# Библиотека PRAND: генерация параллельных потоков случайных чисел для расчетов Монте-Карло с использованием GPU.

# Использование Multi-GPU для многомерного численного интегрирования методом Монте-Карло.

Л.Ю. Бараш
ИТФ РАН

**МГУ, 18 марта 2014 года**

# План доклада

1. Методы генерации случайных чисел
2. Методы генерации параллельных потоков случайных чисел
3. Эффективные алгоритмы пропуска кусков и инициализации независимых последовательностей для выбранных генераторов
4. Библиотека PRAND: однонитевые параллельные алгоритмы генераторов псевдослучайных чисел и их использование
5. Библиотека PRAND: параллельные алгоритмы генераторов псевдослучайных чисел с использованием множества нитей для ускорения вычислений
6. Примеры использования в прикладных задачах
7. Сравнение с другими современными библиотеками по генерации параллельных потоков случайных чисел

Генераторы случайных чисел и их реализации в библиотеках подпрограмм должны удовлетворять ряду существенных требований.

1. Статистическая устойчивость
2. Непредсказуемость
3. Длинный период
4. Эффективность
5. Наличие теории
6. Воспроизводимость
7. Переносимость
8. Пропуск кусков
9. Правильная инициализация

# Основные методы генерации случайных чисел

1. Линейно-конгруэнтные (LCG)
2. Сдвиговый регистр (GFSR)

Примеры современных модификаций методов LCG и GFSR

- генератор Mersenne Twister (Matsumoto, Tishimura, 1998)
- комбинированные LCG-генераторы (L'Ecuyer, 1999)
- комбинированные Tausworthe генераторы (L'Ecuyer, 1996; L'Ecuyer, 1999).

Линейно-конгруэнтные: $x_{n+1} = (ax_n + c)(\mathrm{mod}\ M).$

Сдвиговые регистры: $x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k})(\mathrm{mod}\ 2),$

$$u_n = \sum_{i=1}^{L} x_{ns+i-1} 2^{-i}, \qquad P(z) = z^k - a_1 z^{k-1} - \cdots - a_k.$$

Генератор Mersenne Twister: $\mathbf{x}_{k+n} := \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u | \mathbf{x}_{k+1}^l) A.$

Комбинированный генератор MRG32K3A:

$$x_n = (ax_{n-2} + bx_{n-3})(\mathrm{mod} m_1),$$
$$y_n = (cy_{n-1} + dy_{n-3})(\mathrm{mod} m_2),$$
$$z_n = (x_n + y_n)(\mathrm{mod} m_1).$$

$a = 1403580,\ b = -810728,\ c = 527612,\ d = -1370589,$

$m_1 = 2^{32} - 209,\ m_2 = 2^{32} - 22853$

Комбинированный Tausworthe-генератор LFSR113 является комбинацией четырех сдвиговых регистров:

- $x_n = x_{n-31} \oplus x_{n-25}, \quad s = 18,$
- $x_n = x_{n-29} \oplus x_{n-27}, \quad s = 2,$
- $x_n = x_{n-28} \oplus x_{n-15}, \quad s = 7,$
- $x_n = x_{n-25} \oplus x_{n-22}, \quad s = 13.$

# Метод генерации случайных чисел, основанный на параллельной эволюции автоморфизмов тора

L.Yu. B., Europhysics Letters (EPL) 95, 10003 (2011).
L.Yu. B., L.N. Shchur, Computer Physics Communications, 182 (7), 1518-1527 (2011).
L.Yu. B., L.N. Shchur, Phys.Rev. E 73 , 036701 (2006).

**Основные черты метода:**

**Ансамбль преобразований:** используется ансамбль MRG-преобразований вместо отдельной системы

**Скрытые переменные:** только часть информации идет в выходную последовательность генератора, что помогает подавлять корелляции, усложняет расшифровку и приводит к другим свойствам.

**Длина периода:** Период равен $p^2 - 1$ для сетки $p \times p$, где $p$ - простое число. Период равен $3 \cdot 2^{m-2}$ для сетки $2^m \times 2^m$. Период может быть настолько большим, насколько это требуется.

# Описание метода: конструирование генератора

Множество состояний: $R = L^s$, где $L = \{0, 1, \ldots, g-1\} \times \{0, 1, \ldots, g-1\}$

На практике, $g = 2^t$ или $g = p$ или $g = p \cdot 2^t$, где $p$ – простое число.

Функция перехода генератора определяется действием преобразования

$$\begin{pmatrix} x_i^{(n)} \\ y_i^{(n)} \end{pmatrix} = M \begin{pmatrix} x_i^{(n-1)} \\ y_i^{(n-1)} \end{pmatrix} \mod g,$$

где $s$ точек $(i = 0, 1, \ldots, s-1)$ преобразуются на каждом шаге.

Эквивалентное описание при помощи рекуррентного соотношения:

$$x^{(n)} = kx^{(n-1)} - qx^{(n-2)} (\mathrm{mod}\ g)$$
$$y^{(n)} = ky^{(n-1)} - qy^{(n-2)} (\mathrm{mod}\ g)$$

Здесь $k = \mathrm{Tr}\ (\mathrm{M}),\ \mathrm{q} = \det\ \mathrm{M}$

Характеристический полином рекуррентного соотношения:

$$f(x) = x^2 - kx + q.$$

# Описание метода: конструирование генератора

Пусть $\alpha_i^{(n)}$ обозначает старший бит $x_i^{(n)}$: $\alpha_i^{(n)} = \lfloor 2x_i^{(n)}/g \rfloor$.

Выходная функция генератора $G : L^s \to \{0, 1, \ldots, 2^s - 1\}$

определена следующим образом: $a_n = \sum_{i=0}^{s-1} \alpha_i^{(n)} \cdot 2^i$.

Другими словами, $a_n -$ это s-битовое число, состоящее из битов $\alpha_0^{(n)}, \alpha_1^{(n)}, \ldots, \alpha_{s-1}^{(n)}$. В случае $g = 2^m$, $a_n$ состоит в точности из старших битов целых чисел $x_0^{(n)}, x_1^{(n)}, \ldots, x_{s-1}^{(n)}$.

