大規模並列処理と擬似乱数

斎藤睦夫, 松本眞

広島大学

2019-03-18

This study is granted in part by JSPS Grant-In-Aid #26310211, #18K03213

擬似乱数

擬似乱数生成器はコンピュータの中でアルゴリズムに従って乱数のように見える数列を生成するプログラムである。

擬似乱数生成器を構成する要素

- 状態空間
  有限集合 S
  初期状態 $s_0 \in S$
- 状態遷移関数
  $f : S \to S$
  n 回目の状態 $s_n = f(s_{n-1})$
- 出力関数
  $o : S \to O$
  O は 32bit 整数, 64bit 整数, 倍精度浮動小数点数など。
  n 回目の出力 $o(s_n) \in O$

擬似乱数:評価

求められる性能

- 周期
  - 擬似乱数は、(状態空間が有限なので) 必ず周期的である。
  - 状態空間の大きさによって周期の上限が決まる。
  - 初期値によって周期が変わる可能性がある。
  - 最低周期の保証が必要 (または常に同じ周期である保証)。
- 一様分布
  $[0, 1)$ 区間に一様かつ独立に分布する確率変数の列をシミュレートしたい。
- (予測困難性)
  (シミュレーションに用いる擬似乱数に予測困難性が必要か?)

擬似乱数:評価

統計的検定

- DIEHARD (Marsaglia, n.d.)
  1995 年 に発表され古くから使われている擬似乱数検定。
- NIST 検定
  アメリカ国立標準技術研究所による検定
  A Statistical Test Suite for Random and Pseudorandom.
  （NIST の以前のバージョンの検定法には問題があった。）
- TestU01 (L'Ecuyer & Simard, 2006)
  DIEHARD や NIST 検定も含む大規模な検定が行える。
  - SmallCrush 10 テスト
  - Crush 97 テスト
  - BigCrush 106 テスト
    （1 テスト当たりのサンプルサイズも大きい）

擬似乱数:評価

計算機資源

- メモリ
  状態空間サイズ$+\alpha$。
  GPU などでは問題になることもある。
- 時間
  1 秒間に何個乱数を生成できるか？
  32 bit, 64 bit, single, double で異なる。

古典的擬似乱数生成器

以下の古典的擬似乱数生成器にはいくつか問題がある。

- 平方採中法 (最古?の擬似乱数 John von Neumann)
- 線形合同法 (LCG)
- その他状態空間の小さいもの (2^{32}bit 程度のもの)

問題点

- 周期 (P) が短い ($P \leq 2^{32}$)。
  周期を超えて乱数を使用すると、超えた部分について同一のシミュレーション結果となる。
  特にレアケースに該当すると全体に大きな影響がある。
  （たまたま問題が発生しないこともある）
- 下位ビットの乱数性が低い。
  偶数と奇数を交互に繰り返すなど。

現代的擬似乱数生成器

カオス的擬似乱数生成器

- カオス理論によって予測困難な擬似乱数。
- 現在も新しいカオス的擬似乱生成器が次々と提唱されている。
- （カオス的擬似乱数の連続する出力は、カオス性が現れるほど離れていない？）

$\mathbb{F}_2$-線形フィードバックシフトレジスタ

- 有限体の数論・幾何に基づく擬似乱数。
- 数学的な性質が証明によって保証される。

カウンターベース擬似乱数生成器

- 状態遷移関数 $f(n) = n + 1$。
- （暗号論的に）複雑な出力関数を用いる（AES など）。

RANLUX

カオス的擬似乱数生成器 RANLUX (Lüscher, 1994)

- 1994 年 Martin Lüscher(理論物理学者)
- 1994 年 F. James (Fortran 版)
- 贅沢度 (luxury) パラメータで乱数の質を制御する。

GNU Scientific Library (GNU, 2004) での RANLUX の紹介

> *For the most reliable source of uncorrelated numbers, the second-generation RANLUX generators have the* **strongest proof of randomness**.
>
> — *GSL Manual Section 18.9*

（私たちとしては同意し難い）

Mersenne Twister

$\mathbb{F}_2$-線形擬似乱数生成器 Mersenne Twister

Mersenne Twister 19937 (Matsumoto & Nishimura, 1998) (MT19937) の特徴

- 状態空間 19937 bit (32bit $\times$ 624 - 31)。
- 周期 $2^{19937} - 1$ ($> 10^{6000}$)。
- 均等分布次元（後述）を改善する出力関数。
- 配列による初期化サポート。
- 線形出力。
- C++11 から標準テンプレートライブラリに採用。
  std::mt19937, std::mt19937_64
- 多くの言語、ライブラリで標準（デフォルト）擬似乱数に採用。

