数理解析研究所講究録 1240 巻 2001 年 181-191

長周期非再帰型擬似乱数の並列生成

Parallel generation of long-period nonrecursive pseudorandom numbers

三重大学教育学部 谷口礼偉(Hirotake Yaguchi)
Faculty of Education, Mie Univ.

In this talk we introduce a long-period nonrecursive pseudorandom number generator (SR/4M) which has about 10000 different sequences of random numbers in it, and can easily be applied to parallel computations. We test and compare (SR/4M) with other random number generators and give an example of programming code for MPI parallel computations.

数値計算環境の著しい進歩に伴い，並列計算も MPI(message passing interface) などの標準的な手法が確立され，身近な計算方法となってきた。本講演では，並列度10000程度までの並列計算に適用可能な長周期非再帰型擬似乱数 (SR/4M) を提案し，その特性について調べ，実際の使用法について述べる。

擬似乱数の生成法として色々な方法が知られているが，多くは乱数値を再帰的に算出するものである（＝既に計算した過去の幾つかのデータを使って次の乱数値を計算する，いわば直列的な計算による乱数生成）。この方法は1個の乱数値生成に対する計算量が少なくてすみ高速であるが，並列計算においては各プロセスから発生する乱数値要求が並列（同時）に重なることになるので，これに対する対応を別途考えなくてはならない（→生成方法の複雑化，並列処理の効率低下など）。これに対し，[1](あるいは[2])で述べられている新しい乱数生成法 (SR/4) は，n 番目の10進4桁乱数を浮動小数点演算（もしくは64ビット整数演算）により直接（非再帰的）に発生するという特徴を持っており，乱数生成速度はやや遅いものの周期は 10^{15} 程度あり，容易に並列計算に対応できるので，その乱数生成の仕組み，並列計算への対応法，特性，使用法などについて述べていくことにする。

1．(SR/4)の並列計算対応化

擬似乱数(SR/4)による乱数 z_k, $k=1,2,\cdots$, は，あらかじめ準備されている巨大な 10 進 4 桁の「電子式」乱数表 $\{u_{ij} \mid i=0,1,\cdots,p-1,\ j=0,1,\cdots,q-1\}$, ($p=49933453$, $q=22801201$, いずれも素数) に対して，別途素数の組 (r,s) を用意し，

$$z_k = u_{(rk \bmod p,\ sk \bmod q)},\quad k=1,2,\cdots,$$

とおいたものと考えることができる。（各 u_{ij} は，実際には使用の都度実数シフト(SR)法により直接計算される。）周期は $p\times q$ = 1,138,542,698,477,053 である。(SR/4) では $r=r_0$=491377, $s=s_0$=47513 と設定しているが，素数 r, s を適当に変えて組み合わせればいくらでも周期 pq の異なる乱数系列が得られる。そこで，(SR/4)を並列計算に対応させるため（並列計算対応版を(SR/4M)と呼ぶことにする），r として r_0 を含む連続する 199 個の素数をとり(r_{198}=494041)，また s として s_0 を含む連続する 53個の素数をとり(s_{52}=48049)，計 199×53= 10547 個の (r,s) の組み合わせを準備しておく。そして，(SR/4M) では L 番目 (L=0,$\cdots$,10546) の乱数系列として，(r,s)=(19L mod 199, 5L mod 53) に対応する(SR/4)の乱数系列を使う（乱数値は 81899×7919×L 番目から順次使っていく）。(SR/4M) を使う実際の並列計算においては，L 番目の並列プロセスは例えば (SR/4M) の L 番目の乱数系列を使えば良い。以上が並列計算対応版(SR/4M)の概略である。(SR/4)については次節で触れる。

2．実数シフト法と(SR/4)

まず基本となる，実数シフト(SR)法による擬似乱数生成について簡単に説明しておく（詳細は[1][2]を参照）。

(SR)法は，コンピュータによる以下のような擬似乱数生成法である。

区間 [16,32] を n-1 等分し

$$x_i = 16 + \frac{16}{n-1}\times i,\qquad i=0,1,\cdots,n-1,$$

とおく。$x=x_i$ として，

$$f(x) = x\cdot\frac{x}{2}\cdot\frac{x}{3}\cdot\frac{x}{4}\cdots\frac{x}{23}\cdot\frac{x}{24}$$