Сконструированный генератор имеет много скрытой информации. Именно, $s(m-1)$ битов каждого состояния является скрытой информацией: это те биты, которые не вовлечены в конструирование наблюдаемого числа $a_n$.

# PRNGs based on ensemble of dynamical systems and their properties:

The state of the generator consists of the values $x_i^{(n-1)}, x_i^{(n-2)} \in \{0, 1, \ldots, g-1\}$, $i = 0, 1, \ldots, s-1$. The transition function of the generator is defined by the recurrence relation

$$x_i^{(n)} = kx_i^{(n-1)} - qx_i^{(n-2)} \pmod{g}, \tag{1}$$

where $i = 0, 1, \ldots, s-1$. The values $x_i^{(n)}$, $i = 0, 1, \ldots, s-1$ can be considered as $x$-coordinates of $s$ points $(x_i^{(n)}, y_i^{(n)})^T$, $i = 0, 1, \ldots, s-1$ of the $g \times g$ lattice on the two-dimensional torus, then each recurrence relation describes the dynamics of $x$-coordinate of a point on the two-dimensional torus:

$$\begin{pmatrix} x_i^{(n)} \\ y_i^{(n)} \end{pmatrix} = M \begin{pmatrix} x_i^{(n-1)} \\ y_i^{(n-1)} \end{pmatrix} \pmod{g}, \tag{2}$$

where matrix $M = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix}$ is a matrix with integer elements, $k = \operatorname{Tr} M$, $q = \det M$ and $\operatorname{Tr} M$ is a trace of matrix $M$.

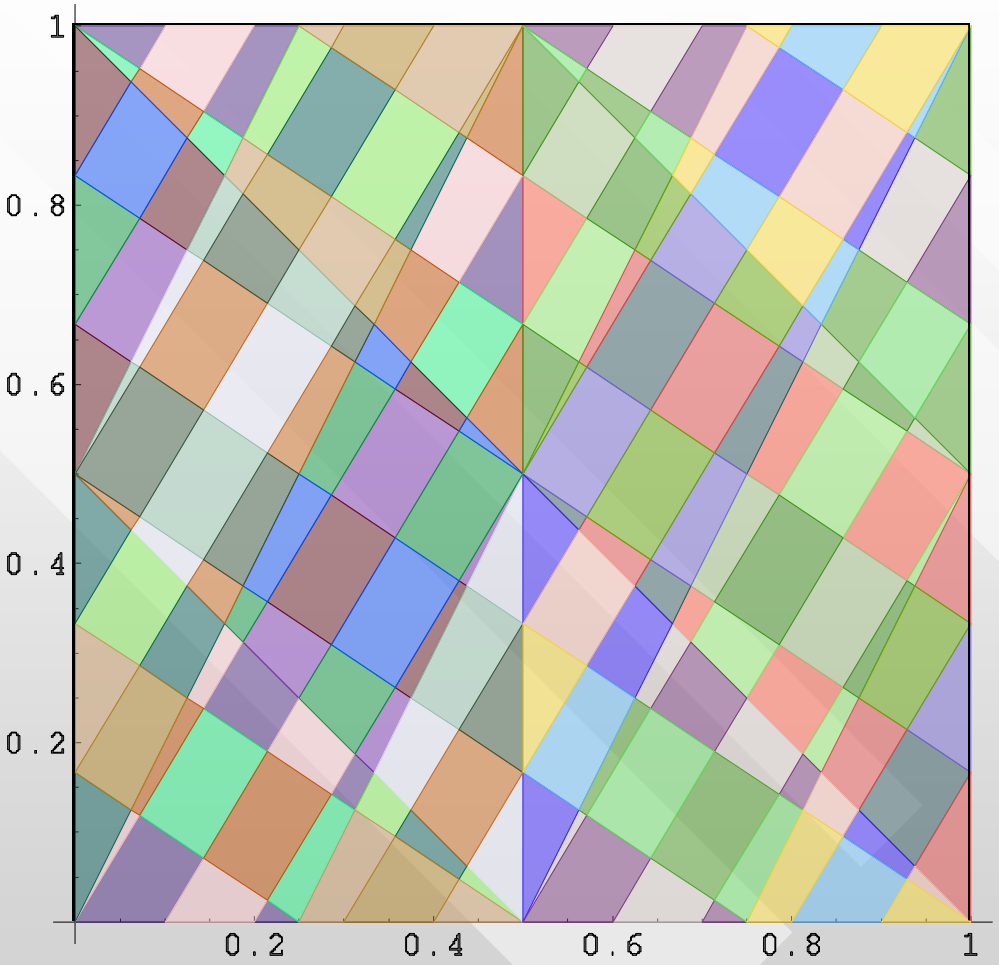
The output function:

$$a^{(n)} = \sum_{i=0}^{s-1} \lfloor 2^v x_i^{(n)} / g \rfloor \cdot 2^{iv}, \tag{3}$$

where $v$ bits are taken from each recurrence and $i = 0, 1, \ldots, s-1$.

The sequence of bits $\{\lfloor 2^v x_i^{(n)} / g \rfloor\}$, where $i$ is fixed and $\{x_i^{(n)}\}$ is generated with relation (2) is designated as a stream of $v$-bit blocks generated with matrix $M$.

# Области на торе и пятибитовые последовательности генерируемые гиперболическим автоморфизмом




L.Yu. B., Europhysics Letters (EPL) 95, 10003 (2011).
L.Yu. B., L.N. Shchur, Phys.Rev. E 73 , 036701 (2006).

## Equidistibution properties of the PRNGs based on ensemble of dynamical systems:

**Proposition 1.** If (i) $M = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix}$ is a matrix with integer values $m_1$, $m_2$, $m_3$, $m_4$, (ii) $m_1$, $q = \det M$ and $g$ are divisible by $2^v$, (iii) the image of the lattice $g \times g$ with the transformation $M^j$ is invariant with respect to the shift $S$ for $j = 0, 1, \ldots, n$, then all the sequences of length $n$ in a stream of $v$-bit blocks generated with matrix $M$ are equiprobable.