Mersenne Twister 一族

- MT64-19937 (Nishimura, 2000)
  64 bit 出力の Mersenne Twister。
- SFMT19937 (Saito & Matsumoto, 2008)
  128 bit SIMD に合わせた擬似乱数生成器。
- dSFMT19937 (Saito & Matsumoto, 2009)
  倍精度浮動小数点数出力専用の 128bitSIMD 擬似乱数生成器。
- MTGP (Saito & Matsumoto, 2013)
  GPU 用の擬似乱数生成器、乱数ベクトル生成用。
- TinyMT
  127 bit の状態空間を持った周期 $2^{127} - 1$ の擬似乱数生成器。
- dcmt (Matsumoto & Nishimura, 2000)
  Mersenne Twister のパラメータを生成するプログラム。

現代的評価指標：均等分布次元

Definition

b ビット整数の周期 P の周期列

$$\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{P-1}, \mathbf{x}_P = \mathbf{x}_0, \ldots$$

は周期 $i = 0, \ldots, P-1$ までみたとき、すべての kb ビットパターンが k 組

$$(\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+k-1})$$

として同じ回数だけ出現する時、k 次元均等分布するという。

(ただし、全部 0 というパターンは他のパターンより 1 回少なくてよいことにする)

現代的評価指標：均等分布次元

Definition

b ビット整数の周期列は、上位 $v(\leq b)$ ビットが k 次元均等分布しているとき、v ビット精度 k 次元均等分布するという。
そういう k の最大値を $k(v)$ と表す。周期 P の擬似乱数列には以下の上限がある。

$$k(v) \leq \lfloor \log_2(P+1)/v \rfloor,$$

- どんな擬似乱数に対しても定義できる。
- $\mathbb{F}_2$-線形擬似乱数の場合は、実際に計算することができる。
- 計算できる $\Rightarrow$ 改善できる。

擬似乱数の評価結果

統計的検定 TestU01 Bigcrush

- 24 bit RANLUX
  TestU01 は 32bit 擬似乱数用なので検定できない。
- 48 bit RANLUX
  上位 32bit の検定で BigCrush を通る。
- MT19937, MT19937-64, SFMT, dSFMT
  $\mathbb{F}_2$-線形性の検査である
  MatrixRank と LinearComplexity に落ちる。
- TinyMT
  非 $\mathbb{F}_2$-線形出力関数の採用により BigCrush を通る。

擬似乱数の評価結果

統計的検定

- 統計的検定に落ちることは擬似乱数の **Badness** を示す。
- 統計的検定に通っても擬似乱数の **Goodness** は分からない。
- $\mathbb{F}_2$-線形性は予測不可能性の検査である。
  （シミュレーション用乱数に必要か？）

擬似乱数の評価結果

v ビット精度均等分布次元 $k(v)$

$k(v)$	$k(2)$	$k(4)$	$k(8)$	$k(16)$	$k(32)$	$k(64)$
mt19937	9968	4984	2492	1246	623	-
mt19937-64	9968	4984	2202	1246	623	311
周期 2^{568}	284	142	71	35	17	8
周期 2^{128}	64	32	16	8	4	2

周期 2^{568}, 2^{128} については理論的上限

- 均等分布次元は擬似乱数の **Goodness** を表す。
  （私たちの信念）
- RANLUX の周期はおよそ $10^{171} \approx 2^{568}$。
- TinyMT の $\mathbb{F}_2$-線形出力は、多くのパラメータで理論的上限。

速度比較

擬似乱数

- GNU Scientific Library
  - taus : a maximally equidistributed combined Tausworthe generator by L ’ Ecuyer.
  - mt19937
  - cmrg : combined multiple recursive generator by L ’ Ecuyer.
  - ranlxd1 : 48 bit RANLUX luxury level 1.
  - ranlxd2 : 48 bit RANLUX luxury level 2.
- TinyMT