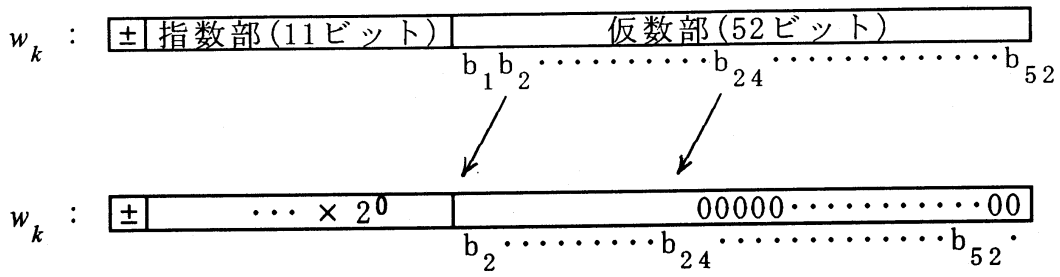
を

$$w_0 := 1 \qquad w_k := w_{k-1}\times\frac{x}{k},\quad k=1,2,\cdots,24,$$

により倍精度計算する。ただし，w_k を1回計算するごとに，w_k を表す倍精度変数に対して，

i) 仮数部の全てのビット値を１ビット左にシフトし，

ii) さらに，仮数部上位23ビット以外のビットは０にする；



iii) また，指数部は　$\cdots \times 2^0$　となるように設定する($\Rightarrow$ $1 \leq w_k < 2$ となる)。

このようにして得られた $f(x_i)$ の計算結果 f_i を浮動小数点表示し，上位3桁の数字を捨て，残った桁から続けて4桁の数をとり，これを乱数値とする。i を 0 から $n-1$ まで変化させれば，n 個の10進4桁数からなる1つの乱数列が得られる。これが(SR)法である。

(SR/4)では，長周期化のため，n として 192000, 192001, $\cdots$, 48060000 を使い，区間 [16, 32] を $n-1$ 分割し，その各分点 x_i における上述の計算結果 f_i を以下に述べる規則「実数シフト計算IV」で変形したのち，(変形後の) f_i から 10進4桁乱数値を取りだし適当に番号を付けて並べ，前述の巨大な電子式乱数表 $\{u_{ij}\}$ を得ている。

「実数シフト計算IV」による f_i の変形は，2つの変形「実数シフト計算IVa」および「実数シフト計算IVb」を「実数シフト計算IVc」により使い分ける形式をとっている。

【実数シフト計算IVa】　実数シフト法(SR)による $f(x_i)$ の計算結果 を f_i とする。さらに

$$b_e = b_6 + b_8 + \cdots + b_{20} \pmod 2 \qquad b_o = b_7 + b_9 + \cdots + b_{21} \pmod 2$$

とおく。このとき，

(i)　「$f_i < 1+\alpha$　または　$f_i \geq 2-\alpha$」 であって $b_e \neq b_o$ のとき，

もしくは

(ii)　「$f_i \geq 1+\alpha$　かつ　$f_i < 2-\alpha$」 であって $b_e = b_o$ のとき，

f_i の仮数部 1～23 ビットの全ビット値を反転する。

【実数シフト計算IVb】　「実数シフト計算IVa」における

(i) の　... $b_e \neq b_o$ ...　と　　(ii) の　... $b_e = b_o$ ...　を

入れ替えて計算したもの。

【実数シフト計算IVc】

$$b_{eo} = b_6+b_7+b_8+\cdots+b_{19}+b_{20}$$

とおく。 $b_{eo}<8$ なら「実数シフト計算IVa」を適用し，$b_{eo}\geq 8$ なら「実数シフト計算IVb」を適用する。

α を決めるのはかなり厄介である。[2]では α として 0.36 を採用しているが，

$1+\alpha = \mathrm{ae0ebb}_{(16)}/800000_{(16)}$ (= 11407035/8388608) = 1.359824538230896

が良さそうなので以後これに基づいて数値計算を進めていく。なお，以下の数値計算結果は幾つかの例外を除いて，統計数理研究所のスーパーコンピュータ Hitachi SR8000 により，倍精度浮動小数点演算に相当する計算を64ビット整数計算でエミュレートし，並列計算したものである。（註：その後の計算結果によると，11407035 のかわりに 11407166 を使っても良さそうである。）