**Proposition 2.** Consider a matrix $M$ with integer elements and the following integer quantities: $g = p \cdot 2^t$, $q = \det M = 2^u w \mod g$, $k = \mathrm{Tr}\, M = 2^m r \mod g$, $u \geq 1$, $t \geq v$, $m \geq 0$. Here $w, r$ are odd integers and $p$ is an odd prime. Then all $2^j$ sequences of length $j$ in a stream of $v$-bit blocks generated with recurrence relation (1) are equiprobable for $j = 1, 2, \ldots, \ell$. Here $\ell = \lceil (t - v)/\lceil u/2 \rceil \rceil$ for $u \leq 2m$ and $\ell = \lceil (t - v)/(u - m) \rceil$ for $u > 2m$;


L.Yu. B., *Applying dissipative dynamical systems to pseudorandom number generation: Equidistribution property and statistical independence of bits at distances up to logarithm of mesh size,* Europhysics Letters (EPL) 95, 10003 (2011).

L.Yu. B., *Geometric and statistical properties of pseudorandom number generators based on multiple recursive transformations*, Springer Proceedings in Mathematics and Statistics, Volume 23, p. 265-280 (2012).
[book chapter in “Monte Carlo and Quasi-Monte Carlo Methods 2010”, Ed. by Henryk Wozniakowski and Leszek Plaskota, Springer-Verlag, 2012. xii,732 pp, ISBN 978-3-642-27439-8].

# Параметры генераторов.

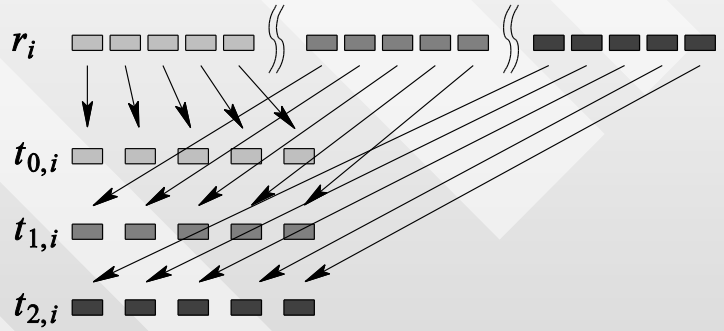
| Generator | $k$ | $q$ | $g$ | $v$ | Period length | Dimension of approx. equidistribution |
|---|---|---|---|---|---|---|
| GM19 | 15 | 28 | $2^{19}-1$ | 1 | $2.7 \cdot 10^{11}$ | 129 |
| GM29 | 4 | 2 | $2^{29}-3$ | 1 | $2.8 \cdot 10^{17}$ | 200 |
| GM31 | 11 | 14 | $2^{31}-1$ | 1 | $4.6 \cdot 10^{18}$ | 210 |
| GM55 | 256 | 176 | $16(2^{51}-129)$ | 4 | $\geq 5.1 \cdot 10^{30}$ | 350 |
| GM61 | 24 | 74 | $2^{61}-1$ | 1 | $5.3 \cdot 10^{36}$ | 415 |
| GQ58.1 | 8 | 48 | $2^{29}(2^{29}-3)$ | 1 | $\geq 2.8 \cdot 10^{17}$ | 200 |
| GQ58.3 | 8 | 48 | $2^{29}(2^{29}-3)$ | 3 | $\geq 2.8 \cdot 10^{17}$ | 200 |
| GQ58.4 | 8 | 48 | $2^{29}(2^{29}-3)$ | 4 | $\geq 2.8 \cdot 10^{17}$ | 200 |
| LFSR113 | – | – | – | – | $1.0 \cdot 10^{34}$ | 30 |
| MRG32K3A | – | – | – | – | $3.1 \cdot 10^{57}$ | 45 |
| MT19937 | – | – | – | – | $4.3 \cdot 10^{6001}$ | 623 |

Результаты статистического тестирования при помощи батарей тестов TestU01.

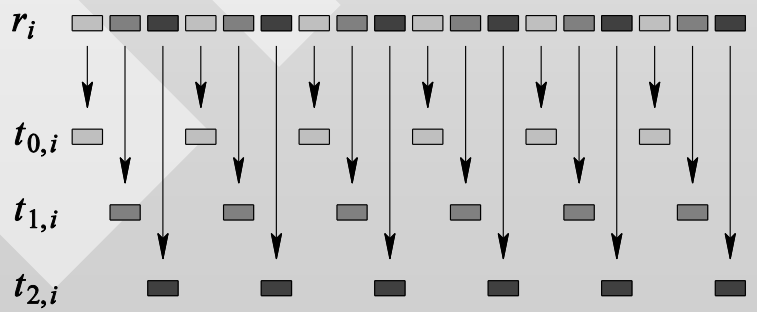
| Generator | SmallCrush | Diehard | Crush | Bigcrush |
|---|---|---|---|---|
| MRG32k3a | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 |
| LFSR113 | 0, 0, 0 | 1, 0, 0 | 6, 6, 6 | 6, 6, 6 |
| MT19937 | 0, 0, 0 | 0, 0, 0 | 2, 2, 2 | 2, 2, 2 |
| GM29-SSE | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 |
| GM55.4-SSE | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 |
| GQ58.1-SSE | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 |
| GQ58.3-SSE | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 |
| GQ58.4-SSE | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 | 0, 0, 0 |

# Методы генерации параллельных потоков случайных чисел

1. Случайный выбор начальных величин
2. Параметризация
3. Расщепление блока




4. Чехарда



# Эффективные алгоритмы пропуска кусков и инициализации последовательностей для выбранных генераторов

1. Пропуск кусков для генераторов GM19, GM31, GM61, GM29.1, GM55.4, GQ58.1, GQ58.3, GQ58.4 , в которых состояние изменяется по правилу $x^{(n)} = kx^{(n-1)} - qx^{(n-2)} (\mathrm{mod}\ g)$.

