A uniform real random number generator obeying the IEEE 754 format using an affine transition

Mutsuo Saito [1]
Makoto Matsumoto [1]

[1]Hiroshima University

July 6-11, MCQMC'08

This study is granted in part by JSPS Grant-In-Aid #19204002, #18654021 and JSPS Core-to-Core Program No.18005.

Introduction

Request: Most scientific Monte-Carlo simulation requires great deal of floating point pseudorandom numbers.

Introduction

Request: Most scientific Monte-Carlo simulation requires great deal of floating point pseudorandom numbers.

Answer: We propose a fast and high quality uniform double precision floating point pseudorandom number generator (PNG).

Introduction

Request: Most scientific Monte-Carlo simulation requires great deal of floating point pseudorandom numbers.

Answer: We propose a fast and high quality uniform double precision floating point pseudorandom number generator (PNG).

There are some methods to generate floating point pseudorandom numbers.

Introduction

Request: Most scientific Monte-Carlo simulation requires great deal of floating point pseudorandom numbers.

Answer: We propose a fast and high quality uniform double precision floating point pseudorandom number generator (PNG).

There are some methods to generate floating point pseudorandom numbers.

1. generate an integer number and convert it into a floating point number by dividing or multiplying a constant.

Introduction

Request: Most scientific Monte-Carlo simulation requires great deal of floating point pseudorandom numbers.

Answer: We propose a fast and high quality uniform double precision floating point pseudorandom number generator (PNG).

There are some methods to generate floating point pseudorandom numbers.

1. generate an integer number and convert it into a floating point number by dividing or multiplying a constant.
2. generate an integer number and convert it into a bit pattern which represent a floating point number using bit operations.

Introduction

Request: Most scientific Monte-Carlo simulation requires great deal of floating point pseudorandom numbers.

Answer: We propose a fast and high quality uniform double precision floating point pseudorandom number generator (PNG).

There are some methods to generate floating point pseudorandom numbers.

1. generate an integer number and convert it into a floating point number by dividing or multiplying a constant.
2. generate an integer number and convert it into a bit pattern which represent a floating point number using bit operations.

We propose a new method to generate floating point pseudorandom numbers.

Introduction

Our idea is simple:

Introduction

Our idea is simple:

- generate pseudorandom 52-bit patterns

Introduction

Our idea is simple:

- generate pseudorandom 52-bit patterns
- supply the most significant 12-bits with a special constant,

Introduction

Our idea is simple:

- generate pseudorandom 52-bit patterns
- supply the most significant 12-bits with a special constant,
- so that they represent floating point numbers in the range [1, 2) obeying the IEEE 754 format in memory.

Introduction

Our idea is simple:

- generate pseudorandom 52-bit patterns
- supply the most significant 12-bits with a special constant,
- so that they represent floating point numbers in the range [1, 2) obeying the IEEE 754 format in memory.

We also used 128-bit SIMD (Single Instruction Multiple Data) instruction set for faster generation. In other words, our PNG generates two floating point numbers at one time.

Introduction

Our idea is simple:

- generate pseudorandom 52-bit patterns
- supply the most significant 12-bits with a special constant,
- so that they represent floating point numbers in the range [1, 2) obeying the IEEE 754 format in memory.

We also used 128-bit SIMD (Single Instruction Multiple Data) instruction set for faster generation. In other words, our PNG generates two floating point numbers at one time.
We have made a new generator, double precision SIMD-oriented Fast Mersenne Twister (dSFMT), which is much faster than

- Mersenne Twister (Matsumoto and Nishimura '98) (MT) and
- SIMD-oriented Fast MT (Saito and Matsumoto '07) (SFMT)

in generating double precision numbers.

IEEE 754

IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) is the most widely-used standard for floating-point.

The standard defines

- single precision (32 bit)
- extended single precision (more than 43 bit)
- double precision (64 bit)
- extended double precision (more than 79 bit)

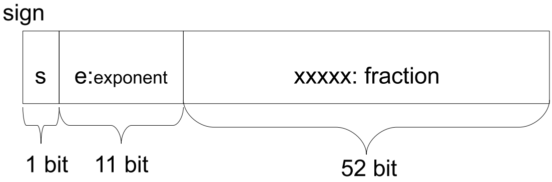
IEEE 754

IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) is the most widely-used standard for floating-point.