3．(SR/4M)の特性および他乱数との比較

(SR/4M)は (r, s) の組み合わせにより 10547 個の乱数系列を持っているが，全ての系列の特性を調べることは現時点の計算機ではほぼ不可能であるので，r, s に割り振った値のうち最大，最小の値を含む各4個をほぼ等間隔に

r = 491377, 492299, 493177, 404041,　s = 47513, 47699, 47869, 48049

と取り出して，その組み合わせについて調べることにする。調べる方法は，基本的に [2] による。また比較のために使う乱数生成法は

略号	乱数生成法
(FSR)	feedback shift register法,
(MT)	Mersenne Twister法,
(BC)	Borland C++ V.55 に付属する乱数関数,
(Ph)	統計数理研究所の Hitachi SR8000 による物理乱数

である。参考のため各乱数生成法の要点を示しておく。

(FSR)feedback shift register法

$$Y_n = Y_{n-32} \text{ XOR } Y_{n-521} \quad \text{(32-bit integer)}, \quad n \geq 521,$$

により，Y_{521}, Y_{522}, ... を発生し，Y_n/429496.7296 の整数部分を取り出して 10 進4桁乱数を得る。

(MT)MT法

http://www.math.keio.ac.jp/~matsumoto/mt.html

よりダウンロードした C プログラムにより，32ビット一様乱数を得，それを429496.7296 で割り，整数部分を取り出して10進4桁乱数とした(seed=4357)。

(Ph)物理乱数

統計数理研究所のコンピュータ HITACHI SR8000 から，物理乱数を31 ビット整数値で取得し，これを 214748.3648 で割ってその整数部分をとり，4桁乱数とした。

(BC)BC++ V.55

Borland社の提供する32ビット版フリーコンパイラ BC++ V.55 の乱数関数 random(10000) により，4桁整数乱数を得た(seed=987654321)。 ((有料の) BC++ V.50 とは出てくる乱数が違うので要注意。)

3．1 「各種検定の組み合わせ」による検定

(r, s) の各組み合わせに対し，4桁乱数を20000個発生し，以下の [検定II] から [検定VIII] まで7種類計10 の検定を危険率 0.05 で行う。

[検定II] 文字0～9の出現頻度のχ^2検定
[検定III] 文字0の出現間隔のχ^2検定
[検定IV] 乱数値の Kolmogorov-Smirnov 検定 (K^+)
[検定IV] 乱数値の Kolmogorov-Smirnov 検定 (K^-)
[検定V] 単純上昇連テスト
[検定V] 単純下降連テスト
[検定VI] 4枚の 0～9 カードによる古典ポーカーテスト
[検定VII] 遅れ 1 の系列相関テスト
[検定VII] 遅れ 2 の系列相関テスト
[検定VIII] 衝突テスト

その結果仮説が棄却された回数 c($0 \leq c \leq 10$) を数える。この操作を1000 回繰り返し，得られた c の値を c_1, c_2, ... , c_{1000} とおく。各検定が危険率 0.05 で独立に行われると仮定すれば (厳密に言えば，正確な仮定とは言えな

いが），c の分布は二項分布 $Bin(10, 0.05)$ になる。これについてχ^2検定を行うと以下の数値が得られる。

		r			
		491377	492299	493177	404041
s	47513	0.334	0.020Δ	0.146	0.218
	47699	0.660	0.573	0.244	0.484
	47869	0.887	0.439	0.018Δ	0.159
	48049	0.373	0.399	0.701	0.855