速度比較

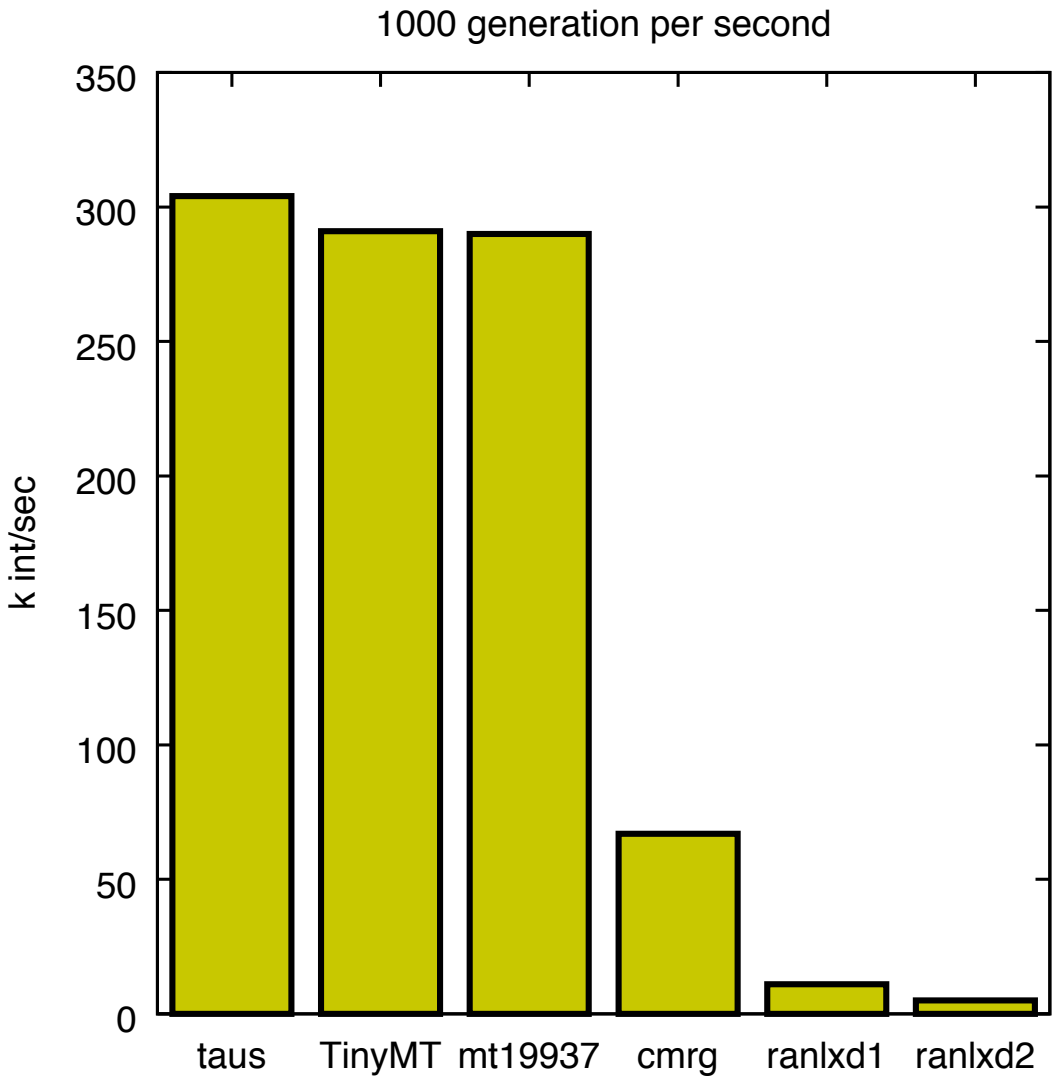
1 秒間に生成する擬似乱数の数（単位 千個）

Computer: iMac

CPU: Intel Core i5-6500

Clock: 3.20GHz

Compiler: clang-1000.11.45.5



並列処理と擬似乱数

並列処理のタイプ

- CPU
  - １プロセスの使用可能メモリは多い
  - 並列度は中
- GPU
  - 並列度は大
  - 1 スレッドの使用可能メモリは少ない
- スーパーコンピュータ
  - 並列度は大
  - 1 スレッドの使用可能メモリは多い
  - GPU も併用できる？
- （FPGA?）