The standard defines

- single precision (32 bit)
- extended single precision (more than 43 bit)
- double precision (64 bit)
- extended double precision (more than 79 bit)

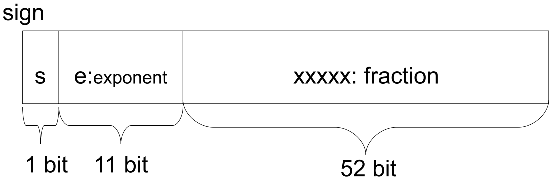
IEEE 754 double precision format



	exponent	fraction	represented number
zero	0	0	± 0
denormalized number	0	$\neq 0$	$\pm 0.xxxx \times 2^{-1022}$
∞	2047	0	$\pm\infty$
NaN	2047	$\neq 0$	Not a Number
normalized number	other	any	$\pm 1.xxxxx \times 2^{e-1023}$

xxxx shows the bit pattern of fraction part.

IEEE 754 double precision format



	exponent	fraction	represented number
zero	0	0	± 0
denormalized number	0	$\neq 0$	$\pm 0.xxxx \times 2^{-1022}$
∞	2047	0	$\pm\infty$
NaN	2047	$\neq 0$	Not a Number
normalized number	other	any	$\pm 1.xxxxx \times 2^{e-1023}$

xxxx shows the bit pattern of fraction part.

If s is 0 and e is 0x3ff, then the format represent a normalized number in the range [1, 2).

Linear Feedbacked Shift Register (LFSR)

Definition

- A bit $\{0, 1\}$ is identified with $\mathbb{F}_2$, the two element field.
- b-bit integers are identified with horizontal vectors in $\mathbb{F}_2^b$.
  b is 64 or 128.
- We consider an array of N b-bit in computer memory as the vector space $(\mathbb{F}_2^b)^N$.

Linear Feedbacked Shift Register (LFSR)

Definition

- A bit $\{0,1\}$ is identified with $\mathbb{F}_2$, the two element field.
- b-bit integers are identified with horizontal vectors in $\mathbb{F}_2^b$.
  b is 64 or 128.
- We consider an array of N b-bit in computer memory
  as the vector space $(\mathbb{F}_2^b)^N$.

Definition

Linear Feedbacked Shift Register (LFSR) is defined by
a recursion formula of rank N:

$$\mathbf{w}_i = g(\mathbf{w}_{i-N}, \ldots, \mathbf{w}_{i-1}),$$

where g is an $\mathbb{F}_2$-linear map $(\mathbb{F}_2^b)^N \to \mathbb{F}_2^b$ and $\mathbf{w}_i \in \mathbb{F}_2^b$.

Pulmonary LFSR

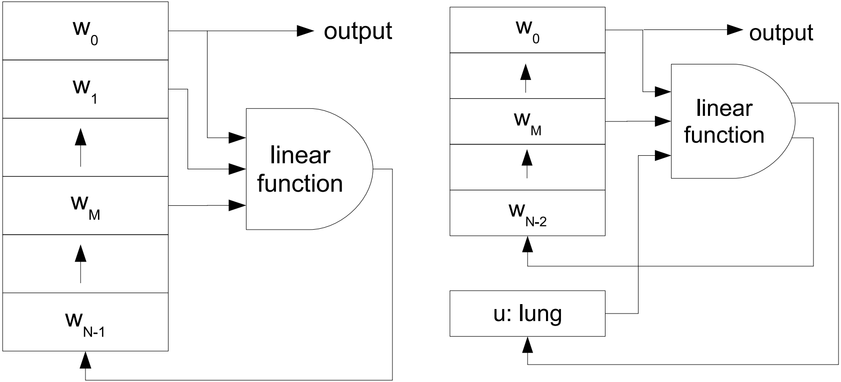
Definition

Pulmonary LFSR is a variant of LFSR, which is defined by two recursion formulas:

$$\begin{aligned}\mathbf{w}_i &= g(\mathbf{w}_{i-N+1}, \ldots, \mathbf{w}_{i-1}, \mathbf{u}_{i-1}),\\ \mathbf{u}_i &= h(\mathbf{w}_{i-N+1}, \ldots, \mathbf{w}_{i-1}, \mathbf{u}_{i-1}).\end{aligned}$$

where g and h are $\mathbb{F}_2$-linear maps $(\mathbb{F}_2^b)^N \to \mathbb{F}_2^b$ and $\mathbf{w}_i, \mathbf{u}_i \in \mathbb{F}_2^b$.