   Можно показать, что справедливо $x^{(2n)} = k_n x^{(n)} - q_n x^{(0)} (\mathrm{mod}\ g)$, где

$$k_1 = k; \quad q_1 = q; \quad k_{2n} = k_n^2 - 2q_n (\mathrm{mod}\ g); \quad q_{2n} = q_n^2 (\mathrm{mod}\ g).$$

Это позволяет быстро вычислить коэффициенты, необходимые для пропуска кусков с количеством элементов, равным степени двойки. Эффективным способом пропуска куска произвольной длины $n$ является разложение числа $n$ в двоичную запись вида $n = 2^{i_0} + 2^{i_1} + \cdots + 2^{i_m}$ и последовательный пропуск кусков длин $2^{i_0}, \ldots, 2^{i_m}$. Также можно показать, что

$$k_0 = 2; \quad k_1 = k; \quad k_{n+1} = kk_n - qk_{n-1} (\mathrm{mod}\ g); \quad q_n = q^n (\mathrm{mod}\ g).$$

## 2. Пропуск кусков для генератора MRG32K3A.

Удобно обозначить $\mathbf{X}_n = \begin{pmatrix} x_n \\ x_{n-1} \\ x_{n-2} \end{pmatrix}$, $\mathbf{Y}_n = \begin{pmatrix} y_n \\ y_{n-1} \\ y_{n-2} \end{pmatrix}$, $A = \begin{pmatrix} 0 & a & b \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} c & 0 & d \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$.

Это позволяет представить правило изменения состояния MRG32K3A в виде

$$\begin{aligned} \mathbf{X}_{n+1} &= A\,\mathbf{X}_n (\mathrm{mod}\ m_1), \\ \mathbf{Y}_{n+1} &= B\,\mathbf{Y}_n (\mathrm{mod}\ m_2). \end{aligned}$$

Состоянием генератора является пара векторов $\mathbf{X}_n$, $\mathbf{Y}_n$. Следовательно, для пропуска кусков длиной $n$ необходимо найти $n$-ую степень матриц $A$ и $B$.

Разлагаем число $n$ в двоичную запись вида $n = 2^{i_0} + 2^{i_1} + \cdots + 2^{i_m}$, тогда $A^n = A^{2^{i_0}} A^{2^{i_1}} \ldots A^{2^{i_m}} (\mathrm{mod}\ m_1)$, $B^n = B^{2^{i_0}} B^{2^{i_1}} \ldots B^{2^{i_m}} (\mathrm{mod}\ m_2)$, т.е. пропуск куска длины $n$ может быть произведен за $O(\log n)$ операций.

Функция `MRG32K3A_init_sequence` позволяет инициализировать до $10^{19}$ независимых параллельных последовательностей случайных чисел длины, не превосходящей $10^{38}$.

## 3. Пропуск кусков для генератора LFSR113.

Алгоритм пропуска куска для комбинированного генератора сводится к пропуску кусков для каждого из этих сдвиговых регистров.

- $x_n = x_{n-31} \oplus x_{n-25}, \quad s = 18,$
- $x_n = x_{n-29} \oplus x_{n-27}, \quad s = 2,$
- $x_n = x_{n-28} \oplus x_{n-15}, \quad s = 7,$
- $x_n = x_{n-25} \oplus x_{n-22}, \quad s = 13.$

$$x_n = (x_{n-p} + x_{n-p+q}) (\mathrm{mod}\ 2), \qquad u_n = \sum_{l=1}^{32} x_{is+l-1} 2^{-l}$$

Тогда справедливо $x_n = (x_{n-2^e p} + x_{n-2^e p+q}) (\mathrm{mod}\ 2)$.

Функция `LFSR113_init_sequence` позволяет инициализировать $4 \cdot 10^{18}$ независимых последовательностей случайных чисел длины, не превосходящей $10^{10}$; функция `LFSR113_init_long_sequence` позволяет инициализировать $4 \cdot 10^9$ независимых последовательностей случайных чисел длины, не превосходящей $3 \cdot 10^{25}$.

*Jumping ahead for the generator LFSR113.*

Algorithm to jump ahead for the combined generator reduces to jumping ahead for every particular shift register sequence.

$$x_n = (x_{n-p} + x_{n-p+q}) (\mathrm{mod}\ 2), \qquad u_n = \sum_{l=1}^{32} x_{is+l-1} 2^{-l} \qquad (5)$$

Then the following holds: $x_n = (x_{n-2^e p} + x_{n-2^e p+2^e q}) (\mathrm{mod}\ 2)$.

Suppose we have the initial values $x_0, x_1, \ldots, x_{31}$. Let's build a table of values, which rows are numerated starting with zero. The zero row of the table contains $p$ values: $x_p, x_{p+1}, \ldots, x_{2p-1}$, which are directly caclulated from the initial values. The $n$-th row of the table contain the following $p$ values: $x_{2^n p}, x_{2^n (p+1)}, \ldots, x_{2^n (2p-1)}$. Each value of the table is calculated from the preivous values with the relation $x_{2^n (p+l)} = x_{2^n l} + x_{2^n (l+q)}$. We note that for $n \geq 5$ the table contains $x_{s \cdot 2^n}$. Indeed, it follows from $p > 16$ that $s < 2p$. If $j \in \{0, 1, 2, 3, 4, 5\}$ is maximal integer such that $s \cdot 2^j < 2p$, then $s \cdot 2^j \in \{p, p+1, \ldots, 2p-1\}$, therefore, $x_{s \cdot 2^n}$ belongs to $(n-j)$-th row of the table. In order to jump ahead a block of length $2^n$, one needs to calculate the 32 bits $x_{s \cdot 2^n}, x_{s \cdot 2^n + 1}, \ldots, x_{s \cdot 2^n + 31}$, which are involved in $u_{2^n}$ in (5) for each of the four shift registers of LFSR113. To this purpose, one constructs 128 tables which start from each of the 32 initial bits for each of the four shift registers. The realization of this operation for GPU is much more efficient than for CPU because the 128 tables can be constructed independently using different threads. In order to efficiently jump ahead $n$ pseudorandom numbers for arbitrary $n$ one can subsequently skip ahead blocks of length $2^{i_0}, 2^{i_1}, \ldots, 2^{i_m}$, where the binary notation of value $n$ is $n = 2^{i_0} + 2^{i_1} + \cdots + 2^{i_m}$.

