410
Reporte Técnico RT 06-18
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, 2006

COMPARACIÓN DEL DESEMPEÑO DE GENERADORES DE NÚMEROS ALEATORIOS

Javier Cohenar González

Departamento de Investigación Operativa
Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Uruguay

ABSTRACT

En este breve reporte se comentan las principales características generales de tres generadores de números aleatorios y se hace un pequeño experimento para comparar su desempeño en términos de tiempo de cómputo. En el estudio se incluye al algoritmo Mersenne Twister de Matsumoto y Nishimura, el cual goza de gran popularidad debido a su buen desempeño en los distintos benchmarks.

I) Introducción

En este pequeño reporte se comentan las principales características generales de tres generadores de números aleatorios, dando especial énfasis a temas relacionados con la implementación práctica de los algoritmos, como ser, facilidad de instalación y precauciones en el uso y disponibilidad en distintos lenguajes. También se mostrarán los resultados de un pequeño experimento para comparar su desempeño en términos de tiempo de cómputo. En el estudio se incluye al algoritmo Mersenne Twister de Matsumoto y Nishimura, el cual ha pasado exitosamente tests estrictos como *diehard* de Marsaglia o *load* de Hellekalek y Wegenkittl.

El reporte está escrito de la siguiente manera: en II se presentan los algoritmos a utilizar; en III se presentan los resultados del estudio de tiempos realizado y en IV las conclusiones. En el Apéndice A, se puede encontrar el código C++ para la adaptación a dicho lenguaje del generador UNIRAN.

II) Breve descripción de los generadores

MERSENNE TWISTER

Mersenne Twister [1] fue creado por Matsumoto y Nishimura y pertenece a la familia de generadores lineales. Algunos méritos del algoritmo según los autores [2] son:

- Fue diseñado considerando las fallas de los generadores existentes.
- Al momento de su creación (1996/1997), poseía el periodo más largo (2^{19937}-1).
- Rápida generación de números.
- Uso eficiente de memoria.

Referencias relacionadas con este generador pueden ser encontradas en [3]. El código

del algoritmo se puede encontrar en versiones para lenguaje C/C++ en [4], PASCAL, JAVA, EXCEL (como Add-In), Haskell y FORTRAN. En [5] hay información referente a estas implementaciones. En [4] hay implementaciones tanto para 32-bits como para 64-bits.

También ha habido experiencias en el empleo de este generador en el área de cómputo paralelo. Como referencia se puede citar [6] y [7].

La versión utilizada para la prueba de este reporte, no permite el uso de varios torrentes (streams), sin embargo existen implementaciones en C++ que contemplan esta posibilidad. Como referencia en el marco de una aplicación concreta se puede ver [11].

Instalación y uso

La instalación y el uso del generador no presento ningún inconveniente. En este sentido, el esfuerzo de los autores por lograr la mayor portabilidad posible del código dio buen resultado.

La única precaución que se debe tener es la de no utilizar variables, constantes, etc. nombradas N o M, ya que estos son empleadas por el algoritmo como constantes.

Mecanismos para uso de semillas

El generador se inicializa haciendo una invocación a la función SGENRAND(x). En el mismo código se aclara que x puede ser cualquier entero positivo. No se provee de lista de semillas "buenas".

UNIRAN C

UNIRAN es un generador ideado inicialmente para FORTRAN [8]. La versión empleada para este reporte es la brindada en [9] para el lenguaje C.

El principal motivo por el cual es incluido en esta comparación es el de ser el generador sugerido en [9], una referencia clásica en el área de simulación.

El periodo del generador es de 2^{31}-1 y en la implementación brindada se utilizan bloques de streams de largo 10^5 números aleatorios.

Instalación y uso

La instalación y el uso del generador no presento ningún inconveniente. El código sugerido en [9] fue hecho de acuerdo al ANSI C.

Mecanismos para uso de semillas

La generación de variables aleatorias se realiza mediante la invocación de la función tipo float rand(stream).

El código incluye un array con semillas por defecto que puede ser cambiado mediante la función randst(zset, stream).

UNIRAN C++

Esta es una adaptación del código UNIRAN C para el lenguaje C++ hecha para este reporte. El propósito es comparar los tiempos de ejecución de UNIRAN C y contrastarlos con UNIRAN C++ para comparar como la programación orientada a objetos influye en dichos tiempos.

DRAND48

Este generador es el estándar de SOLARIS y por tanto se utiliza como base de comparación. Según [10] dicho generador no produce números aleatorios con buenas propiedades de aleatoriedad.

El periodo del generador es de 2^{48}.

III) Estudio de tiempos

El experimento consistió en generar hasta 10^8 números aleatorios y registrar los tiempos de ejecución en determinados intervalos. Se empleó la función clock() de la biblioteca time.h de C para el cálculo de dichos tiempos.

Para esto se utilizo un equipo SUN Ultra 5 con SOLARIS 1.3 de 256 MB.

En laTabla 1 se presentan los datos de los tiempos registrados (en segundos) para el experimento junto con el número de iteraciones realizadas. Los valores obtenidos para un número de iteraciones bajas (menor a 10^3).

Iteraciones	DRAND48	MT	UNIRAN C	UNIRAN C++
1000	0.0	0.0	0.0	0.0
10000	0.5	0.0	0.0	0.0
100000	0.4	0.3	0.3	0.3
1000000	4.1	2.9	2.8	3.0
10000000	40.8	28.5	28.4	30.1
100000000	407.2	283.5	284.2	302.8

Todos los tiempos son en segundos

Tabla 1. Tiempos de ejecución registrados (en segundos)

De forma de sintetizar la tabla anterior, calculamos la razón de tiempos de los generados MT y UNIRAN (para las dos versiones) en función de los tiempos de DRAND48.