LFSR and Pulmonary LFSR



Standard LFSR (e.g. MT) and Pulmonary LFSR

Detailed description of dSFMTv2

We released dSFMT based on this idea in 2007 from our web page.
Here we propose an improved version of dSFMT.
dSFMTv2 is a pulmonary LFSR, whose recursion formulas are:

$$\begin{aligned} \mathbf{u}_i &= A\mathbf{w}_{i-N+1} + \mathbf{w}_{i-N+M+1} + B\mathbf{u}_{i-1}, \\ \mathbf{w}_i &= \mathbf{w}_{i-N+1} + D\mathbf{u}_i, \end{aligned}$$

where $\mathbf{w}_0, ..., \mathbf{w}_{N-2} \in \mathbb{F}_2^{128}$, $A, B, D \in M_{128}(\mathbb{F}_2)$. M is pick up position $0 < M < N - 2$.

Detailed description of dSFMTv2

We released dSFMT based on this idea in 2007 from our web page.
Here we propose an improved version of dSFMT.
dSFMTv2 is a pulmonary LFSR, whose recursion formulas are:

$$\begin{aligned} \mathbf{u}_i &= A\mathbf{w}_{i-N+1} + \mathbf{w}_{i-N+M+1} + B\mathbf{u}_{i-1}, \\ \mathbf{w}_i &= \mathbf{w}_{i-N+1} + D\mathbf{u}_i, \end{aligned}$$

where $\mathbf{w}_0, \ldots, \mathbf{w}_{N-2} \in \mathbb{F}_2^{128}$, $A, B, D \in M_{128}(\mathbb{F}_2)$. M is pick up position $0 < M < N - 2$.

1. $\mathbf{w}_0, \ldots, \mathbf{w}_{N-2}$ are set to the values in [1, 2) with the format IEEE 754
2. D is chosen appropriately,

so that the consecutive $\mathbf{w}_i$s are uniformly distributed in the range [1, 2).

Detailed description of dSFMTv2

Because 24 bits of each **w** is fixed, we can consider above formula as an affine recursion formula:

$$
\begin{aligned}
\mathbf{y}_i &= F\mathbf{x}_{i-N+1} + \mathbf{x}_{i-N+M+1} + G\mathbf{y}_{i-1} + c \\
\mathbf{x}_i &= \mathbf{x}_{i-N+1} + H\mathbf{y}_i,
\end{aligned}
$$

where $\mathbf{y}_i, c \in \mathbb{F}_2^{128}$, $\mathbf{x}_i \in \mathbb{F}_2^{104}$, $F, G \in M_{128,104}(\mathbb{F}_2)$, $H \in M_{104,128}(\mathbb{F}_2)$.
To assure the period and distriution property, we need to develop algorithms to compute theses for affine transforation generalized those for linear transformation. (we ommit)