## 4. Пропуск кусков для генератора MT19937.

Алгоритм генератора MT19937 имеет линейную структуру и может быть записан в виде $\mathbf{Y}_{n+1} = A\mathbf{Y}_n \pmod 2$, где $\mathbf{Y}_n$ – состояние генератора, матрица $A$ имеет размер $19937 \times 19937$, вычисление $A^n$ будет исключительно медленным и потребует много памяти.

Другой способ [*]: для любого $v \in \mathbb{N}$ имеет место выражение

$$A^v\mathbf{Y}_0 = g_v(A)\mathbf{Y}_0 = a_k\mathbf{Y}_{k-1} + a_{k-1}\mathbf{Y}_{k-2} + \cdots + a_2\mathbf{Y}_1 + a_1\mathbf{Y}_0, \qquad (1)$$

где $k = 19937$, коэффициенты $a_i \in \{0, 1\}, i = 1, \ldots, k$, и полином $g_v(x) = a_k x^{k-1} + \cdots + a_2 x + a_1$ в поле $\mathbb{F}_2$ зависит от $v$. В работе [*] приведен способ вычисления полинома $g_v$ для произвольного фиксированного $v$.

В отличие от методов, использующихся в других существующих пакетах, в PRAND коэффициенты вычисляются заранее для следующих значений $v$: $v = 2^n$, $n = 0, 1, \ldots, 511$ и $v = n \cdot 2^i$, $n = 15, 16, \ldots, 24$, $i = 0, 1, \ldots, 127$.

Вычисление (1) можно проводить используя массивный параллелизм графического процессора для ускорения вычислений.

[*] Haramoto et.al., INFORMS Journal on Computing 20 (3), 385-390 (2008).

Table 2: Initialization of pseudorandom streams for RNGs in PRAND library

| Function initializing sequence | Number of sequences | Maximal length |
|---|---|---|
| `gm19_init_sequence_` | $1000$ | $6 \cdot 10^6$ |
| `gm29_init_short_sequence_` | $10^8$ | $8 \cdot 10^7$ |
| `gm29_init_medium_sequence_` | $10^6$ | $8 \cdot 10^9$ |
| `gm29_init_long_sequence_` | $10^4$ | $8 \cdot 10^{11}$ |
| `gm31_init_short_sequence_` | $10^9$ | $8 \cdot 10^7$ |
| `gm31_init_medium_sequence_` | $10^7$ | $8 \cdot 10^9$ |
| `gm31_init_long_sequence_` | $10^5$ | $8 \cdot 10^{11}$ |
| `gm55_init_short_sequence_` | $10^{18}$ | $10^{10}$ |
| `gm55_init_long_sequence_` | $4 \cdot 10^9$ | $10^{20}$ |
| `gq58x1_init_short_sequence_` | $10^8$ | $8 \cdot 10^7$ |
| `gq58x1_init_medium_sequence_` | $10^6$ | $8 \cdot 10^9$ |
| `gq58x1_init_long_sequence_` | $10^4$ | $8 \cdot 10^{11}$ |
| `gq58x3_init_short_sequence_` | $2 \cdot 10^8$ | $8 \cdot 10^7$ |
| `gq58x3_init_medium_sequence_` | $2 \cdot 10^6$ | $8 \cdot 10^9$ |
| `gq58x3_init_long_sequence_` | $2 \cdot 10^4$ | $8 \cdot 10^{11}$ |
| `gq58x4_init_short_sequence_` | $3 \cdot 10^8$ | $8 \cdot 10^7$ |
| `gq58x4_init_medium_sequence_` | $3 \cdot 10^6$ | $8 \cdot 10^9$ |
| `gq58x4_init_long_sequence_` | $3 \cdot 10^4$ | $8 \cdot 10^{11}$ |
| `gm61_init_sequence_` | $1.8 \cdot 10^{19}$ | $10^{10}$ |
| `gm61_init_long_sequence_` | $4 \cdot 10^9$ | $3 \cdot 10^{25}$ |
| `lfsr113_init_sequence_` | $3.8 \cdot 10^{18}$ | $10^{10}$ |
| `lfsr113_init_long_sequence_` | $4 \cdot 10^9$ | $10^{24}$ |
| `mrg32k3a_init_sequence_` | $10^{19}$ | $10^{38}$ |
| `mt19937_init_sequence_` | $10^{19}$ | $10^{130}$ |

Table 3: Multi-threaded generation of pseudorandom numbers with GPGPU.

| Generator | Number of blocks | Number of threads per block | Number of threads per array section | Number of array sections |
|---|---|---|---|---|
| GM19 | 512 | 128 | 32 | 2048 |
| GM29 | 512 | 128 | 32 | 2048 |
| GM31 | 512 | 128 | 32 | 2048 |
| GM55 | 128 | 128 | 8 | 2048 |
| GM61 | 512 | 128 | 32 | 2048 |
| GQ58.1 | 512 | 128 | 32 | 2048 |
| GQ58.3 | 128 | 192 | 12 | 2048 |
| GQ58.4 | 128 | 128 | 8 | 2048 |
| LFSR113 | 512 | 128 | 128 | 512 |
| MRG32K3A | 64 | 1024 | 1 | 65536 |
| MT19937 | 64 | 227 | 227 | 64 |

Table 5: Comparing performance of PRAND, NAG Numerical routines for GPUs and Nvidia cuRand. The task fills $100$ times an array of $N = 2^{29}$ double-precision floating point numbers in GPU memory with pseudorandom numbers. CPU: Intel Xeon E5630; GPU: Nvidia Tesla X2070; OS: CentOS 6.1; Compiler: CUDA 5.0; Optimization: -O2.