仮定が厳密でないとはいえ，良い値がでていると言えよう。他乱数の結果は次の通りである：

(FSR)	(MT)	(BC)	(Ph)
0.729	0.182	0.484	0.546

各検定は，(r, s) の組み合わせに対してそれぞれ1000回行われるが，その棄却回数は次のようになる：

r	s	文字	間隔	K^+	K^-	上連	下連	pokr	相関	相関	衝突	平均
491377	47513	58	54	47	44	49	62	60	49	52	50	52.5
491377	47699	50	39	44	53	54	40	52	43	44	64	48.3
491377	47869	54	49	40	43	52	59	47	49	50	61	50.4
491377	48049	55	56	42	46	64	56	43	54	47	52	51.5
492299	47513	45	48	46	58	53	41	36	49	53	49	47.8
492299	47699	54	38	44	45	50	44	44	52	50	50	47.1
492299	47869	43	46	52	51	57	54	50	29	46	48	47.6
492299	48049	51	39	46	39	45	52	42	51	40	72	47.7
493177	47513	55	52	51	43	44	52	55	40	50	63	50.5
493177	47699	48	53	50	50	50	54	54	42	47	57	50.5
493177	47869	49	36	37	42	52	60	42	42	44	41	44.5
493177	48049	55	57	48	41	53	49	47	43	46	61	50
494041	47513	42	43	49	51	41	47	32	36	56	59	45.6
494041	47699	54	45	46	50	57	61	45	37	42	54	49.1
494041	47869	52	47	39	51	44	50	53	52	49	66	50.3
494041	48049	47	41	45	59	54	60	50	48	52	51	50.7
各検定平均		50.8	46.4	45.4	47.9	51.2	52.6	47.0	44.8	48.0	56.1	49.0

これらの結果から，(r, s) を上述の範囲で変化させたとき乱数特性が大きく

変わるものではない，ということがいえよう。

異なる乱数系列に属する乱数間の特性

以上の結果は，個々の(r,s)に対応して作られる乱数列に関しての特性であった。実際の並列計算では，(r,s)の組み合わせを同時に多数使うわけであるから，異なる (r,s) によって発生される乱数間の関係についても，特性を調べておく必要がある。

並列的に発生される乱数系列間の特性を調べる方法として，複数の (r,s) 系列から発生される乱数値を，1つの乱数系列にまとめて，今までの検定法を利用することにする。即ち，(SR/4M)の0～9999 番の各乱数系列から，0番目の乱数を順に取り出していき，最後まで行ったらまた最初(=0番)の乱数系列に戻って1番目の乱数を順に取り出していくようにして得られる乱数列について，上述の検定法を適用する。結果は以下のようになり，今までの結果と比べて特に目立つ所はない。

[7種類計10検定に対する棄却回数 c の分布]

c の値 ⇒	0	1	2	≥3	CHITEST
発生度数	623	281	81	15	0.0986
$Bin(10,0.05)$	0.5987	0.3151	0.0746	0.0115	

[各検定に対する棄却回数の内訳]

文字	間隔	K^+	K^-	上連	下連	pokr	相関	相関	衝突	平均
60	50	50	53	46	51	48	34	47	50	48.9

３．２ Kolmogorov-Smirnov統計量に対するχ^2検定

[**検定IX**] 4桁乱数を80000個発生し，一様に分布に関する Kolmogorov-Smirnov 統計量を求める。すなわち，発生した4桁の数値 u_1, u_2, u_3, ･･･, u_n, n=80000, に対し，[0,1) 上の数値 $0.u_1$, $0.u_2$, ..., $0.u_n$, を考え，

$$F_n(x) = \frac{1}{n} \#\{\, i \mid 0.u_i \le x\} \,, \quad x \in [0,1)$$

とおく。また，$F(x)=x$, $x\in[0,1)$ とする。 Kolmogorov-Smirnov統計量は，

$$K_n^+ = \sqrt{n}\ \max_x\{F_n(x)-F(x)\}, \qquad K_n^- = \sqrt{n}\ \max_x\{F(x)-F_n(x)\}$$

で与えられる。これを10000回繰り返し，K_n^+，K_n^- の分布状態を調べる。K_n^+，K_n^- の理論頻度は，n が十分大きいとき，分布 $P(x)=1-\exp(-2x^2)$ に近づくことが知られているので，危険率 0.05 でχ^2検定を行い，棄却されるかどうか調べる。

各 (r,s) の組み合わせに対してこの［検定IX］を1000回繰り返したとき，棄却された回数は次表の通りである。

		r							
		491377		492299		493177		494041	
s	47513	315	352	324	354	311	337	304	358
	47699	326	349	331	328	319	364	336	314
	47869	313	338	326	334	327	358	329	350
	48049	311	330	336	349	351	334	341	326