Diagram of dSFMTv2

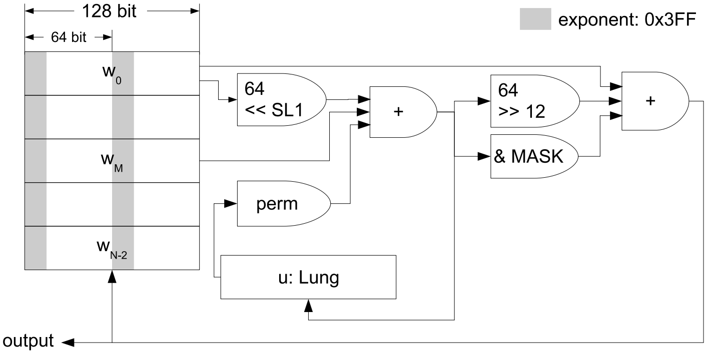


Diagram of dSFMTv2

Diagram of dSFMTv2

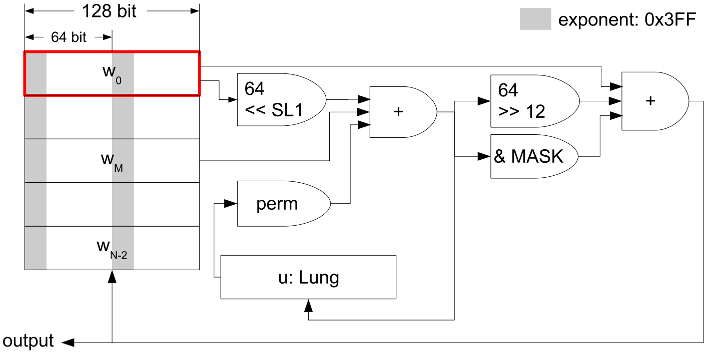
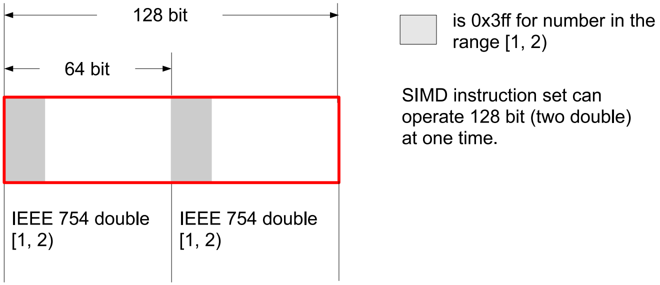


Diagram of dSFMTv2

Diagram of dSFMTv2



SIMD instruction set can operate 128 bit (two double) at one time.