| Library | Time (sec) MRG32K3A | Time (sec) MT19937/MTGP32 |
|---|---|---|
| CURAND | 9.6 | 16.3 |
| NAG | 10.5 | 13.7 |
| PRAND | 14.5 | 13.4 |

Table 4: Speed of generation for different realizations. CPU: Intel Xeon X5670; GPU: nVidia Fermi C2050; OS: Linux SLES 11 SP1 / CentOS 5.5; Compiler: CUDA 5.0; Optimization: -O2.

| Generator | CPU, ANSI C (numbers/sec) | CPU, SSE (numbers/sec) | GPU, single-threaded (numbers/sec) | GPU, multithreaded (numbers/sec) |
|---|---|---|---|---|
| GM19 | $7.3 \cdot 10^6$ | $3.3 \cdot 10^7$ | $1.6 \cdot 10^5$ | $2.6 \cdot 10^8$ |
| GM29 | $8.5 \cdot 10^6$ | $3.6 \cdot 10^7$ | $2.1 \cdot 10^5$ | $2.5 \cdot 10^8$ |
| GM31 | $9.1 \cdot 10^6$ | $3.1 \cdot 10^7$ | $7.3 \cdot 10^4$ | $2.1 \cdot 10^8$ |
| GM55 | $2.1 \cdot 10^7$ | $4.6 \cdot 10^7$ | $1.9 \cdot 10^5$ | $5.6 \cdot 10^8$ |
| GM61 | $4.2 \cdot 10^6$ | $8.9 \cdot 10^6$ | $1.5 \cdot 10^4$ | $1.6 \cdot 10^8$ |
| GQ58.1 | $7.8 \cdot 10^6$ | $1.3 \cdot 10^7$ | $2.2 \cdot 10^4$ | $1.9 \cdot 10^8$ |
| GQ58.3 | $1.5 \cdot 10^7$ | $2.4 \cdot 10^7$ | $1.3 \cdot 10^5$ | $3.8 \cdot 10^8$ |
| GQ58.4 | $2.1 \cdot 10^7$ | $4.0 \cdot 10^7$ | $1.9 \cdot 10^5$ | $5.4 \cdot 10^8$ |
| LFSR113 | $2.2 \cdot 10^8$ | $1.6 \cdot 10^8$ | $1.2 \cdot 10^6$ | $1.3 \cdot 10^8$ |
| MRG32K3A | $5.0 \cdot 10^7$ | $1.5 \cdot 10^8$ | $7.3 \cdot 10^5$ | $2.1 \cdot 10^9$ |
| MT19937 | $2.0 \cdot 10^8$ | $2.5 \cdot 10^8$ | $6.1 \cdot 10^5$ | $2.9 \cdot 10^9$ |

# Параллельные алгоритмы для GM19,GM31,GM61,GM55.4,GQ58.1,GQ58.3,GQ58.4

## Алгоритм «один генератор на $s$ нитей»

Каждая нить вычисляет величины $x_i^{(n)} = kx_i^{(n-1)} - qx_i^{(n-2)} (\mathrm{mod}\ g)$ и $a_i^{(n)} = \lfloor 2^v x_i^{(n)}/g \rfloor \cdot 2^{iv}$ для одного из значений $i = 0, 1, \ldots, s-1$. Затем происходит синхронизация нитей блока, ожидание завершения всех вычислений. После этого каждая нить, номер которой делится на $s$, складывает вычисленные значения $a_i^{(n)}$ и получает очередное выходное значение $a^{(n)} = \sum_{i=0}^{s-1} a_i^{(n)}$. Каждые $s$ нитей генерируют одну последовательность заданной заранее длины `length`. Для каждого набора из $s$ нитей имеются свои начальные условия $\left(x_0^{(i)}, x_1^{(i)}\right)$, инициализация происходит таким образом, чтобы на выходе была выходная последовательность длины `length*N/s`, где `N` – общее число запускаемых нитей. Для этого, прежде чем начать генерацию последовательности чисел, каждая нить производит соответствующий пропуск куска последовательности. При использовании данной реализации необходимо, чтобы число нитей запускаемых в каждом блоке делилось нацело на $s$.

## Параллельный алгоритм для MT19937: один генератор на 227 нитей.

$$\mathbf{x}_{k+n} := \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u | \mathbf{x}_{k+1}^l) A.$$

Алгоритм может быть существенно ускорен, если параллельно использовать $n - m = 227$ нитей для обновления состояния генератора. Действительно, чтобы вычислить величину $x_{k+n}$ требуется знать значения величин $x_k, x_{k+1}, x_{k+m}$, поэтому вычисления $x_n, x_{n+1}, \ldots, x_{2n-m-1}$ можно проводить независимо, зная значения величин $x_0, x_1, \ldots, x_{n-1}$, а для вычисления величины $x_{2n-m}$ уже потребуется вычисленное значение величины $x_n$.

Для дальнейшего ускорения генерации случайных чисел MT19937, мы будем использовать разные наборы из 227 нитей для заполнения разных участков из массива случайных чисел. Для этого перед началом вычислений каждый набор нитей производит пропуск куска, чтобы перейти к началу своего участка. Размер участка массива, заполняемого одним набором из 227 нитей, мы подбираем экспериментально таким образом, чтобы время, затрачиваемое набором нитей на пропуск куска, составляло лишь небольшую часть времени, затрачиваемого на генерацию случайных чисел.

# Примеры использования библиотеки PRAND в прикладных задачах, для которых применение гибридных систем приносит значительный экономический эффект

1. Задача о построении двумерных структур роста в рамках модели агрегации, ограниченной диффузией
2. Алгоритмы многомерного численного интегрирования
3. Моделирование магнитных систем при помощи спиновых моделей

# Другие современные библиотеки по генерации параллельных потоков случайных чисел. Сравнение.

1. Tina's Random Number Generation Library (TRNG), (Universität Magdeburg, Germany)

Из представленных в библиотеке генераторов пропуск кусков и инициализация параллельных потоков имеются только для линейно-конгруэнтных генераторов, для генераторов вида MRG и для генераторов вида YARN. Это недостаточно надежные генераторы случайных чисел.