Ratio MT/DRAND	UC/DRAND	UC++/DRAND
0.06	0.06	0.08
0.69	0.69	0.74
0.70	0.70	0.75
0.70	0.70	0.74
0.70	0.70	0.74

Tabla 2. Relación de tiempos

De la tabla anterior vemos que en cuanto a tiempos, el desempeño de los generadores MT y UNIRAN C son similares. Sin embargo, cuando consideramos el hecho de que el MT tiene un período más largo, concluimos que el MT es superior al UNIRAN C en ese aspecto.

IV) Conclusiones

Se describieron brevemente tres generadores aleatorios (uno de ellos en una versión para lenguaje C y otra C++) para los cuales se realizaron pruebas de velocidad. Dichas pruebas mostraron que el Mersenne Twister (MT) logra la mejor relación longitud de periodo y velocidad. A su vez, la versión UNIRAN C no es significativamente más veloz que UNIRAN C++, por lo cual podrían implementarse generadores mediante programación orientada a objetos sin grandes pérdidas en el desempeño general de la aplicación.

Apéndice A

Código de la implementación en C++ del generador UNIRAN

```
long zrng[] = { 0,1973272912, 281629770, 20006270, 1280689831,
2096730329, 1933576050,913566091, 246780520, 1363774876, 604901985,
1511192140, 1259851944,824064364, 150493284, 242708531, 75253171,
1964472944, 1202299975,233217322, 1911216000, 726370533, 403498145,
993232223, 1103205531,762430696, 1922803170, 1385516923, 76271663,
413682397, 726466604,336157058, 1432650381, 1120463904, 595778810,
877722890, 1046574445,68911991, 2088367019, 748545416, 622401386,
2122378830, 640690903,1774806513, 2132545692, 2079249579, 78130110,
852776735, 1187867272,1351423507, 1645973084, 1997049139, 922510944,
2045512870, 898585771,243649545, 1004818771, 773686062, 403188473,
372279877, 1901633463,498067494, 2087759558, 493157915, 597104727,
1530940798, 1814496276,536444882, 1663153658, 855503735, 67784357,
1432404475, 619691088,119025595, 880802310, 176192644, 1116780070,
277854671, 1366580350,1142483975, 2026948561, 1053920743, 786262391,
1792203830, 149467770,1923011392, 1433700034, 1244184613, 1147297105,
539712780, 1545929719,190641742, 1645390429, 264907697, 620389253,
1502074852, 927711160,364849192, 2049576050, 638580085, 547070247};

class rand {

  public:


          /*metodos*/
          float u(int);
```

```
            void randst(long, int);
            long randgt(int);

            rand();
            ~rand();


};



rand::rand( ){}

rand::~rand(){};

float rand::u(int stream)
{

            #define MODLUS 2147483647
            #define MULT1 24112
            #define MULT2 26143

          long zi, lowprd, hi31;

          zi = zrng[stream];
          lowprd = (zi & 65535) * MULT1;
          hi31 = (zi>>16) * MULT1 + (lowprd >> 16);
          zi = ((hi31 & 32767) << 16) + (hi31 >> 15);
          if (zi< 0) zi += MODLUS;
          lowprd = (zi & 65535) * MULT2;
          hi31 = (zi>> 16) * MULT2 + (lowprd >> 16);
          zi = ((lowprd & 65535) - MODLUS) + ((hi31 & 32767) << 16) +
(hi31>>15);
          if (zi < 0) zi += MODLUS;
          zrng[stream] = zi;
          return((zi >> 7 | 1) +1)/16777216.0;


}

/* set the current zrng for stream "stream" to zset. */

void rand::randst( long zset, int stream)
{
zrng[stream] = zset;
}

/* return the current zrng for stream "stream".*/

long rand::randgt(int stream)
{
          return zrng[stream];
}
```

Referencias

[1] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998)

[2] Mersenne Twister Home Page: What is MT?. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ewhat-is-mt.html (Fecha de consulta 19/7/2005)

[3] Mersenne Twister Home Page: Academic Paper. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/earticles.html(Fecha de consulta 19/7/2005)

[4] Mersenne Twister Home Page: C code http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html (Fecha de consulta 19/7/2005)

[5] Mersenne Twister Home Page: MT in various Languages http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/eversions.html (Fecha de consulta 19/7/2005)

[6] Mersenne Twister Home Page: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/DC/dc.html (Fecha de consulta 19/7/2005)

[7] Makoto Matsumoto and Takuji Nishimura, "Dynamic Creation of Pseudorandom Number Generators", Monte Carlo and Quasi-Monte Carlo Methods pp 56—69 , Springer (1998).

[8] K. Marse, S.D. Roberts, Implementing a Portable FORTRAN Uniform (0,1) Generator, Simulation, 41: 135-139 (1983).

[9] A.M. Law, W.D. Kelton, Simulation Modelling & Analysis 2nd Edition, pp 449-457, McGrawHill (1991).

[10] GNU Scientific Library: Reference Manual http://www.gnu.org/software/gsl/manual/gsl-ref_toc.html#TOC272 (Fecha de consulta 19/7/2005)

[11] EOSimulator Developer Manual v1.0 http://www.fing.edu.uy/inco/cursos/simulacion/paginaEosim/doc/Developer%20manual%20v1.0.pdf (Fecha de consulta 26/8/2005)