Diagram of dSFMTv2

Diagram of dSFMTv2

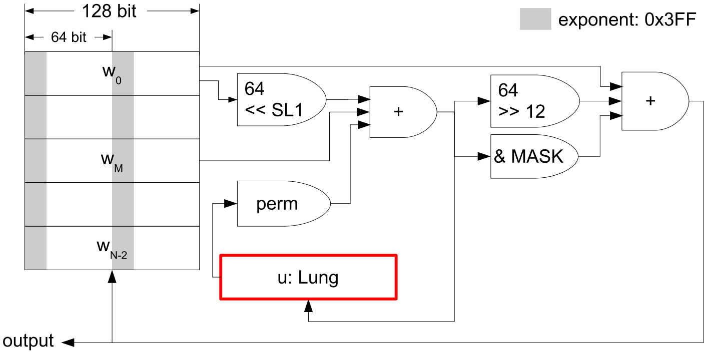


Diagram of dSFMTv2

Diagram of dSFMTv2

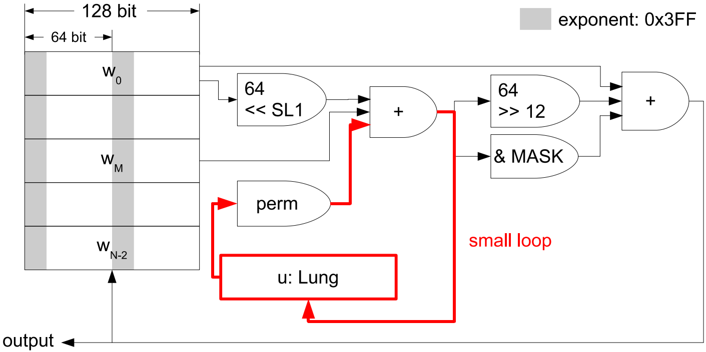


Diagram of dSFMTv2

Diagram of dSFMTv2

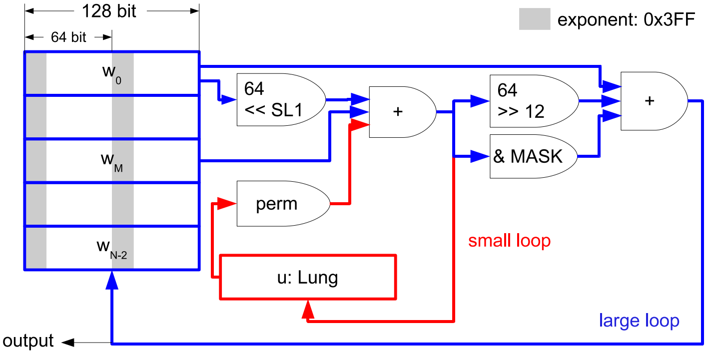


Diagram of dSFMTv2

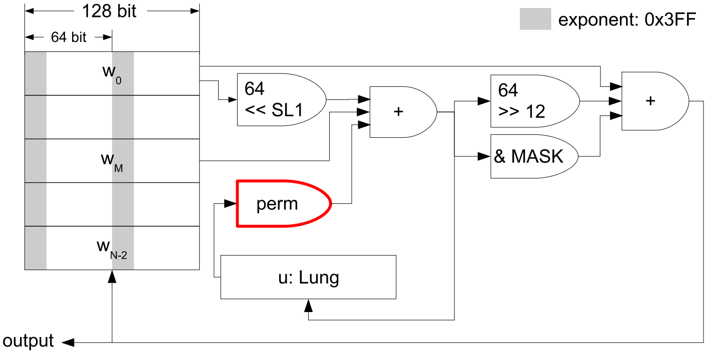


Diagram of dSFMTv2

Diagram of dSFMTv2

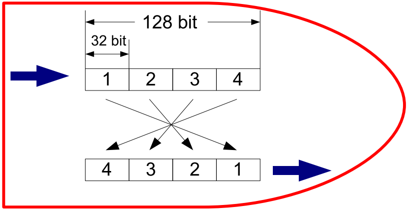


Diagram of dSFMTv2

Diagram of dSFMTv2

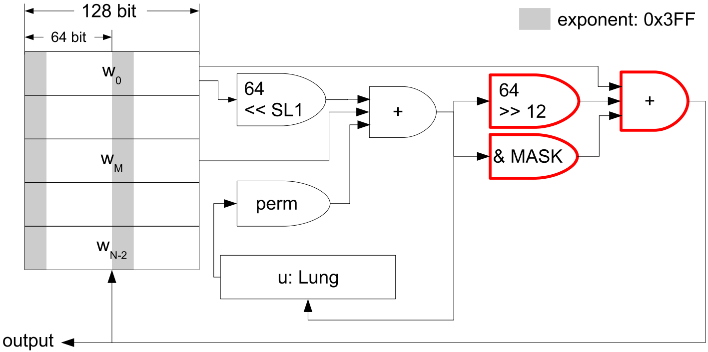


Diagram of dSFMTv2

Diagram of dSFMTv2

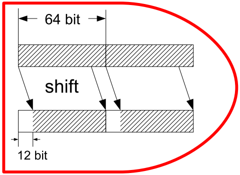


Diagram of dSFMTv2

Diagram of dSFMTv2

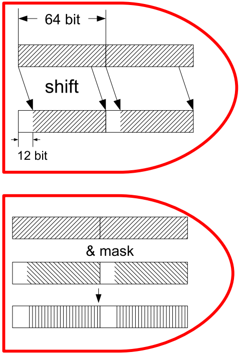


Diagram of dSFMTv2

Diagram of dSFMTv2

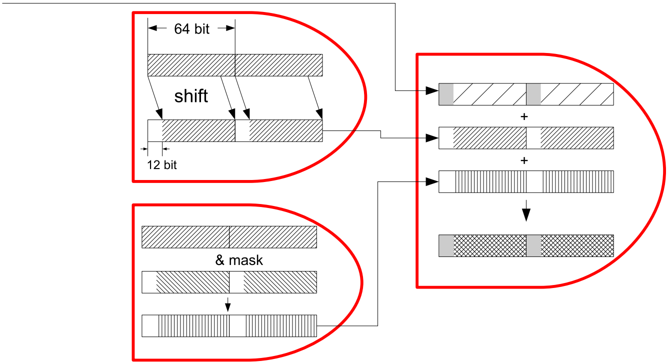


Diagram of dSFMTv2

Diagram of dSFMTv2

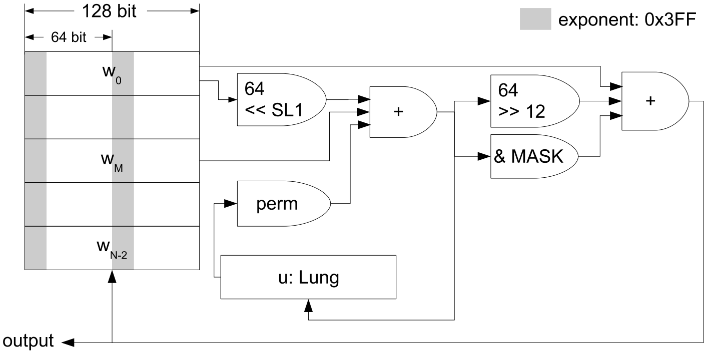


Diagram of dSFMTv2

dSFMTv2-19937

dSFMTv2-19937 has following specification.

- the least period: $2^{19937} - 1$
- size of array N: 192
- state space of affine transition: $\mathbb{F}_2^{19992}$
- shift value SL1: 19
- pick up position M: 117
- constant mask: 0x000ffafffffffb3f000ffdfffc90fffd

Dimension of equidistribution

Definition

A periodic sequence with period P

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

of v-bit integers is said to be k-dimensionally equidistributed if any kv-bit pattern occurs equally often as a k-tuple

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

for a period $i = 0, \ldots, P-1$.

Dimension of equidistribution

Definition

A periodic sequence with period P

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

of v-bit integers is said to be k-dimensionally equidistributed if any kv-bit pattern occurs equally often as a k-tuple

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

for a period $i = 0, \ldots, P-1$.
(The all-zero pattern occurs once less often than the others.)