並列処理と擬似乱数

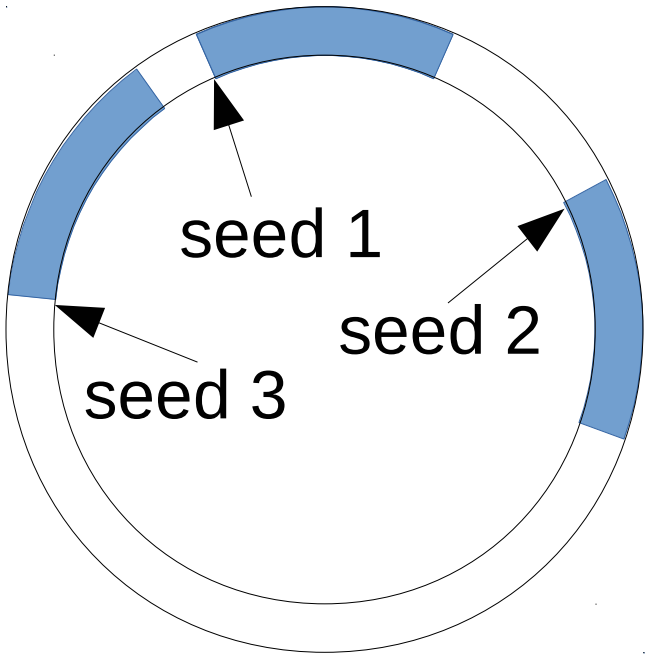
並列計算における擬似乱数の生成法

- 乱数ベクトル
  乱数の配列を生成し、並列処理による行列計算やベクトル計算に使う。
- 共有
  ひとつの擬似乱数生成器を複数のスレッドで共有する。
  Java のマルチスレッドなど。
  （シミュレーションにおいては実用的ではない）
- 個別インスタンス
  プロセスまたはスレッド毎に別の実体として擬似乱数生成器を使う。

個別インスタンス

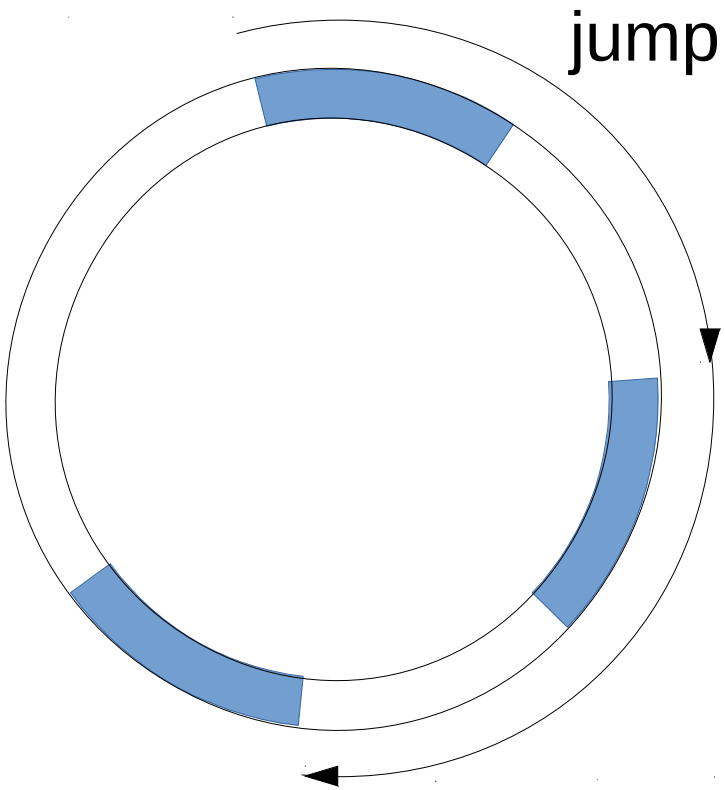
並列実行単位ごとにひとつの擬似乱数を使用する。
それぞれに異なる乱数列を生成する必要がある。
その方法は、