（左［右］側の数値が K^+［K^-］の分布に対する棄却数）

検定の有意水準を 0.05 に設定した割には棄却数が多いが，他の乱数生成法および物理乱数においても，次に示すように同様の傾向を示しており，特に悪い結果ではない：

	(FSR)		(MT)		(BC)		(Ph)	
K^+ K^-	780	593	349	344	332	344	307	346

［検定IX］を1回行うと，10000個のKolmogorov-Smirnov統計量 $K_n^+(j)$，$K_n^-(j)$，j=0, 1, . . . , 9999，が得られるが，

$$\tilde{D} = \left[\ \#\{j \mid K_n^+(j)<K_n^-(j)\} - \#\{j \mid K_n^+(j)>K_n^-(j)\}\ \right]$$

とおくと，分布の非対称性に関する情報が得られる。［検定IX］は (r,s) の組み合わせごとに1000回行われるので，対応する 1000 個の $\tilde{D}$ について平均をとると次表が得られる。

		r			
		491377	492299	493177	404041
s	47513	0.9	1.2	-2.6	-4.6
	47699	0.9	2.7	2.3	0.8
	47869	3.5	-1.5	3.2	-4.7
	48049	-7.9	3.4	2.5	2.2

他乱数の結果は次の通りである：

(FSR)	(MT)	(BC)	(Ph)
130.9	1.8	-6.4	-1.5

(SR/4M)の結果は多少ばらつきがあるが，他乱数と同程度並であることが分る。(FSR)の値はかなり大きいようである。

異なる乱数系列に属する乱数間の特性

前と同様に (SR/4M)の０～9999 番の乱数系列から，１つずつ順に乱数を取り出していく乱数列について，上述の $K^+[K^-]$ に対するχ^2検定を1000回繰り返したとき，棄却される回数は 328[349] 回である。また，1000 個の $\tilde{D}$ に対する平均は 0.1023 である。

４．並列計算における (SR/4M) の使用法

(SR/4M)は，単に複数の乱数系列を持つ疑似乱数生成器であるということであり，並列計算専用と言う訳ではない。並列計算の各プロセスで異なる乱数系列を使えば，自然に乱数を並列生成することになる。以下，ＭＰＩ(Message Passing interface,メッセージ通信インターフェイス)による並列計算について解説する。

(SR/4M)は２つのエントリを持っている。

```
void SRIniM(int m, double n)
```

は乱数列初期化のためのもので，m番目の乱数系列を使い，n番目の乱数から生成することを指定する。指定が無い場合 m=0，n=0.0 である。もう１つの

```
int  SR4Dg4M(void)
```