Dimension of equidistribution

Definition

A periodic sequence of b-bit integers is said to be k-dimensionally equidistributed with v-bit accuracy if the most significant $v(< b)$ bits of each integer are k-dimensionally equidistributed.
We denote by $k(v)$ the maximum such k.

Dimension of equidistribution

Definition

A periodic sequence of b-bit integers is said to be k-dimensionally equidistributed with v-bit accuracy if the most significant $v(< b)$ bits of each integer are k-dimensionally equidistributed.
We denote by $k(v)$ the maximum such k. We have an upper bound of the sequence with period P

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

Dimension of equidistribution

Definition

A periodic sequence of b-bit integers is said to be k-dimensionally equidistributed with v-bit accuracy if the most significant $v(< b)$ bits of each integer are k-dimensionally equidistributed.
We denote by $k(v)$ the maximum such k. We have an upper bound of the sequence with period P

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

and define the dimension defect $d(v)$ at v as the gap between the bound and the realized dimension of equidistribution:

$$d(v) := \lfloor \log_2(P+1)/v \rfloor - k(v),$$

Dimension of equidistribution

Definition

A periodic sequence of b-bit integers is said to be k-dimensionally equidistributed with v-bit accuracy if the most significant $v(< b)$ bits of each integer are k-dimensionally equidistributed.
We denote by $k(v)$ the maximum such k. We have an upper bound of the sequence with period P

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

and define the dimension defect $d(v)$ at v as the gap between the bound and the realized dimension of equidistribution:

$$d(v) := \lfloor \log_2(P+1)/v \rfloor - k(v),$$

and the total dimension defect Δ as the sum of these gaps.