- 初期化の種 (seed) を変える
  長い周期の乱数列の中の異なる部分列を利用することになる。
  周期が長ければ、部分列が一致する確率は小さい。



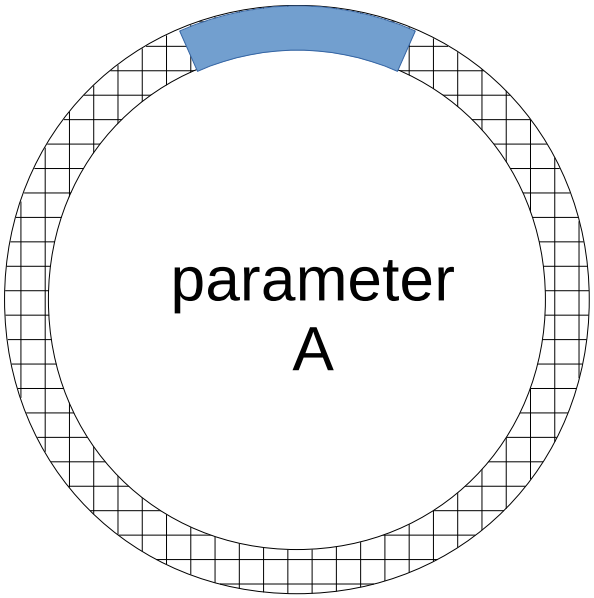
個別インスタンス

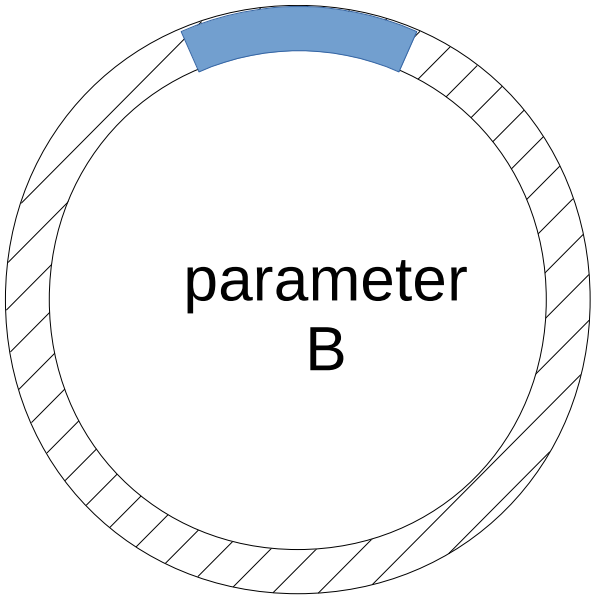
- Jump
  指定した数だけ離れた部分列を使用する。
  （Jump が大きければ）部分列は一致しない。

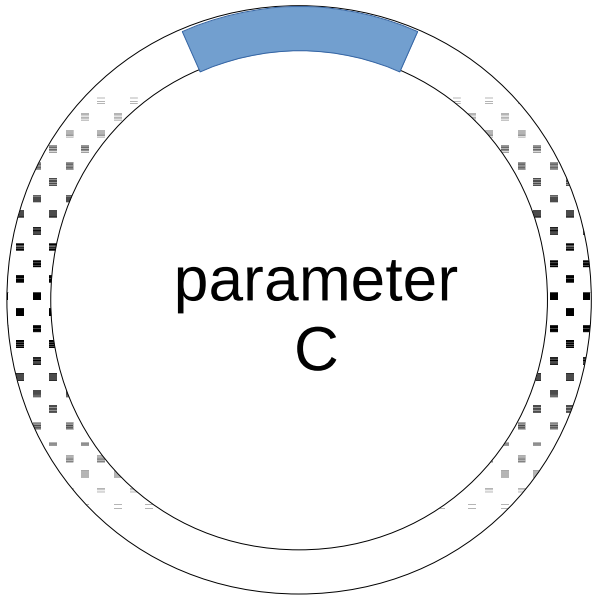


個別インスタンス

- パラメータ
  擬似乱数生成器のインスタンスごとに状態遷移関数が異なる。
  異なる乱数列が生成される。







初期化の問題

擬似乱数生成法とは別に初期化の問題がある。

- 初期化アルゴリズムが悪いと、最初の方の出力に偏りと相関が生じる。
- 多くの擬似乱数生成器で seed が 2^{32} 通りしか選べない。
- seed が衝突すると同じ擬似乱数列が生成される。
- seed が衝突しなくても、部分列が一致する可能性がある。

初期化の問題

（一般化された）バースデーパラドックス

d: 誕生日の数に相当するもの, seed の範囲
$n(d)$: 人数に相当するもの, 並列度, これだけの人数があつまると同一誕生日の人が存在する確率が 0.5 を超える。

$$n(d) \approx \sqrt{2d \ln 2} \approx 1.18\sqrt{d}$$

- $d = 2^{32}$ の場合、スレッド数 2^{16} 程度で seed が衝突する確率が 0.5 を超える。
- seed の衝突を避けるために、プロセス ID, スレッド ID などを seed にすればよい。
  $\Rightarrow$ 系統的な **seed** が系統的に相関する出力を生成する問題。

初期化の問題: 部分列が一致する可能性

並列計算における擬似乱数のサーベイ論文 (L'Ecuyer, Munger, Oreshkin, & Simard, 2017) によると

- 多くの並列実行部のどれかひとつで部分列が一致する確率: p
- 擬似乱数の周期: L
- 並列度: s
- 部分列の長さ ℓ
- $p \approx s^2 \frac{\ell}{L}$