により，10進4桁の乱数を得ることができる。

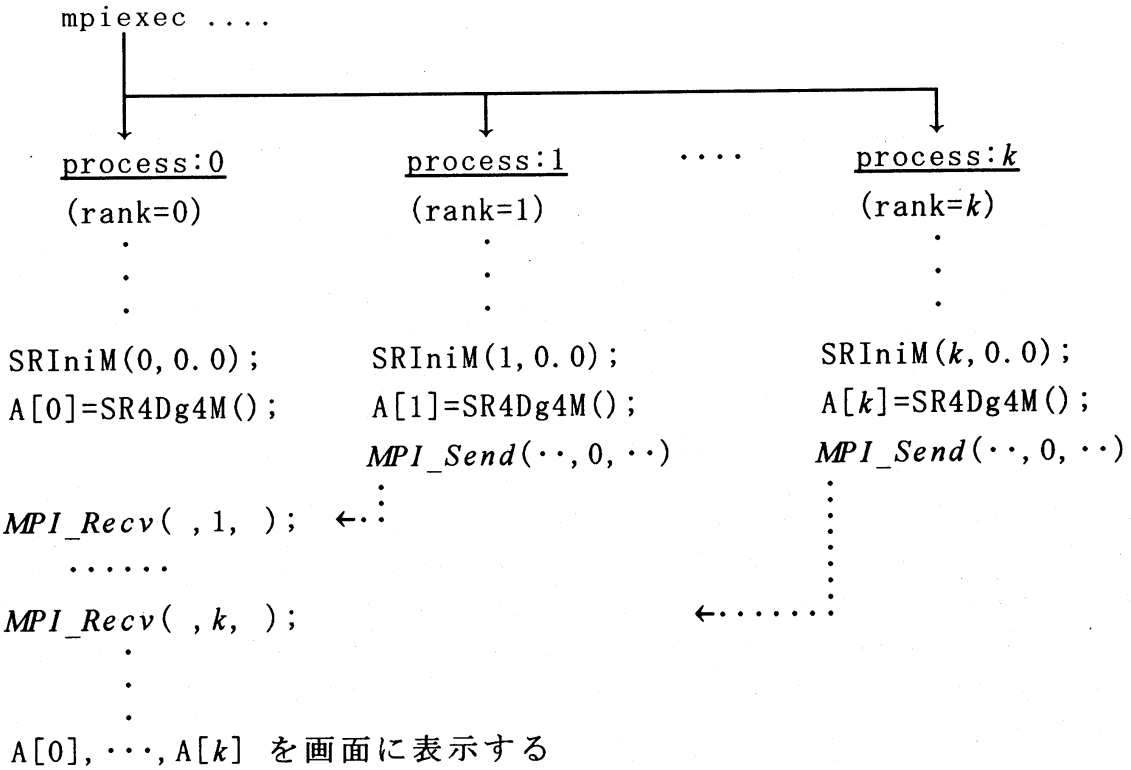
次の例プログラムは，並列して実行される各プロセスにおいて，プロセス番号と同じ番号を持つ乱数系列の最初の乱数値を並列計算して表示するものである。

```
#include <stdio.h>
#include <mpi.h>
#include "sr4m.c"          /* (SR/4M)本体 */
/*****************/
int main(int argc, char * argv[])
    {
    MPI_Status mpistat;
    int  A[10547];
    int  mpisize, myrank, RecvNo, mpierr, Trans1=1;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    SRIniM(myrank, 0.0);              /* init SR4M */
    A[myrank]=SR4Dg4M();     /* get a random number */
    if (myrank==0)                    /* receive data */
        {
        for (RecvNo=1; RecvNo<mpisize ; RecvNo++)
            {
            mpierr=MPI_Recv(&A[RecvNo], Trans1, MPI_INT,
                        RecvNo, 1234, MPI_COMM_WORLD, &mpistat);
            if (mpierr) {printf("mpi recv err : %d¥n", RecvNo);}
            }
        }
      else                            /* send data */
        {
        mpierr=MPI_Send(&A[myrank], Trans1, MPI_INT,
                                    0, 1234, MPI_COMM_WORLD);
        if (mpierr) {printf("mpi send err : %d¥n", myrank);}
        }

   MPI_Finalize();
   if (myrank==0)                     /* display data */
       {
       for (RecvNo=0; RecvNo<mpisize ; RecvNo++)
           {printf("process:%d  random#:%04d¥n", RecvNo, A[RecvNo]);}
       }
   return(0);
   }
```

このプログラムの並列計算は次のように行われる：



(SR/4M) の乱数生成速度は，(SR/4) と同等であるが，倍精度浮動小数点演算に相当する計算を，Hitachi SR8000 の６４ビット整数計算でエミュレートした (SI/4) (今回の結果を得た方法) では，(SR/4)よりスピードが速くなる。[2]の結果と組み合わせた速度の比は以下のようになる：

$$\begin{aligned}&(\mathrm{FSR}):(\mathrm{MT}):(\mathrm{Ph}):(\mathrm{BC}):(\mathrm{SR4}):(\mathrm{SI4})\\ =\ &0.89:1.00:4.40:0.32:9.86:5.42\ .\end{aligned}$$

以上の内容をまとめれば，(SR/4M)は生成速度は少し遅いが，並列計算用として使い易い乱数生成法ということができよう。

参考文献

[1] Yaguchi, H.: Construction of a long-period nonalgebraic and nonrecursive pseudorandom number generator.

[2] Rokko Lectures in Mathematics No.8.「数値計算誤差と乱数生成」神戸大学理学部数学教室(2001)

[3] 湯浅，安村，中田 編：はじめての並列プログラミング．共立出版 (1999)

[4] Wilkinson, B. and Allen, M.: Parallel programming. Prentice-Hall (1999) (飯塚，緑川 訳：並列プログラミング入門．丸善 (2000))