Dimension of equidistribution

$k(v)$ and $d(v)$ of 52-bit fraction part of dSFMTv2-19937.

v	$k(v)$	$d(v)$	v	$k(v)$	$d(v)$	v	$k(v)$	$d(v)$	v	$k(v)$	$d(v)$
1	19937	0	14	1423	1	27	734	4	40	383	115
2	9967	1	15	1328	1	28	702	10	41	383	103
3	6644	1	16	1245	1	29	620	67	42	383	91
4	4983	1	17	1172	0	30	538	126	43	383	80
5	3987	0	18	1107	0	31	536	107	44	383	70
6	3322	0	19	1049	0	32	535	88	45	383	60
7	2847	1	20	996	0	33	384	220	46	383	50
8	2491	1	21	949	0	34	384	202	47	383	41
9	2215	0	22	772	134	35	384	185	48	383	32
10	1993	0	23	772	94	36	384	169	49	383	23
11	1812	0	24	772	58	37	383	155	50	383	15
12	1661	0	25	772	25	38	383	141	51	383	7
13	1533	0	26	766	0	39	383	128	52	383	0

Δ is 2608.

c.f. Δ of MT19937 is 6750.

Comparison of speed

Generators

- dSFMTv2: dSFMT ver. 2, improved dSFMT, described in this talk.
- dSFMTv1: dSFMT unpublished, but the code is available from our homepage.
- MT mask: Mersenne Twister with bit mask to fit to IEEE 754.
- MT 64 mask: 64-bit MT with bit mask to fit to IEEE 754.
- SFMT mask: SFMT with bit mask to fit to IEEE 754.
- SFMT $\times$ const: SFMT standard double generation.

Comparison of speed

Generators

- dSFMTv2: dSFMT ver. 2, improved dSFMT, described in this talk.
- dSFMTv1: dSFMT unpublished, but the code is available from our homepage.
- MT mask: Mersenne Twister with bit mask to fit to IEEE 754.
- MT 64 mask: 64-bit MT with bit mask to fit to IEEE 754.
- SFMT mask: SFMT with bit mask to fit to IEEE 754.
- SFMT $\times$ const: SFMT standard double generation.

CPUs and compilers

- Pentium M 1.4GHz and Intel C compiler (ICC)
- Pentium 4 3GHz and ICC
- core 2 duo 1.83GHz and ICC
- Athlon 64 2.4GHz and GNU C Compiler (GCC)
- Power PC G4 1.33GHz and GCC

Comparison of speed

Output

- blk: Generate 50000 of double precision floating point numbers in an array, at one call.
  This is iterated for 2000 times (10^8 generations).
- seq: Generate 10^8 of double precision floating point numbers sequentially, one by one.

All outputs are formatted in the range [0, 1).
(i.e. outputs of dSFMTs are subtracted by 1.0).

Comparison of speed

Using SIMD instruction set.
The time (sec.) required for 10^8 of double float generations.

		dSFMTv2 (new)	dSFMTv1 (old)	MT mask	SFMT mask	SFMT $\times$ const
Pentium M	blk	0.626	0.867	1.526	0.928	2.636
1.4 Ghz	seq	1.422	1.761	3.181	2.342	3.671
Pentium 4	blk	0.254	0.640	0.987	0.615	3.537
3 Ghz	seq	0.692	1.148	3.339	3.040	3.746
core 2 duo	blk	0.199	0.381	0.705	0.336	0.532
1.83GHz	seq	0.380	0.457	1.817	1.317	2.161
Athlon 64	blk	0.362	0.637	1.117	0.623	1.278
2.4GHz	seq	0.680	0.816	1.637	0.763	1.623
PowerPC G4	blk	0.887	1.151	2.175	1.657	8.897
1.33GHz	seq	1.212	1.401	5.624	2.994	7.712

Comparison of speed

Using SIMD instruction set.
The time (sec.) required for 10^8 of double float generations.

		dSFMTv2 (new)	dSFMTv1 (old)	MT mask	SFMT mask	SFMT $\times$ const
Pentium M	blk	0.626	0.867	1.526	0.928	2.636
1.4 Ghz	seq	1.422	1.761	3.181	2.342	3.671
Pentium 4	blk	0.254	0.640	0.987	0.615	3.537
3 Ghz	seq	0.692	1.148	3.339	3.040	3.746
core 2 duo	blk	0.199	0.381	0.705	0.336	0.532
1.83GHz	seq	0.380	0.457	1.817	1.317	2.161
Athlon 64	blk	0.362	0.637	1.117	0.623	1.278
2.4GHz	seq	0.680	0.816	1.637	0.763	1.623
PowerPC G4	blk	0.887	1.151	2.175	1.657	8.897
1.33GHz	seq	1.212	1.401	5.624	2.994	7.712

Comparison of speed

without using SIMD instruction set.