$L = 2^{128}, s = 2^{30}, \ell = 2^{40}$ の場合

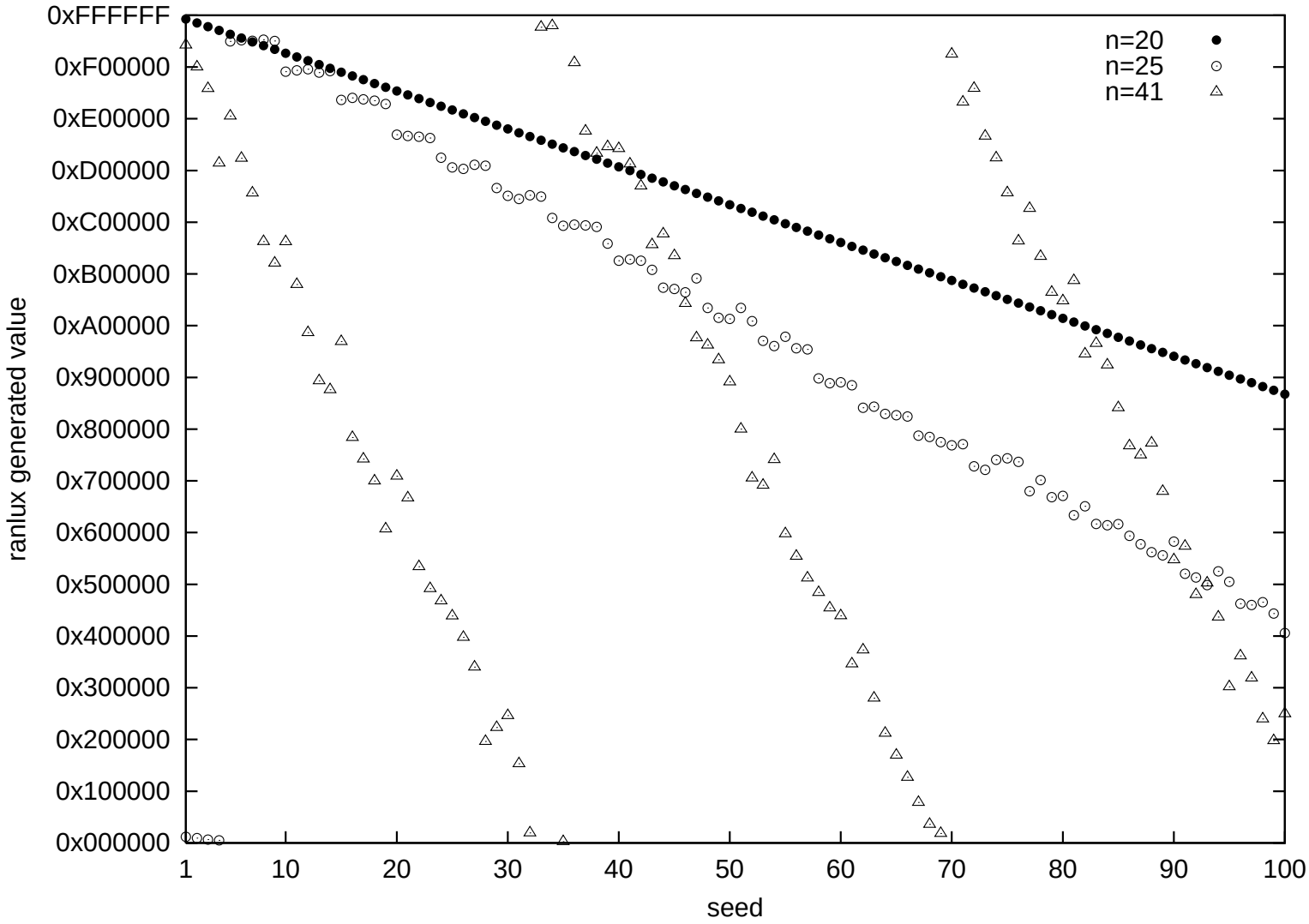
- $p \approx 2^{-28} \approx 10^{-8.4}$
- 周期 2^{128} 程度なら大丈夫か？
- （部分列が一致しなくても、状態遷移が線形の場合線形関係が維持される。）

初期化の問題: 系統的 seed

seed を系統的に選んだときの非乱数性 (RANLUX) (Matsumoto, Wada, Kuramoto, & Ashihara, 2007)

- 24 bit 出力のオリジナル RANLUX
- luxury 199 (James による Fortran 版のデフォルト値)
- 24 個出力して 199 個捨てる。
- seed を $1, 2, 3, \ldots, 100$ と動かして
  20, 25, 41 番目の出力値 (24bit 値) をプロット

初期化の問題: 系統的 seed



初期化の問題: 系統的 seed

- 20 番目の出力が seed 1 で大きな値で、seed が大きくなるに連れてゆっくり小さくなる。
- 25 番目の出力は最初の 4 個の seed は 0 付近、次に大きな値でその後 seed が大きくなるにつれて小さくなる。
- 41 番目の出力は、seed が大きくなるにつれて小さくなり、最低になると再び大きなところから繰り返す。
- この 3 つは seed が近い値だと出力も近い値になる。

GNU Scientific Library(2004 年版)
58 個の擬似乱数発生法のうち、45 個にこのような問題が観測された。(付録 A 参照)

初期化の問題: 配列による初期化

- mt19937 では配列による初期化 init_by_array がサポートされている。

例えば、並列処理で 3 次元のシミュレーションを行う場合、各プロセスには 3 次元のインデックス idx, idy, idz が割り当てられているとするとシミュレーションの度に異なる seed と合わせて、次のように MT を初期化できる。

init_by_array の使用例

```
uint32_t length = 4;
uint32_t seed_ar[] = {seed, idx, idy, idz};
init_by_array(seed_ar, length);
```