2. NAG Numerical Routines for GPU, (Oxford, UK) выпущена в конце 2011 года.

В библиотеке представлены параллельные алгоритмы для генераторов MRG32K3A и MT19937. Библиотека является коммерческой, исходные коды недоступны.

3. cuRAND library, NVIDIA CUDA Toolkit 4.1, актуальная версия выпущена в феврале 2012 года.

В библиотеке представлены генераторы MTGP Mersenne Twister, MRG32K3A, XORWOW. Для MTGP Mersenne Twister инициализация проводится методом параметризации, что не является достаточно обоснованным методом. Исходные коды недоступны.

# Другие библиотеки по генерации случайных чисел

*GNU Scientific Library* http://www.gnu.org/software/gsl/
18 стандартных генераторов случайных чисел, включая и старые и некоторые Современные, реализовано стандартно на Си. Без распараллеливания

*Intel MKL Library* http://software.intel.com/en-us/articles/intel-mkl/
Около 7 стандартных генераторов реализованных для CPU эффективно при помощи SIMD , т.е. SSE-команд и 128-битных XMM-регистров. Без распараллеливания.

*RNGSSELIB* L.Yu. B., L.N. Shchur, Computer Physics Communications, 182 (7), 1518-1527 (2011).

6 современных генераторов. Реализовано для CPU при помощи SIMD , т.е. SSE-команд и 128-битных XMM-регистров. Еще эффективнее ,чем Intel MKL. Без распараллеливания.

*SPRNG* http://sprng.cs.fsu.edu/
Стандартные генераторы (старые). Распараллелены методом параметризации, без особой теории.
(Florida State University, США)

*Other existing software for random number generation, which have been reviewed and compared in Comp.Phys.Comm. 185(2014) 1343:*

- *GNU Scientific Library.* Parallel streams, SSE and GPU are not supported.
- *Intel Math Kernel Library.* Parallel streams – parametrization method. SSE is supported. GPU is not supported.
- *SPRNG and GASPRNG.* Parallel streams – parametrization method. SSE is not supported. GPU is supported
- *Tina's random number generator library (TRNG).* Parallel streams – block splitting method, only for six generators. SSE is not supported. GPU is supported for six generators of MRG type.
- *NAG Numerical routines for GPUs.* Parallel streams – block splitting method. SSE is not supported. GPU is supported. Source codes are unavaliable.
- *Nvidia cuRand library.* Parallel streams – parametrization method. SSE is not supported. GPU is supported. Source codes are unavaliable.

# Выводы

1. Разработаны библиотека PRAND и библиотека RNGSSELIB по параллельной генерации псевдослучайных чисел, которая включает генераторы, основанные на параллельной эволюции автоморфизмов тора (GM19, GM31, GM61, GM29.1, GM55.4, GQ58.1, GQ58.3, GQ58.4), генератор MRG32K3A, генератор LFSR113, генератор MT19937.

2. Для каждого из генераторов реализованы:

- инициализация параллельных потоков методом расщепления блока; имеется возможность инициализировать до 10^19 независимых потоков
- эффективные версии для CPU с использованием SIMD-параллелизма и SSE-команд;
- однонитевые версии, которые можно использовать в вычислениях Монте-Карло на графических процессорах, распределенных по нитям и вычислительным узлам произвольным образом;
- параллельные версии с использованием множества нитей графического процессора для ускорения вычислений.
- совместимость с фортраном; имеются примеры использования на Фортране и на Си.

Ошибка вычисления интеграла для простого метода Монте-Карло: $\frac{\sigma}{\sqrt{n}}$

$$\sigma^2 = \langle f^2 \rangle - \langle f \rangle^2, \qquad \text{где} \qquad \langle f^2 \rangle = \frac{1}{n} \sum_{i=1}^{n} f(x_i)^2.$$

Методы уменьшения дисперсии:

1. Выборка по значимости:

$$p(x) = \frac{|f(x)|}{\int |f(x)| dx}, \quad F = \int_R \left( \frac{f(x)}{p(x)} \right) p(x) dx,$$

$$p(u_1, \ldots, u_d) = p_1(u_1) p_2(u_2) \ldots p_d(u_d).$$

2. Стратифицированная выборка: итерационный метод с разбиением на гиперкубы и с количеством точек пропорциональным дисперсии в гиперкубе для уменьшения дисперсии полного интеграла от итерации к итерации.

# Схема вычислений

1. Инициализация параметров
2. Генерация N пробных точек, каждая из которых является вектором из k псевдослучайных чисел, если размерность равна k.
3. Вычисление подынтегральной функции на множестве пробных точек.
4. Суммирование значений подынтегральной функции и их квадратов для всей области интегрирования, а также в каждом гиперкубе. Вычисление средних величин и дисперсий.
5. Оптимизация сетки и других параметров после вычисления значений.
6. Повторение шагов 2–5 вплоть до M итераций