The time (sec.) required for 10^8 of double float generations

		dSFMTv2 (new)	dSFMTv1 (old)	MT64 mask	MT mask	SFMT mask	SFMT × const
Pentium M	blk	1.345	2.023	2.031	3.002	2.026	3.355
1.4 Ghz	seq	2.004	2.386	2.579	3.308	2.835	3.910
Pentium 4	blk	1.079	1.128	1.432	2.515	1.929	3.762
3 Ghz	seq	1.431	1.673	3.137	3.534	3.485	4.331
core 2 duo	blk	0.899	1.382	1.359	2.404	1.883	1.418
1.83GHz	seq	0.777	1.368	1.794	1.997	1.925	2.716
Athlon 64	blk	0.334	0.765	0.820	1.896	1.157	1.677
2.4GHz	seq	0.567	0.970	1.046	2.134	1.129	2.023
PowerPC G4	blk	1.834	3.567	2.297	4.326	4.521	12.685
1.33GHz	seq	1.960	2.865	4.090	5.489	5.464	9.110

Comparison of speed

without using SIMD instruction set.
The time (sec.) required for 10^8 of double float generations

		dSFMTv2 (new)	dSFMTv1 (old)	MT64 mask	MT mask	SFMT mask	SFMT × const
Pentium M	blk	1.345	2.023	2.031	3.002	2.026	3.355
1.4 Ghz	seq	2.004	2.386	2.579	3.308	2.835	3.910
Pentium 4	blk	1.079	1.128	1.432	2.515	1.929	3.762
3 Ghz	seq	1.431	1.673	3.137	3.534	3.485	4.331
core 2 duo	blk	0.899	1.382	1.359	2.404	1.883	1.418
1.83GHz	seq	0.777	1.368	1.794	1.997	1.925	2.716
Athlon 64	blk	0.334	0.765	0.820	1.896	1.157	1.677
2.4GHz	seq	0.567	0.970	1.046	2.134	1.129	2.023
PowerPC G4	blk	1.834	3.567	2.297	4.326	4.521	12.685
1.33GHz	seq	1.960	2.865	4.090	5.489	5.464	9.110

Conclusion

We proposed dSFMT pseudorandom number generator.

Conclusion

We proposed dSFMT pseudorandom number generator.

- dSFMTv2-19937 generates IEEE 754 floating point numbers directly.

Conclusion

We proposed dSFMT pseudorandom number generator.

- dSFMTv2-19937 generates IEEE 754 floating point numbers directly.
- It has a period at least $2^{19937} - 1$.

Conclusion

We proposed dSFMT pseudorandom number generator.

- dSFMTv2-19937 generates IEEE 754 floating point numbers directly.
- It has a period at least $2^{19937} - 1$.
- It has good equidistribution property.

Conclusion

We proposed dSFMT pseudorandom number generator.

- dSFMTv2-19937 generates IEEE 754 floating point numbers directly.
- It has a period at least $2^{19937} - 1$.
- It has good equidistribution property.
- It is much faster than MT and SFMT in double precision floating point generation.

Conclusion

We proposed dSFMT pseudorandom number generator.

- dSFMTv2-19937 generates IEEE 754 floating point numbers directly.
- It has a period at least $2^{19937} - 1$.
- It has good equidistribution property.
- It is much faster than MT and SFMT in double precision floating point generation.

Remark:

Someone may think cancellation error will occur when we convert numbers form the range [1, 2) to [0, 1).

Conclusion

We proposed dSFMT pseudorandom number generator.

- dSFMTv2-19937 generates IEEE 754 floating point numbers directly.
- It has a period at least $2^{19937} - 1$.
- It has good equidistribution property.
- It is much faster than MT and SFMT in double precision floating point generation.

Remark:

Someone may think cancellation error will occur when we convert numbers form the range [1, 2) to [0, 1).

This is negligible: The generated numbers in the range [0, 1) by our method have the same accuracy as ones obtaind by dividing 52-bit integers by a constant.

Thank you for your kind attention.