Jump

状態 s から n 個先の状態 $f^n(s)$ を高速に計算する。

- Jump を使えば部分列が一致しない擬似乱数列が得られる。
- 使用した乱数の個数が分かっていれば、
  並列度の異なる環境で再現シミュレーションが出来る。

具体的な計算方法

- MT19937 については NTL(Number Theory Library) などのライブラリを使用する。
- TinyMT については、Jump 関数が付属している。
- カウンターベース擬似乱数なら容易。
- cuda の乱数ライブラリ cuRAND では offset という名で jump 機能が提供されている。（すべての乱数ではない）

Jump

注意点

- カウンターベースの擬似乱数
  - Jump は高速。
  - 出力関数は一般に遅い。
  - 出力関数の速度と乱数性のトレードオフ関係。
- $\mathbb{F}_2$-線形擬似乱数
  - Jump は速いとはいえ、それなりの時間がかかる。
    特に周期が長い場合。
  - 多項式の計算をするためにメモリを消費する。
    特に周期が長い場合。

パラメータ付き擬似乱数生成器

パラメータ付きの擬似乱数生成器は、パラメータを変えることによって異なる擬似乱数生成器を生成するものである。（状態遷移関数が異なる）

パラメータ付きの擬似乱数生成器

- Mersenne Twister
  - Dynamic Creator (dcmt)
  - 周期と id を指定して擬似乱数生成器を生成する。
  - id が異なれば異なる状態遷移関数（擬似乱数生成器）となる。
  - 周期が長いとパラメータの計算に時間がかかる。
- TinyMT
  - パラメータの計算時間が短い。
  - 力武健次氏によって大量（2.6 億個）のパラメータが計算済み。
    `https://github.com/jj1bdx/tinymtdc-longbatch`

パラメータ付き擬似乱数生成器

特徴と注意点

- 他の系列と部分列が一致する心配はない。
- パラメータは計算によって求めた値でなければならない。
- パラメータの計算にはそれなりの時間がかかる。
  特に周期が長い場合。
- 前もって計算しておいた場合
  （並列度の応じた）大量のパラメータを読み込まなければならない。

上記問題の解決法として、Jump や seed との併用が考えられる。

- １つの CPU プロセスで１つのパラメータを読み込み、そのプロセスから生成される GPU のスレッドではそのパラメータで異なる seed を使うなど。

制限のある並列実行

GPGPU などでは並列実行環境に制限がある。

- メモリ制限
  周期の長い擬似乱数は使用メモリが大きい。
  $2^{127} - 1$ 程度の周期ならば十分であろう。
- 倍精度計算、64bit 整数計算
  実行できるが遅い場合がある。
- 条件分岐
  実行できるが遅い場合がある。

使用メモリの比較的小さい擬似乱数生成器

- 周期 2^{32} シミュレーションには周期が短かすぎる
- 周期 $2^{128}(\approx 10^{38.5})$ 1 周使い切ることはない。
  部分列が一致する可能性は、並列度および seed、Jump、パラメータのどの方法を使うかに依存する。
  出力関数を工夫すれば、TestU01 の BigCrush に通る

cuda cuRAND

- XORWOW 状態空間:160bit 周期:約 10^{48}
- MRG32K3A 状態空間:192bit 周期:約 7.6×10^{22}
- PHILOX4_32_10 カウンターベース, 周期:約 $10^{38.5}$

- TinyMT 状態空間:127bit 周期:約 $10^{38.2}$

注意点のまとめ

- 初期化は重要である。
- 配列による初期化は seed の衝突を避ける上で効果がある。
- 周期の長い擬似乱数の方がよい。（信念）
- seed を変える方法では、部分列が一致する可能性がある。
- パラメータの異なる擬似乱数は、並列度が高くても部分列が一致することはない。

シミュレーションに擬似乱数を使う方に聞きたいこと

- 最大並列度（プロセス数またはスレッド数）
- 最大の擬似乱数使用個数 10^n, 2^n
- 並列度を変えた再現シミュレーションの需要
- どのような分布の擬似乱数を使っているか（正規分布、1..N の整数一様分布、0..1 の一様分布)
- ひとつのシミュレーションで複数の分布を使うか
- メモリの余裕はあるか？ GPU などで 256bit, 384bit, 512bit の擬似乱数生成器の需要はあるか？
- 512bit SIMD の擬似乱数生成器の需要はあるか?
- FPGA 用の擬似乱数生成器の需要はあるか?
- 整数の均等分布の需要はあるか、その場合整数の範囲はどのくらいか？