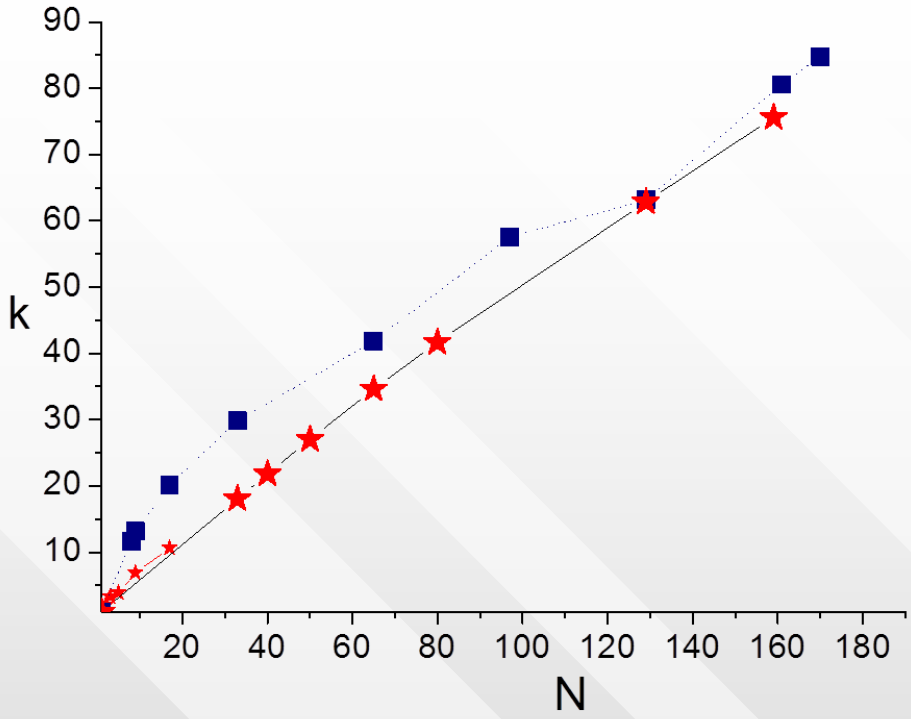



Рисунок 1. Зависимость эффективности параллелизации k = T(1)/T(N) от N, где T(N) – время вычисления многомерного интеграла с заданной точностью (не зависящей от N) при запуске вычисления в режиме N mpi-узлов. Звезды обозначают вычисления на суперкомпьютере «Ломоносов» Московского государственного университета. Квадраты обозначают вычисления на суперкомпьютере «К-100» Института прикладной математики им. М.В. Келдыша Российской академии наук. Программное обеспечение было разработано при помощи технологий MPI и CUDA.

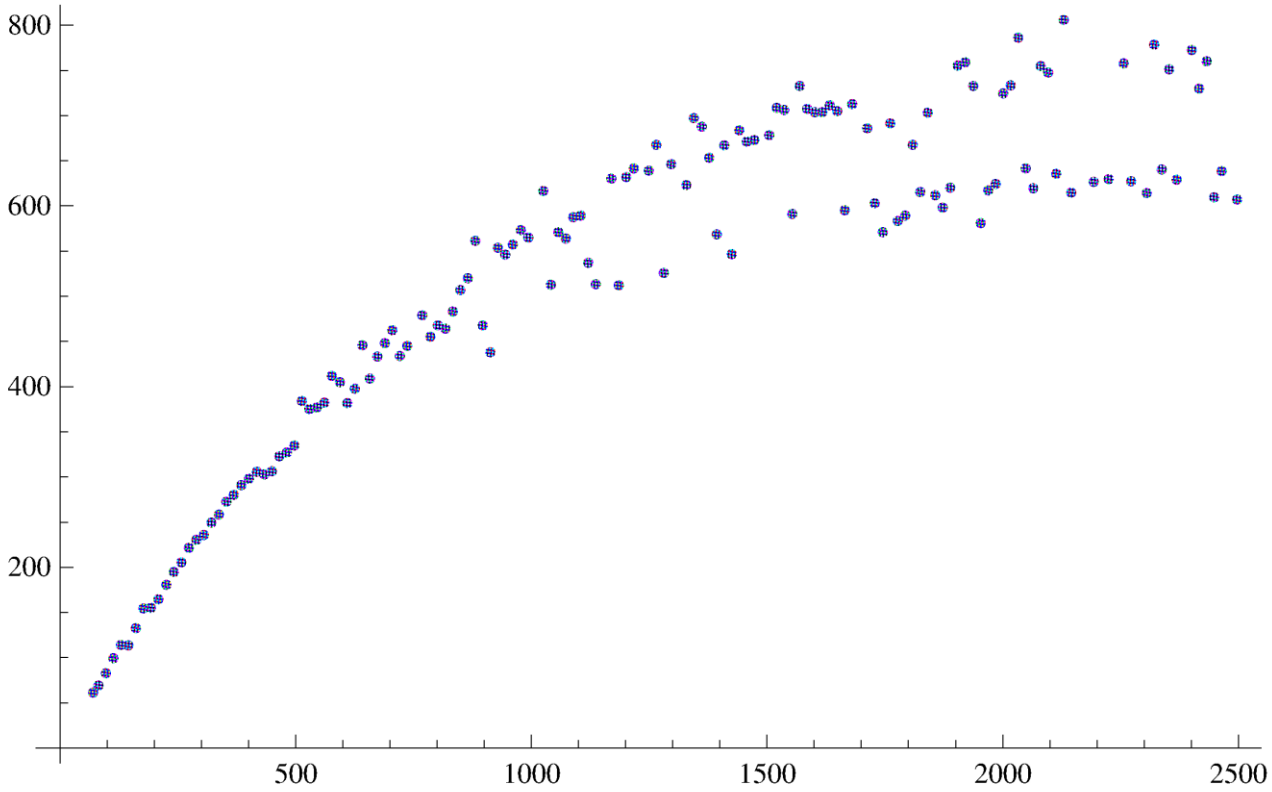



Рисунок 2. Зависимость эффективности параллелизации k = T(1)/T(N) от N, где T(N) – время вычисления многомерного интеграла с заданной точностью (не зависящей от N) при запуске вычисления в режиме N mpi-узлов. Вычисления были проведены на суперкомпьютере «Ломоносов» Московского государственного университета. Программное обеспечение было разработано при помощи технологий MPI и CUDA.

# Спасибо за внимание!

**Литература**


1. T. Hahn, *CUBA — a library for multidimensional numerical integration,* Computer Physics Communications 176 (2007) 712–713.
2. J. Kanzaki, *Monte Carlo Integration in GPU,* Eur. Phys. J. C (2011)71:1559.
3. L.Yu. Barash, L.N. Shchur, *RNGSSELIB: Program library for random number generation. More generators, parallel streams of random numbers and Fortran compatibility*, Computer Physics Communications 184 (2013) 2367.
4. L.Yu. Barash, L.N. Shchur, *PRAND: GPU accelerated parallel random number generation library, Computer Physics Communications 185 (2014) 1343.*
5. Л.Ю. Бараш, Л.Н. Щур, Использование гибридных суперкомпьютеров для многомерного численного интегрирования методом Монте-Карло, готовится к печати