リクエスト受付

前ページの聞きたいことの範囲でなくても、擬似乱数に関する要望があれば、

- なんでも
- 気にしないで
- 予想外の要望の方が面白い
- タイトルに「擬似乱数要望」と書いて

斎藤睦夫 (mailto:sai10@hiroshima-u.ac.jp) まで
お手間でなければ mmat あっと hiroshima-u.ac.jp にも
（なお、要望に応えられるとは限りません）

参考文献 I

GNU. (2004, December). *Gnu scientific library version1.6.* (http://www.gnu.org/software/gsl/)

L'Ecuyer, P., Munger, D., Oreshkin, B., & Simard, R. (2017). Random numbers for parallel computers: Requirements and methods, with emphasis on gpus. *Mathematics and Computers in Simulation*, *135*, 3-17.

L'Ecuyer, P., & Simard, R. (2006). TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, *15*(4), 346-361.

Lüscher, M. (1994, February). A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications*, *79*(1), 100–110.

Marsaglia, G. (n.d.). *The Marsaglia Random Number CDROM, with the DIEHARD Battery of Tests of Randomness.* Retrieved from `http://www.stat.fsu.edu/pub/diehard/` (Department of Statistics, Florida State University, (1996) `http://www.stat.fsu.edu/pub/diehard/`)

参考文献 II

Matsumoto, M., & Nishimura, T. (1998, January). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation*, *8*(1), 3-30. (`http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`)

Matsumoto, M., & Nishimura, T. (2000). Dynamic creation of pseudorandom number generator. In *Monte carlo and quasi-monte carlo methods 1998* (p. 56-69). Springer-Verlag. (`http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/DC/dc.html`)

Matsumoto, M., Wada, I., Kuramoto, A., & Ashihara, H. (2007, September). Common defects in initialization of pseudorandom number generators. *ACM Trans. on Modeling and Computer Simulation*, *17*. (`doi:10.1145/1276927.1276928`)

Nishimura, T. (2000, October). Tables of 64-bit mersenne twisters. *ACM Trans. on Modeling and Computer Simulation*, *10*(4), 348-357.

Saito, M., & Matsumoto, M. (2008). SIMD-oriented fast Mersenne twister : a 128-bit pseudorandom number generator. In *Monte carlo and quasi-monte carlo methods 2006* (p. 607-622). Springer.

参考文献 III

Saito, M., & Matsumoto, M. (2009, Dec.). A prng specialized in double precision floating point number using an affine transition. In *Monte carlo and quasi-monte carlo methods 2008* (p. 589-602). Springer.

Saito, M., & Matsumoto, M. (2013, February). Variants of mersenne twister suitable for graphic processors. *ACM Transactions on Modeling and Computer Simulation*, *39*. (doi:10.1145/1570256.1570353)

付録A

初期化に問題のある GSL の擬似乱数 (Matsumoto et al., 2007)
次ページの表

column affine:
×:persistent nearly affine dependence observed.
△:transient nearly affine dependence observed.
◯:affine dependence not observed.

column collision:
×:dense difference collision observed.
△:sparse difference collision observed.
◯:difference collision not observed.
(We do not mean ◯is recommendable.)

付録A

generator	affine	collision	generator	affine	collision
borosh13	×	×	cmrg	◯	×
coveyou	◯	×	fishman18	×	×
fishman20	×	×	fishman2x	×	×
gfsr4	◯	◯	knuthran	◯	◯
knuthran2	×	×	lecuyer21	×	×
minstd	×	×	mrg	◯	×
mt19937	◯	◯	mt19937 1999	◯	◯
mt19937 1998	◯	◯	r250	△	◯
ran0	×	×	ran1	◯	◯
ran2	◯	◯	ran3	×	×
rand	×	×	rand48	×	×
random8-glibc2	×	×	random32-glibc2	△	△
random64-glibc2	△	△	random128-glibc2	△	△
random256-glibc2	◯	◯	random8-libc5	×	×
random32-libc5	×	×	random64-libc5	×	×
random128-libc5	×	×	random256-libc5	×	×
random8-bsd	×	×	random32-bsd	×	×
random64-bsd	×	×	random128-bsd	×	×
random256-bsd	×	×	randu	×	×
ranf	×	×	ranlux	△	△
ranlux389	△	△	ranlxd1	◯	◯
ranlxd2	◯	◯	ranlxs0	◯	◯
ranlxs1	◯	◯	ranlxs2	◯	◯
ranmar	◯	◯	slatec	×	×
taus	◯	◯	taus2	◯	◯
taus113	◯	◯	transputer	×	×
tt800	△	◯	uni	×	×
uni32	△	×	vax	×	×
waterman14	×	×	zuf	◯	◯